Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

Marko Seppänen

TIETOJÄRJESTELMÄN TOTEUTTAMINEN SMARTGWT-SOVELLUSKEHYKSELLÄ

Opinnäytetyö 2010

TIIVISTELMÄ

Marko Seppänen

Tietojärjestelmän toteuttaminen SmartGWT-sovelluskehyksellä, 147 sivua, 1 liite
Saimaan ammattikorkeakoulu, Lappeenranta
Tekniikka, Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto
Opinnäytetyö, 2010
Ohjaajat: IT-koordinaattori Simo Horsmanheimo ja lehtori Martti Ylä-Jussila

Tässä opinnäytetyössä on asiakkaan toiminnallisten vaatimusten mukainen tietojärjestelmä toteuttamalla analysoitu Java-pohjaisen SmartGWT-sovelluskehyksen soveltuvuutta ja vaikutusta järjestelmän kehitystyöhön. Yleisinä kelpoisuuskriteereinä tarkasteltiin muun muassa ohjelmointimukavuutta, testattavuutta, mallinnettavuutta ja johdonmukaisuutta sekä yhteentoimivuutta muiden teknologioiden ja ohjelmistojen kanssa.

Osoittautui, että SmartGWT integroituu hyvin Eclipse-kehitysympäristöön, paikallinen debuggaus on ongelmatonta, testattavuus on hyvä, sovellusten selainriippumattomuus saavutettiin ja SmartGWT toimii hyvin yhteen liitännäisteknologioiden kanssa. Muihin tietojärjestelmiin liittymisiin ei asettuisi SmartGWT:stä johtuvia rajoitteita.

Ongelmaksi muodostui pitkälle kehitetyn sovelluksen muokattavuus ja sovelluksen takaisinmallinnus koodista kaavioiksi, mikä johtuu varsinaisesti kolmansien osapuolien ohjelmien rajoitteista. Useamman kehittäjän tiimissä tämä olisi ongelma, sillä se heijastuu dokumentaation päivitystahtiin. Järjestelmää kehitettiin muunnellun vesiputousmallin mukaisesti; suunnittelu- ja toteutusvaiheiden dokumentteja paranneltiin jatkuvasti.

Käyttöliittymien suunnittelussa oli mahdollista hyödyntää kolmannen osapuolen visuaalista editoria, mutta käytännössä vain prototyypittelytarpeissa, sillä vaativampaan refaktorointiin se ei soveltunut. SmartGWT:n valmiskomponentit ovat riittävän monipuolisia, mutta niiden etädebuggausta varten sovelluskehittäjä joutuu kehittämään omat ratkaisunsa. SmartGWT soveltuu pienimuotoisen tietojärjestelmän toteuttamiseen, riittävästi perehtymisaikaa varaamalla ja sen omat sekä kolmansista osapuolista johtuvat rajoitukset tiedostamalla.

asiasanat: smartgwt, java, google web toolkit, lomake, käyttöliittymä, uml, sovelluskehys, tietojärjestelmä, testaus, takaisinmallinnus, kolmitasorakenne, etämetodi, sql, lgpl

ABSTRACT
Marko Seppänen
Developing an information system by using SmartGWT application framework, 147 pages, 1 appendix
Saimaa University of Applied Sciences, Lappeenranta
Technology, Degree Programme in Information Technology
Software engineering
Bachelor's thesis, 2010
Instructors: IT-coordinator Simo Horsmanheimo and lecturer Martti Ylä-Jussila

It has been analyzed in this thesis, by implementing an information system based on the customer's functional requirements, how suitable Java-based application framework SmartGWT is for development of an information system that is to be used online. Among others, the following points were examined as a general eligibility criteria: programming convenience, testability, modelling capabilities and consistency, as well as interoperability with other technologies and software.

It appeared that SmartGWT integrates well with the Eclipse development environment, the local debugging is straightforward, testability is good, browser-independence was achieved and SmartGWT works well together with the related technologies.

Customization of the advanced application code and reverse engineering from code to diagrams emerged as problematic issues, partly due to third-party software constraints. In developer team consisting of many persons this would become a problem, because it is reflected in the documentation update pace. The system was developed in accordance with the modified waterfall model; documents related to design and implementation were improved continually.

In the user interface design it was possible to take advantage of the third-party visual editor, but in practice this applied only while prototyping. It was not approriate for demanding refactoring. SmartGWT's components are diversified enough, but remote debugging them requires the application developer to develop his own solutions. SmartGWT is suitable for small-scale information system implementation, but sufficient time should be allocated to make oneself familiar with it. One should also be aware of the limitations imposed by itself and third parties.

asiasanat: smartgwt, java, google web toolkit, form, user interface, uml, software framework, information system, testing, reverse engineering, 3-tier, remote procedure call, sql, lgpl

SISÄLTÖ

1 JOHDANTO

Tässä opinnäytetyössä oli lähtökohtana selvittää, kuinka hyvin SmartGWT-sovelluskehys soveltuu asiakkaan toiminnallisten vaatimusten mukaisen tukipyyntöjärjestelmän toteuttamiseen, ohjelmistotuotannon käytänteitä noudattaen. Soveltuvuus merkitsee tässä enemmän kuin lopputuotteen käytettävyyttä ja vaadittujen toimintojen toteuttamista. Jotta sovelluskehys sopisi aikarajoitteisen ja projektimuotoisena läpivietävän tietojärjestelmän toteuttamiseen, on sovelluskehyksen täytettävä tiettyjä kriteereitä.

1.1 Kelpoisuuskriteereistä ja muista sovelluskehyksen valintaperusteista

Yleisinä kelpoisuuskriteereinä on tarkasteltu ohjelmointimukavuutta, testattavuutta, mallinnettavuutta, johdonmukaisuutta ja yhteentoimivuutta muiden teknologioiden ja ohjelmistojen kanssa. Dokumentaation, käyttäjäyhteisön aktiivisuus ja tuotetuen kattavuus ovat olleet merkittävänä kriteerinä, kuten myös Eclipse-kehitysympäristön ominaisuuksien hyödynnettävyys SmartGWT-pohjaisen järjestelmän toteuttamisessa.

Käytettyyn sovelluskehykseen perustuvan tietojärjestelmän toiminnallisuuden, käyttöliittymien tai arkkitehtuurin muuttamisen helppous, kehitystyön myöhemmissä vaiheissa, kertoo paljon sovelluskehyksellä toteutettavien järjestelmien mallinnettavuudesta ja erillisiin toiminnallisiin osiin jaoteltavuudesta. Osatekijöihin jakamisen helppoudella on merkitystä myös arvioitaessa aikatauluja ja tehtäessä suunnitelmia henkilöresurssien käytöstä.

Testattavuus on tarkasteltavista kriteereistä vaikeimmin rajattavissa, sillä erilaisia testitapauksia on mahdollista kehittää ääretön määrä. Sovelluskehystä ei ole verrattu miltään osin muihin tarjolla oleviin sovelluskehyksiin, vaan pyrkimyksenä on ollut muodostaa kokonaisvaikutelma siitä, minkälaista oli toteuttaa tukipyyntöjenkäsittelyjärjestelmä SmartGWT:llä.

Primääreinä valintakriteereinä verkossa käytettävän tukipyyntöjenkäsittelyjärjestelmän toteuttamiseksi SmartGWT:llä ovat olleet opinnäytetyön tekijän itsensä

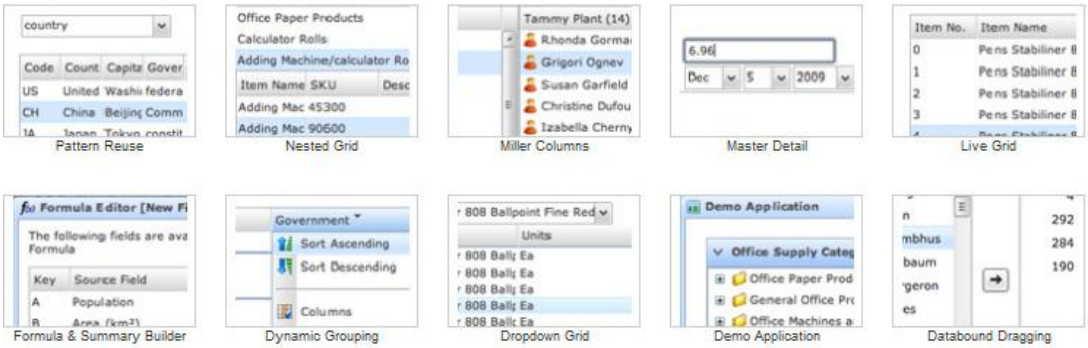
asettamat vaatimukset siitä, että toteutus tehdään käyttäen Java-ohjelmointikieltä ja sovelluskehystä, jossa on runsaasti muokattavia valmiskomponentteja käytettäväksi. Näihin kriteereihin olisivat sopineet myös erilaisia tagikirjastoja hyödyntävät sovelluskehykset kuten ICEFaces tai Backbase, mutta SmartGWT:n valintaa puolsi myös aiempi, ohjelmointiharjoituksesta saatu kokemus toisesta Google Web Toolkit -pohjaisesta ohjelmistokehyksestä, Ext GWT:stä (nykyisin Sencha).

Enterprise-tason alustoista Spring ja Apache Struts 2 vaikuttivat liian laajoilta, jotta niihin olisi ehtinyt syventyä riittävästi ennalta, opinnäytetyölle asetettavan aikataulun puitteissa, joten ennen lopullista päätöstä valittavina olivat SmartGWT ja samantapainen Vaadin. SmartGWT:n Google Web Toolkit –pohjaisuus tarkoitti kuitenkin sitä, että kirjallisuutta ja muuta lähdetietoa (blogi- ja artikkelilähteet erityisesti) olisi runsaasti, joten eri aspektit huomioon ottaen SmartGWT tuntui luontevalta valinnalta.

Opinnäytetyötä aloittaessa avoimina kysymyksinä olivat muun muassa, mutta ei näihin rajoittuen: ohjelmointimukavuus (sisältää dokumentaation ja käyttäjäyhteisön tuen), yhteentoimivuus kehitysympäristön ja muiden teknologioiden kanssa, muodostetun ohjelmistoarkkitehtuurin muunneltavuus kehitystyön aikana, mallinnettavuus UML-kaavioista ohjelmointikoodiksi ja siitä takaisin kaavioiksi, käyttöliittymien käyttötuntuma sekä testattavuuteen liittyvinä järjestelmän suorituskyky ja vasteajat.

1.2 Tarkemmin SmartGWT:stä

SmartGWT:n komponentteja demoavalla sivulla on kaikista käytettävissä olevista komponenteista (poimintoja kuvassa 1) esimerkkejä ohjelmointikoodeineen, mitkä yhdessä kattavaan API-dokumentaatioon tutustumisen ja omakohtaisen komponenttien kokeilun kautta riittivät synnyttämään uskomuksen, että kyseinen sovelluskehys on sovelias tukipyyntöjenkäsittelyjärjestelmän toteuttamiseen. Kattavaa esikokeilua ei ollut mahdollista tehdä, koska tiettyjen olennaisten komponenttien ominaisuuksista muodostuisi liian monia yhdistelmiä läpikäytäväksi. Lisäksi tiettyjä komponentteja saattoi sisällyttää toisiinsa (esimerkkinä ruudukon soluun sijoitettu tekstieditori).



Kuva 1. Poimintoja SmartGWT:n toiminnallisuutta esittelevältä demosivulta

Järjestelmän toteutukseen vaikutti paljonkin se, että SmartGWT-sovelluskehyksen lisenssiksi valittiin maksullisen editionin sijaan ilmainen LGPL-lisenssillinen editio. Sen koettiin riittävän projektin tarpeisiin, mutta sen valinta heijastui SmartGWT:n komponenteille ominaiseen datalähteeseen kytkeytyvyys -ominaisuuden käyttämättä jättämiseen. SmartGWT:ssä on pro-versiosta alkaen mahdollista kytkeä esimerkiksi kalenteri- tai ruudukkokomponentti suoraan tietokantaan (ks. Taulukko 1), jolloin esimerkiksi tiedonhakua ja tiedon talletusta varten ei olisi tarpeen ohjelmoida erillistä kontrollilogiikkaa (Isomorphic Software 2010) näitä tarkoituksia varten.

Taulukko 1. Poimintoja SmartGWT:n editioiden eroista

OMINAISUUS / EDITIO	**LGPL**	**Pro**	**Power**	**Enterprise**
Server Data Binding	-	Kyllä	kyllä	Kyllä
Server Validation	-	Kyllä	Kyllä	Kyllä
SQL Connector	-	Kyllä	Kyllä	Kyllä
Hibernate Connector	-	Kyllä	Kyllä	Kyllä
HTTP Proxying	-	Kyllä	Kyllä	Kyllä
XPath Java Binding	-	Kyllä	Kyllä	Kyllä
Excel Export	-	Kyllä	Kyllä	Kyllä
Visual Builder	-	kyllä	Kyllä	Kyllä
SQL / HQL Templating	-	-	Kyllä	Kyllä
HINTA	ilmainen	$745 / kehittäjä	$1950 / kehittäjä	neuvoteltavissa

LGPL-lisenssisessä editiossa mahdollisuus rajautuu datalähteisiin, joina ovat joko XML-tiedosto tai tekstitiedosto. Niiden käyttö tukipyyntöjenkäsittelyjärjes-

telmän tapauksessa olisi kuitenkin aiheuttanut tarpeetonta tiedon konvertointia tietorakenteesta toiseen, sillä käsitelty data sijaitsisi tietokannassa.

Komponentteja tarkemmin tutkimalla kävi kuitenkin ilmeiseksi, että niiden muita ominaisuuksia voi käyttää täydessä laajuudessaan ilman suoraa datalähteeseen kytkeytymistäkin. Päätöksenä ominaisuuden poisjättäminen oli harmistuttava sen vuoksi, että datalähteisiin kytkeytyvyys on SmartGWT-arkkitehtuurin peruspilareita (Jivan 2008).

Tällaisessa sovelluskehyksessä kaikki ohjelmointikoodi kirjoitetaan Java-ohjelmointikielellä, mutta vietäessä lähdekoodi automaattitoimintaisen kääntäjän läpi, muodostuu lopputuloksena ajettava sovellus, jonka palvelinpuolen koodi on edelleen Javaa, mutta asiakaspuolella (selaimen puoli) ajettava ohjelmointikoodi on konvertoitu JavaScriptiksi.

SmartGWT:n etuihin kuuluu se, että sitä käytettäessä ei ole välttämätöntä osata sivunkuvauskieli HTML:ää tai muodostaa CSS-tyylisivuja komponenttien ulkoasun ja asettelun määrittelemiseksi – tarvittaessa valmiskomponenttien ulkoasun muokkaaminen CSS:llä on mahdollista. Toisin sanoen kerran ohjelmoituna sovellus toimii kaikissa yleisimmissä selaimissa yhtäläisesti (lähtöolettamus).

1.3 Opinnäytetyön alkuperäinen näkökulma

Alkuperäisenä tarkoituksena oli kehittää käyttöönottovalmis, verkossa käytettävä tukipyyntöjen käsittelyjärjestelmä Etelä-Karjalan koulutuskuntayhtymäkonserniin (EKKY) kuuluvien Etelä-Karjalan ammattiopiston ja Etelä-Karjalan aikuisopisto AKTIVAn yhteiselle, IT-palveluita tuottavalle osastolle. Kuntayhtymään kuuluu yhdeksän jäsenkuntaa: Imatra, Lappeenranta, Lemi, Luumäki, Parikkala, Rautjärvi, Ruokolahti, Savitaipale ja Taipalsaari.

Ehdotus on alun perin ollut muutaman päivän ajan esillä Saimaan ammattikorkeakoulun tietotekniikan insinööriopiskelijoiden yhteisellä harjoittelupaikoista ja opinnäytetöistä kertovalla foorumilla Moodlessa, huhtikuun lopulla (vuonna 2010). Pian ehdotuksen havaitsemisen jälkeen suoritettu yhteydenotto asiakkaaseen johti asiakkaan muiden työtehtävien järjestelyjen vuoksi ensimmäiseen tapaamiseen vasta toukokuun lopulla. Projektin käynnistymisestä kerrotaan li-

sää alaluvussa 6.1 ("Yhteistoiminnan toimivuudesta ja viestinnästä"). Yhteyshenkilö vaihtui kertaalleen ja voidaankin sanoa, että projekti käynnistyi todella vasta heinäkuun alussa.

Projektisuunnitelmaan tehtiin 14.9.2010 pidetyssä ohjauspalaverissa käsitelty ja hyväksytty muutos, jonka nojalla järjestelmän mahdollinen jatkokehitys saatetaan sijoittaa Projektityö-kurssiin puitteisiin ja samalla opinnäytetyön näkökulmaa hieman muutettiin. Asiaan vaikutti jossakin määrin myös konsensuksen saavuttamattomuus tekijänoikeussopimukseen liittyvistä pykälistä (ks. alaluku 6.5, "Tekijänoikeussopimuksen allekirjoittamista").

Opinnäytetyön näkökulmaksi vaihdettiin "sen koetteleminen kuinka hyvin SmartGWT-sovelluskehys sopii helpdesk-järjestelmän version 1.0 toteuttamiseen". Tämä tapahtui siinä vaiheessa, kun toiminnoista lähinnä raportointi oli enää toteuttamatta. Pieniksi luonnehdittavia muutosehdotuksia oli kehitystyön aikana hyväksytty mukaan lukuisin määrin, mutta ehdotuksia muutoksiksi oli kerääntynyt näkemyksien jalostuessa enemmän kuin oli käytännössä mahdollista toteuttaa opinnäytetyön rajoissa.

Projektia tällä tavoin jatkaen voitiin asettaa uudelleen tarkasteltavaksi kokonaisia osa-alueita, jotka karsittiin määrittelyvaiheessa pois, kuten knowledge-base, johon kerrytettäisiin faktatietoa muun muassa erilaisista laitteista. Lisäksi koska asiakkaalla ei varsinaisesti missään vaiheessa ollut erityistä akuuttia kiirettä saada järjestelmää käyttöönsä johonkin tiettyyn ajankohtaan mennessä, oli hyödyllistä antaa opinnäytetyön tekijän kehittyä lisää testauksen hallitsemisessa ja testausvälineiden käyttämisessä. Erilaisia testausohjelmia, plugineita Eclipse-ohjelmistokehitysympäristöön ja muita testausta hyödyttäviä apuvälineitä on saatavilla verkosta useita satoja erilaisia – useiden ollessa erittäin laadukkaita ja monipuolisia. Niihin syventyminen vaatii aikaa.

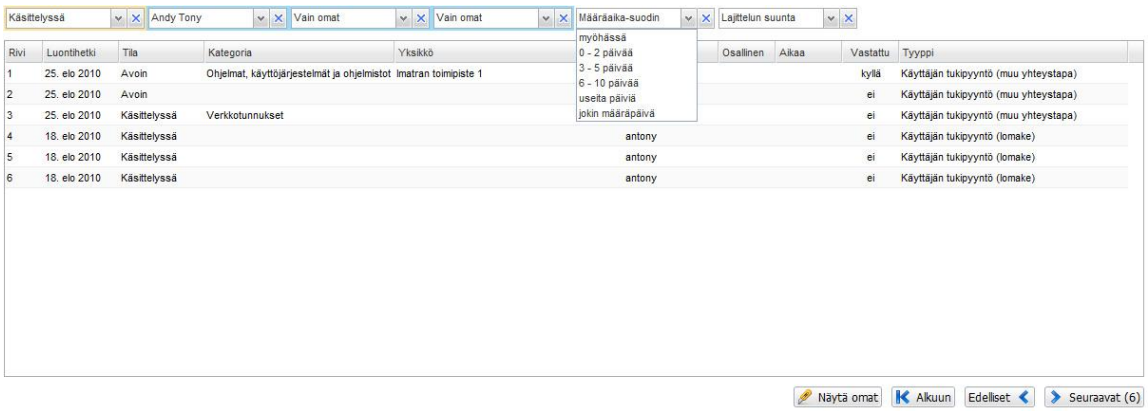
Käytännössä opinnäytetyön näkökulman vaihdos vaikutti eniten opinnäytetyön testauksesta kertovien lukujen merkittävään laajentumiseen. Lisäksi näkökulman vaihdoksen ansiosta pyrkimys edelleenkehittyä SmartGWT-pohjaisen tietojärjestelmän kehittämiseen liittyvien muiden teknologioiden käyttämisessä, vahvistui.

2 TOTEUTETUSTA TIETOJÄRJESTELMÄSTÄ

Kaikki se, mitä tässä opinnäytetyössä kerrotaan kehitetystä järjestelmästä, vastaa sitä järjestelmää, josta mitään jo toteutettua poistamatta, aiotaan jatkokehittää asiakkaan uusien, täydennettyjen vaatimusten mukainen järjestelmä. Vanhalla järjestelmällä viitataan siihen, joka on aiemmin ollut käytössä ja uudella järjestelmällä tarkoitetaan opinnäytetyön aikana toteutettua tukipyyntöjenkäsittelyjärjestelmän versiota 1.0.

2.1 Kuvausta uudesta ja vanhasta tukipyyntöjenkäsittelyjärjestelmästä

Tukipyynnöt, joita sekä vanhan, että uuden järjestelmän kautta välitetään ja käsitellään, ovat peräisin yksiköiden opiskelijoilta, opettajilta ja muulta henkilökunnalta. Osa tukipyynnöistä voi olla myös käsittelijöiden itsensä johonkin tarpeeseen luomia. Tukipyynnön tarve voi kohdistua esimerkiksi tietokoneiden tai ohjelmistojen toimimattomuuteen, mutta se voi olla myös kehitysehdotus (esimerkiksi perustelu jonkin ohjelmiston asentamiseen opiskelijoiden käytössä oleviin tietokoneisiin). Kuvassa 2 on esimerkki siitä, kuinka uudessa tukipyyntöjenkäsittelyjärjestelmässä esitetään joukko tukipyyntöjä listamuodossa.



Kuva 2. Tukipyyntöjä listaava ruudukko käsittelijän käyttöliittymässä

Uudella selaimella käytettävässä tukipyyntöjenkäsittelyjärjestelmässä on kolme eri käyttäjäryhmää (tavallisen tukipyyntöjä lähettävien käyttäjien lisäksi on tukipyyntöjen käsittelijä ja harvemmin tarvittava hallinnoija). Tässä opinnäytetyössä ei tuoda käyttäjän käyttöliittymän toiminnallisuutta esiin kattavasti, koska sen

olemukseen tulevat vaikuttamaan muun muassa tietyt myöhemmin vahvistettavat tietokantarakenteen muutokset. Lisäksi käyttäjän käyttöliittymä rajautuu yhteen monivaiheiseen lomakkeeseen, jolla tukipyyntöjä lähetetään, eikä se teknisesti sisällä mitään erilaista kuin mitä käsittelijän ja hallinnoijan käyttöliittymissä jo esiintyy ja mitä niiden toiminnallisuuksista on jo toteutettu.

Asiakkaan kertoman mukaan nykyisen vanhan järjestelmän laajennettavuuden ja muunneltavuuden heikkous yhdessä toiminnallisten vajeiden kanssa muodostavat perusteen sille, miksi on kannattavampaa toteuttaa kokonaan uusi järjestelmä uusien toiminnallisten määrittelyjen pohjalta.

Kuva 3. Uuden järjestelmän looginen rakenne

Uusi järjestelmä muodostuu kolmesta moduulista (Kuva 3), jotka rajaavat täsmällisesti kolmen eri käyttäjäryhmän käyttöliittymät ja käytettävissä olevat toiminnot toisistaan. Kaikkien kolmen moduulin palvelimelle suuntautuva viestiliikenne kulkee kutsuvälittäjän kautta kunkin moduulin omalle etäkutsukontrollerille.

On ollut toiveena, että käyttäjät oppisivat ja mieltyisivät välittämään tukipyyntönsä verkossa olevan lomakkeen kautta puhelimella soittamisen sijaan, minkä vuoksi opinnäytetyön teoreettiseksi pohjaksi on kerätty tietoa myös lomakkeiden suunnittelemista käytettävyyden ja saavutettavuuden kannalta. Tavoitteena on, että käyttäjä tulisi luontevasti johdatelluksi antamaan tiedot sellaisessa muodossa ja laajuudessa, jotta tukipyynnön kohteesta ja tarpeesta jäisi mahdollisimman vähän epäselvyyttä.

Tukipyyntöjä käsittelevien henkilöiden käyttöliittymät ovat olleet niitä, jotka ovat vaatineet lisätoimintoja, sekä toimintojen selkiyttämistä ja parempaa ryhmittelyä. Erityisen ongelmalliseksi on koettu työnjaon kannalta tarpeettoman informaation näkyminen tukipyyntöjen käsittelijälle, tietynkaltaisten tukipyyntöjen ollessa osoitettuna jonkun toisen tehtäväksi. Minkäänlaista hakutoimintoa ei vanhassa järjestelmässä ollut. Uuteen järjestelmään toteutettiinkin monipuoliset hakutoiminnot ja tukipyyntölistojen suodatusmahdollisuudet.

Hallinnoijan rooli uudessa järjestelmässä asettuu tiettyjen harvemmin tarvittavien muutoksien tekemiseen, kuten uusien tukipyynnöille asetettavien tilojen luomiseen ja käsittelijän asettamiseen jollekin tietylle kategorialle (käsittelijän "omat kategoriat"). Lisäksi hallinnoija luo käsittelijöiden tunnuksia.

Nykyinen järjestelmä tullaan uuden järjestelmän valmistuttua poistamaan käytöstä ilman, että aiempia tukipyyntöjä siirretään uuteen järjestelmään, eikä uusi järjestelmä tule muutenkaan olemaan tietosisällöllisesti tai arkkitehtuurisesti riippuvainen vanhaksi jäävästä järjestelmästä. Järjestelmän käyttämiseen riittää tavanomainen Internet-yhteydellä ja uudehkolla webselaimella varustettu tietokone. Järjestelmä on tavallisten käyttäjien osalta käytettävissä yksiköiden sisäverkoissa. Käsittelijöiden osalta voidaan tarvittaessa luoda salattuja VPN-yhteyksiä, jotta käsittelijät voivat käyttää järjestelmää myös ulkoverkosta. Tässä opinnäytetyössä ei käsitellä verkon fyysistä rakennetta tämän tarkemmin.

Asiakaskoneelle ei prosessitehon suhteen ole tarpeen asettaa nykyistä perustasoa korkeampaa kyvykkyyttä. Näytönohjaimeksi riittää mikä tahansa, mikä soveltuisi esimerkiksi toimisto-ohjelmien käyttöön, vaatimuksena ollen 1280 x 1024 -resoluution käytön mahdollisuus (32-bittiset värit). Äänikortti ei ole välttämätön. Ohjainlaitteiksi riittävät hiiri ja näppäimistö.

2.2 Asiakkaan tarpeiden kartoittamisesta ja siinä onnistumisesta

Ensimmäisten asiakkaan edustajien kanssa käytyjen palaverien pohjalta muodostui alustava näkemys siitä, minkälaisia tarpeita ja vaatimuksia asiakkaalla on. Kyse oli siis nykyisen operatiivisen järjestelmän korvaamisesta, alusta alkaen uudelleen suunnitellen.

Tarvekartoitusta haittasi se, että pyydettyjä tietoa, joiden avulla olisi voinut hahmottaa tarkemmin vanhaksi jäävää järjestelmää ja sen käyttöä, ei joko saanut, niistä puuttui jotain olennaista tai sitten niitä sai liian vähän. Esimerkiksi sähköpostilla ja puhelimitse esitetty pyyntö saada aiemmista tukipyynnöistä esimerkkejä, joiden pohjalta voisi hahmottaa tyypillisiä käyttäjien tukipyyntötarpeita, johti yhden tukipyynnön saamisen sähköpostilla, vaikka opinnäytetyön tekijä korosti, että niitä olisi tarpeen saada lukuisia ja erilaisia. Tämä oli yksi niistä seikoista projektin kulun aikana, joka teki asiakkaan intressien ymmärtämisen hankalaksi. Myöhemmässä lähitapaamisessa tukipyyntöjä pääsi tarkastelemaan lähemmin, mutta asiakas—ratkaisun toimittaja -suhteen välille oli jo syntynyt pieni vaurio, joka heijastui hivenen myös motivaatioon.

Tukipyyntöjärjestelmän toiminnallisuusvaatimuksista yleensä jäi ja pysyi aina elokuun loppupuolelle asti vaikutelma, että monin kohdin toiminnallisuuden voi pitää melko yksinkertaisena. Esimerkiksi tukipyynnön lähettäjän kanssa ei käydä keskustelua järjestelmän itsensä kautta, vaan jokaista tukipyyntöä kohden oli vain yksi (viimeisin) vastaus.

Toteutuksen loppuvaiheilla alkoi asiakkaan suunnalta ilmetä joitakin uusia ideoita, jotka sinänsä olivat täysin tarkoituksenmukaisia järjestelmän käytettävyyden kannalta, mutta jotka suhteessa käytettävissä olevaan aikatauluun ja toteutuksen filosofiaan (erityisesti käyttöliittymien ja tietokannan suunnittelulliset ratkaisut), eivät olleet sovitettavissa mukaan versioon 1.0. Tarpeiden ennakoimiseen vaikutti alkuvaiheen tiedonsaantiongelmien lisäksi sekin, että järjestelmän ehtiessä olemaan useita viikkoja demottavana, ei asiakas esittänyt mitään sellaisia uusia ehdotuksia, joista olisi voinut johtaa päätelmän merkittävien muutoksien tarpeille.

Asiakkaan oli myös erittäin vaikea löytää sellaista yhteistä hetkeä, jolloin useita ihmisiä asiakkaan IT-osaston henkilöstöstä ehtisi kokeilemaan järjestelmän demoversiota ja esittämään siitä mielipiteitä. Palaute kehittyvästä järjestelmästä olisi ollut erittäin tarpeellista, mutta käytännössä palautteen esittäminen delegoitiin yhden asiakkaan edustajan tehtäväksi, jolla ei myöskään ollut riittävästi aikaa perehtymiseen ennen kuin elokuun loppupuolelta alkaen. Tuossa vaiheessa järjestelmää oli kehitetty jo lähes 2 kuukautta.

EKKYn eri järjestelmien yhteentoimivuutta, mahdollista järjestelmäintegrointia tai niiden välistä synkronisaatiota ei käsitelty sellaisenaan, mutta jonkin verran oli hyödyntämättömäksi jäänyttä keskustelua siitä, voisiko tukipyyntöjenkäsittelyjärjestelmä hyötyä jossain toisessa tietojärjestelmässä tai -varastossa olevasta tiedosta, mahdollisesti adapteria tai tiedonmuuntajaa välissä käyttäen.

Tämä olisi vastannut suppean määritelmän mukaista järjestelmäintegrointia, joka kertoisi vain teknisen näkökulman integraatioon. Tällöin kyse on vain valikoimasta teknologioita ja toimintatapoja, joiden avulla muutoin keskenään yhteensopimattomat tietojärjestelmät saadaan kommunikoimaan automatisoidusta keskenään (Tähtinen 2005, 25). Tukipyyntöjenkäsittelyjärjestelmään harkittiin mahdollisuutta hyödyntää muun muassa käyttäjätietokantaa, jota voisi käyttää käyttäjätunnistukseen tukipyyntölomakkeilla, mutta toteutukseen asti tämä idea ei päätynyt. Lopullisesti ideaa ei hylätty, sillä projekti saattaa jatkua Projektityökurssin puitteissa.

Hyväksi ideaksi koettiin sellaisen tietokannan käyttäminen, josta olisi saatavilla osastotiedot (rakennustiedot, osaston koodit, huoneiden numerot, vastuuhenkilöt, jne.), mutta tällaista tietolähdettä ei ollut saatavilla. Järjestelmäintegrointi olisi tässä tapauksessa voinut tarkoittaa sitä, että osastotietokantaa päivitettäessä tukipyyntöjenkäsittelyjärjestelmä pystyisi mukautumaan muutokseen automaattisesti, eikä yksittäisen tukipyynnön osasto-attribuuttia tarvitsisi olla manuaalisesti muuttamassa. Tätä olisi synkronointi eri järjestelmien välillä (Tähtinen 2005, 24).

Suunnitteluun liittyvät neuvottelut ja muut keskustelut rajautuvat myöhemmissäkin vaiheissa yksittäisen järjestelmän eli tukipyyntöjenkäsittelyjärjestelmän suunnitteluun, eikä mukaan keskusteluihin tuotu Business Intelligence -aspektia, joka on merkitykseltään lähellä järjestelmäintegroinnin liiketoiminnan kehittäminen -aspektia: hyvä järjestelmäintegraatio voi toimia tehokkaasti raportoinnin ja monitoroinnin työvälineenä, siltä voidaan kysellä yksityiskohtaista informaatiota yrityksen liiketoimintaprosessien tilasta, sekä se helpottaa analysointia ja nopeuttaa reagointia (Tähtinen 2005, 28).

Business Intelligence -termille ei ole vakiintunutta suomennosta (Hovi ym. 2009, 78), mutta seuraavia termejä on esiintynyt: yritystiedon rikastus, analyyttinen

tiedonhallinta, tiedonhallinnan prosessi ja liiketoimintatiedon hallinta. Termin yritystoimintaan viittaavasta sävystä huolimatta, BI-ratkaisuja käytetään myös julkishallinnon organisaatioissa. Käytännössä BI-ratkaisuissa organisaatiot pyrkivät ymmärtämään ja kehittämään omaa toimintaansa omasta liiketoiminnastaan syntyvän informaation avulla (Hovi, Hervonen & Koistinen 2009, 79).

Toteutusta ja suunnitelmia jälkeenpäin tarkastellessa opinnäytetyön tekijälle heräsi tunne, että olisi ollut hedelmällisempää lähestyä tukipyyntöjenkäsittelyjärjestelmän kehitystä aloittamalla mahdollisten sidosryhmien miettimisellä ja sen pohtimisella millä muilla tahoilla kuin IT-osastolla olisi tarpeita kerääntyvän tiedon käsittelyyn ja niistä johdettujen raporttien ja analyysien tarkasteluun. Voidaan myös retorisesti kysyä, mikä on ratkaisun toimittajan vastuu siitä, että asiakkaan saa tuomaan mahdollisimman varhaisessa vaiheessa esille kaiken sen, minkä asiakas itsekin kokisi hyväksi ideaksi. Kaikkihan ei koskaan tule mieleen yhdellä kertaa. Tämän tukipyyntöjenkäsittelyjärjestelmän kehityksessä on painottunut merkittävästi riittäväksi katsotun toiminnallisuuden toteuttaminen ja testauksen osuus.

Toiminnallista määrittelyä kirjoitettaessa ei otettu huomioon muihin järjestelmiin kytkeytymistä piirretyissä UML-kaavioissa, mikä johtui todennäköisesti siitäkin, että opinnäytetyön tekijä ei vielä tuolloin ollut käynyt Ohjelmistoarkkitehtuurit-kurssia. Tästäkin huolimatta, pyrittiin suunnittelun ja toteutuksen aikana huomioimaan sijainnit potentiaalisille rajapinnoille, joiden kautta ulkoiset järjestelmät voivat kommunikoida tukipyyntöjenkäsittelyjärjestelmän kanssa.

Tukipyyntöjenkäsittelyjärjestelmän omat mahdolliset tarpeet ulkoisten tietolähteiden hyödyntämiselle tai toisen tietojärjestelmän kanssa kommunikoimiselle on helppo keskittää yksittäisen komponentin tehtäväksi, eikä ohjelmistoarkkitehtuuria tarvitsisi muuttaa paljoakaan. Tämä johtuu siitä, että mahdollisessa järjestelmäintegraatiossa olisi mahdollista ohittaa SmartGWT-sovelluskehys, sillä se vaikuttaa enimmäkseen käyttöliittymäpuolella, järjestelmäintegraatioon liittyvien kytkentäpisteiden sijoittuessa palvelinpuolelle (lisätietoa alakohdassa 5.8, "Tietokannasta", jossa kuvaillaan lyhyesti kolmitasorakennetta).

Suunnittelun kulusta johtuen on oltu lähellä vaaraa, josta kirjassa Järjestelmäintegraatio varoitetaan (Tähtinen 2005, 34) eli "yritykseen voi syntyä koko joukko

erilaisia, eri yksiköiden ja eri ihmisten eri aikoina ja eri tekniikoilla rakennettuja *ad-hoc*-linkkejä ohjelmistojen välillä". Vastaavasti, kirjassa Tietovarastot ja Business Intelligence varoitetaan (Hovi ym. 2009, 19) siitä, että "jos organisaatiossa ei ole tietovarastostrategiaa, tekevät eri organisaatioyksiköt ratkaisujaan tilanteen mukaisesti, *ajopuu*-tyyppisesti".

Johtuen nykypäivänä vähällä rahalla saatavista nopeista ja hyvälaatuisista kiintolevyistä, muisteista ja prosessoreista, voi jälkeenpäin arvioiden todeta, että kapasiteetin riittävyys ja tietokantakyselyiden nopeus vaikutti merkittävästi päätökseen jättää kerääntyneen tiedon arkistoinnin ja niistä johdettujen raportoinnillisten tarpeiden miettiminen melko myöhäiseen vaiheeseen. Kirjassa "Tietokantojen suunnittelu ja indeksit" todetaan (Hovi, Huotari & Lahdenmäki 2005, 80) raportoinnin olevan perinteisesti sellainen tietojärjestelmän suunnittelun osa-alue, jonka ajatellaan voitavan tehdä "joskus myöhemmin".

Raportoinnin osalta tämä myöhemmässä vaiheessa miettiminen oli osittain perusteltuakin, sillä se oli vahvasti riippuvaista siitä, että tietokannan rakenne ei enää muuttuisi. Käytännössä tietokannan rakenne ei projektin edetessä muuttunut merkittävästi mutta täydentyi kuitenkin muutamia kertoja, opinnäytetyön tekijän hyväksyessä toteutukseen mukaan lukuisia asiakkaan kehitystyön aikana ehdottamia pieniä muutoksia tai lisäyksiä toiminnallisuuteen.

Erillistä tietovarastoa ei missään vaiheessa harkittu, mutta arkistoinnista oli keskusteltu toiminnallisen määrittelyn ensimmäisen version tuottamisen yhteydessä. Tukipyyntöjenkäsittelyjärjestelmän hakutoimintojen tullessa demokäytettäväksi ei enää nähty tarvetta erilliselle arkistointitoiminnalle. Päätös perustui SQL-hakulauseiden nopeuteen ja hakutoiminnon käytettävyyteen.

Raportointi- ja hakutoiminnot tulevat olemaan eniten järjestelmää kuormittavista toiminnoista, mutta tällöinkin yksittäisen IT-osaston käsittelijän pitäisi suorittaa useita vastaavia toimintoja peräkkäin, jotta niillä olisi kuormituksen kannalta merkitystä.

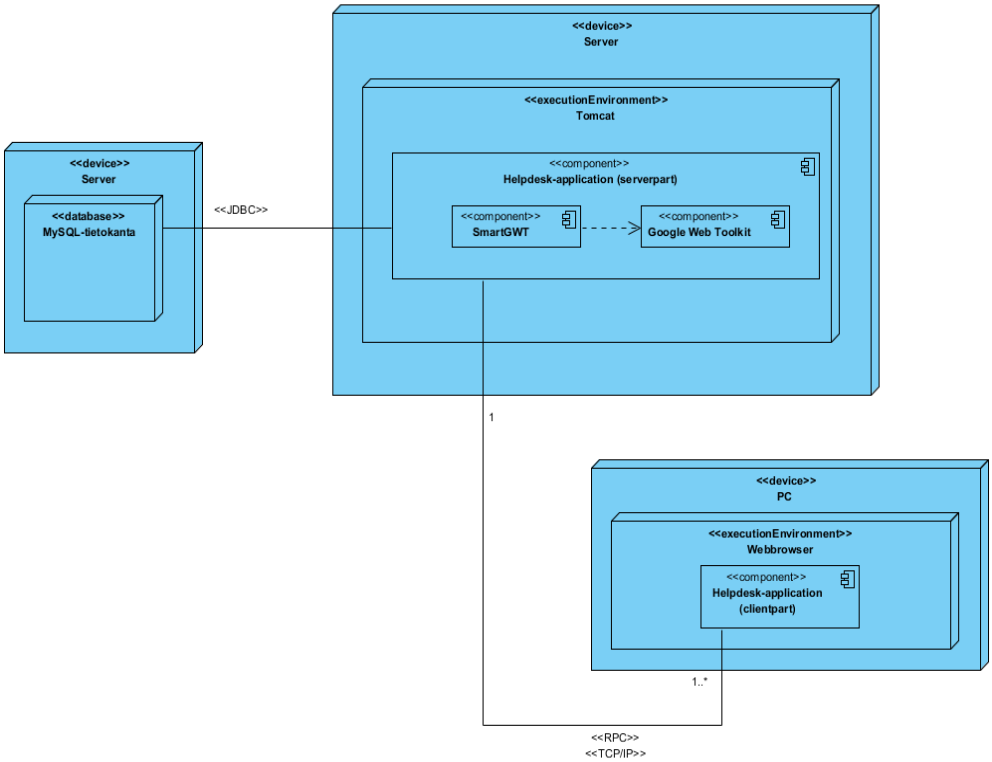
Jos tukipyyntöjenkäsittelyjärjestelmään sisällytettäisiin eräajo-toiminto eli toiminto joka suorittaa automatisoidusti lukuisia samankaltaisia, peräkkäisiä toimintoja (esimerkiksi tukipyyntöjen hakuja ja niiden tulostamista), voisi erillisen tietova-

raston käyttö olla perusteltua. Tietovaraston tiedot olisivat vain lukukäytössä ja siihen luettaisiin tiedot operatiivisesta tukipyyntöjenkäsittelyjärjestelmästä vain sellaiseen aikaan, jolloin tiedon luenta ei haittaisi sen toimintaa. Jos tietovarasto otettaisiin käyttöön, olisi tietenkin hyödyllistä miettiä, kuinka se voisi hyödyntää koko organisaation (EKKYn) toimintaa.

2.3 Tietojärjestelmän laitteisto- ja ohjelmistoympäristöstä

Toiveena asiakkaan puolelta oli alun perin esitetty, että Linuxin sijaan käytettäisiin Windows-ympäristöä, perusteluna ollen, että IT-palveluita tarjoavassa tiimissä on kerrotun mukaan vain Windows-asiantuntijoita. Tämä olisi voinut olla merkitsevä seikka tilanteessa, jossa henkilöstöresursseja ei ole käytössä tarpeeksi suhteessa työmäärään. Projektin alkuvaiheessa pidettiin vielä mahdollisena, että tukipyyntöjenkäsittelyjärjestelmän ylläpidollisista syistä olisi poikkeustilanteissa tarpeen osata konfiguroida SmartGWT:n käyttämää sovelluspalvelinta tai Linux-pohjaista käyttöjärjestelmää, johon sovelluspalvelin on asennettu.

Tämä Windows-rajoittuneisuus kumottiin myöhemmin (heinäkuussa), mutta ennen asiakkaan näkemyksen vaihtumista Windows-alustasta Linuxiin, pidettiin lähtöolettamuksena sitä, että olisi käytettävä Windows Server 2008 r2:ta. Siihen sopivista sovelluspalvelimista oli saatavilla Glassfish v3, joka oli käypä vaihtoehto SmartGWT:n alustaksi (Glassfishiin liittyvistä lisensiointiongelmista tarkemmin alaluvussa 6.6 ("Ohjelmistojen lisenssit ja niiden yhteensovittaminen"). Tomcatin ja Microsoft Server 2008 r2:n yhteensovittaminen olisi ollut enimmäkseen verkossa luettavissa olevien yksityishenkilöiden blogiviestien varassa, joissa kuvaillaan kuinka kyseinen yhdistelmä teoriassa olisi mahdollinen tai on ainakin kertaalleen saatu toteutettua.



Kuva 4. Järjestelmän sijoittelukaavio

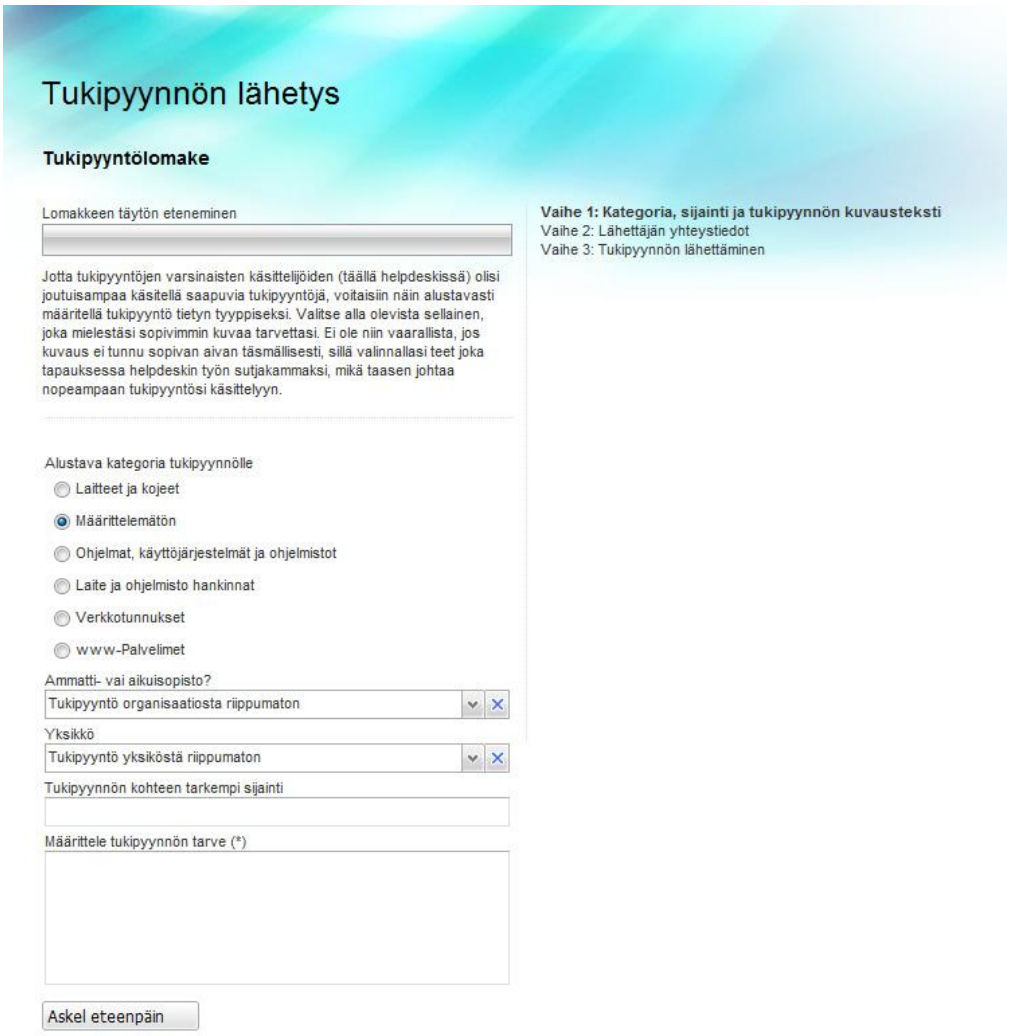
Uuden järjestelmän arkkitehtuuri on sijoittelukaaviona esitettynä (Kuva 4) varsin yksinkertainen. Se koostuu fyysisestä palvelimesta, jolla saattaa lopulta tulla sijaitsemaan myös tietokanta, joka kuvassa on esitetty sijaitsevan omalla fyysisellä palvelimellaan. Varsinainen sovellusosuus koostuu palvelinosasta ja asiakasosasta, jotka viestivät keskenään TCP/IP -protokollan kautta. SmartGWT:n valinta sovelluskehykseksi asettaa sovelluspalvelimelle tiettyjä rajoituksia. Java-pohjaisista sovelluspalvelimista käyttökelpoisia ovat Tomcat (versio 6.0 tai uudempi), Glassfish (v3 tai uudempi) ja JBoss AS (5.0 tai uudempi), joiden välillä voi vaihtaa tuotteen käyttöönoton jälkeenkin ilman, että ohjelmakoodiin tarvitsee tehdä konfigurointia suurempia muutoksia.

Kehitysvaiheessa tietokannan hallintajärjestelmänä käytettiin MySQL:ää, mutta myöhemmässä käyttöönottovaiheessa siirrytään käyttämään Microsoft SQL Server Expressiä tai PostgreSQL:lää. Koska kaikki tietokantatransaktiot on toteutettu käyttäen JDBC:tä (Java Database Connectivity), ei myöskään tietokannan hallintajärjestelmän vaihdos tulisi aiheuttamaan ohjelmointikoodin erityisiä muutoksia, koska JDBC yhtenäistää metodien käytön.

Järjestelmän jatkokehitysvaiheissa saatetaan hyödyntää EKKYssä nykyisin käytössä olevia käyttäjätietokantoja kahdella tapaa: käsittelijöiden autentikointiin (LDAP) ja käyttäjien tunnistamiseen sähköpostin osoitteen perusteella (Microsoft Active Directory). Ensin mainittuun on jo varauduttu: muutostöitä tulisi tehtäväksi vain yhdessä palvelinpuolen Java-luokassa, johon autentikointi on keskitetty. LDAP:n käsittelyyn Javalla on kuitenkin allokoitava aikaa myös teoriapohjan luomiseksi. Toistaiseksi tukipyyntöjen lähettäjien tunnistus perustuu käyttäjän itsensä antamiin tietoihin kuten sähköpostiosoitteeseen.

Nykyisen järjestelmän käyttämiin ohjelmistoteknologioihin tai alustaratkaisuihin ei ole tarpeen sitoutua tai perustaa mitään niiden päälle, sillä uusi järjestelmä asennetaan tarkoitukseen dedikoidulle virtuaalipalvelimelle, jonka voi varustaa sopivaksi katsotuilla teknologioilla. Opinnäytetyön rajoissa järjestelmä toimii ilman riippuvaisuuksia muista organisaation (EKKY) järjestelmistä, eikä ulkopuolisia tietovarastoja ei käytetä – poikkeuksena sähköpostin lähettämiseen käytetty sähköpostipalvelin.

Palvelun tavallisille käyttäjille näkyvä käyttöliittymä tullaan upottamaan Iframe-tekniikkaa käyttäen EKKYn verkkosivuille, jotka tulevat sijaitsemaan eri palvelimella kuin upotettava sivu ja jotka tuotetaan dynaamisesti jo nykyisinkin käytössä olevaa Joomla-sisällönhallintajärjestelmää käyttäen. Tätä havainnollistaa kuvan 5 mukainen havainnekuva, jossa tausta ja pääotsikko ovat osa sivua, johon lomake saumattomasti upotetaan Iframe-tekniikkaa käyttäen.



Kuva 5. Havainnekuva käyttäjän tukipyyntölomakkeesta

Kertomansa mukaan asiakkaan aikomuksena on ollut päättää lopullisesta laitteistoympäristössä vasta myöhemmin. Tätä ennakoiden on järjestelmää suunnitellessa muodostettu seuraavanlaisia suosituksia laitteistoympäristön osatekijöiksi (katso myös alaluku 6.6, "Ohjelmistojen lisenssit ja niiden yhteensovittaminen").

Käyttöjärjestelmänä palvelimella pitää olla jokin Linux-levitysversioista, sovelluspalvelimena Apache Tomcat (6.0) ja tietokannan hallintajärjestelmänä viiteavaimia tukeva tietokannan hallintajärjestelmä kuten MySQL 5.1 (InnoDB-moottorilla). Koska opinnäytetyön tekijä ei halua asettaa järjestelmää GPL-lisenssin alaiseksi (lisensseistä tarkemmin alaluvussa 6.6, "Ohjelmistojen lisenssit ja niiden yhteensovittaminen"), suositellaan muina sopivina vaihtoehtoina Microsoft SQL Server Expressiä tai PostgreSQL:lää. Ensin mainittua ei ole

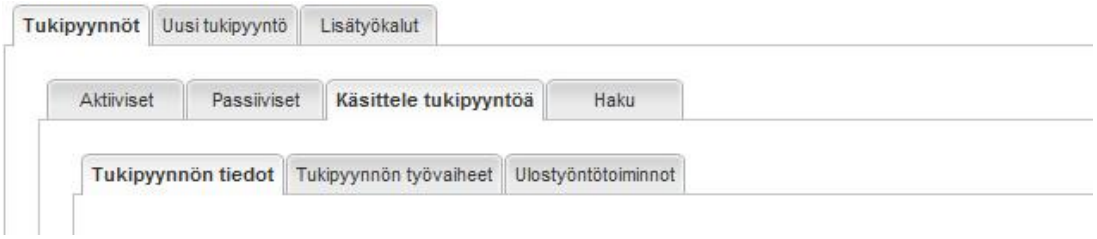
saatavilla Linux-alustalle, joten sen vaihtoehdon osalta olisi tarpeen harkita uudemman kerran Windows-pohjaisen alustaratkaisun kokoonpanoa.

Linux-pohjaisena toteutettuna voidaan muihin tarpeellisiin ohjelmistoihin lukea myös phpMyAdmin, joka mahdollistaa tietokannan hallitsemisen selaimen kautta. Useimmat Linuxin levitysversiot sisältävät vakiona tuen PHP-ohjelmointikielisillä ohjelmille, jollainen myös phpMyAdmin on. Se asettaa tarpeen myös HTTP-yhteyksien käsittelylle, mihin suositellaan Apache HTTP Serveriä 2.2.16 tai uudempaa. SSH-yhteydet mahdollistava daemoni on tarpeellinen, jos palvelinta on tarpeen käsitellä muualta kuin paikallisesti. Javasta tarvitaan JDK 6 (Standard Edition) Update 21 tai uudempi.

Jos käytetään MySQL:lää ja InnoDB-tietokantamoottoria, olisi prosessorin hyödyllistä olla 64-bittinen, sillä InnoDB-tietokantamoottori hyötyy suoritustehon osalta 64-bittisestä laitealustasta (sekä 64-bittisestä käyttöjärjestelmästä ja 64-bittisestä MySQL:n versiosta). InnoDB:n hyödyntää tehokkaasti saatavilla olevan lisämuistin, pyrkien myös optimisoimaan käytönaikaista muistinkulutusta ja levynkäyttöä, sekä ennakoimaan tulevaa (Oracle 2010): sen havaitessa tietojen luennassa tutun kaavan, se lähettää joukon asynkronisia levyn lukemisia hakemaan hetkeä myöhemmin tarvittavaa tietoa jo ennalta.

2.4 Käyttäjäryhmien käytössä olevat toiminnot

Toteutetussa tukipyyntöjenkäsittelyjärjestelmässä jokaisella käyttäjäryhmällä on omat erilliset toiminnot käytettävissään, jotka on pyritty sijoittamaan välilehdistä muodostuvaan hierarkiaan (esimerkkinä kuva 6) siten, että yhdellä välilehdellä olisi vain yhden käyttötapauksen toteuttava toiminto. Tämä oli lähtökohtainen pyrkimys, minkä hyödyllisyyttä pyrittiin korostamaan nimeämällä välilehdet siten, että niissä olisi mahdollisimman vähän semanttista päällekkäisyyttä. Kaikkien toimintojen tapauksessa tämä ei ollut perusteltua, koska tietyssä tilanteessa tarvittavien toimintojen haluttiin olevan tavoitettavissa mahdollisimmin vähin siirtymän ja klikkauksin. Tämän vuoksi esimerkiksi "Käsittele tukipyyntöä" -välilehdellä on enemmän kuin yhden samantyyppisen käyttötapauksen mahdollistava toiminnallisuus käytettävissä.



Kuva 6. Käsittelijän käyttöliittymän välilehdistä muodostuu hierarkia

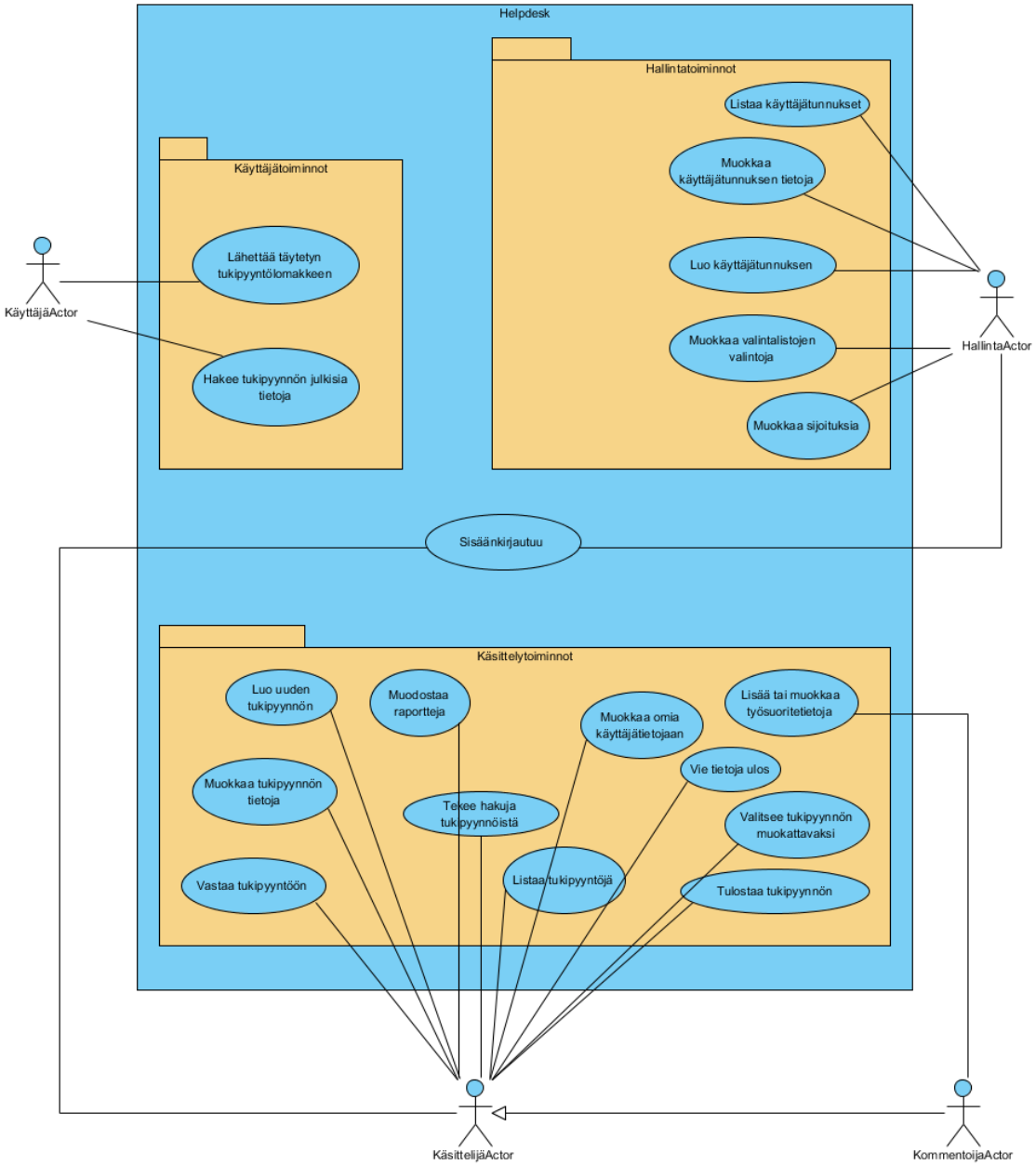
Toimintojen ollessa sanallisesti ilmaistuna lähes samat kuin käyttötapausten kuvaukset ja joista välilehtien korvakkeissa esiintyvät tekstit oli johdettu, muodostui järjestelmän toteutuksesta helpompaa, sillä tämä yhtenäisti arkkitehtuurin eri abstraktiotasojen (tai näkökulmien) kohdistuneisuutta toisiinsa nähden. Tätä yhtenäistämispyrkimystä tukee sekin, että kunkin kolmen eri käyttäjäryhmän käyttöliittymät sijaitsevat omassa moduulissaan. Moduuli tarkoittaa kokonaisuutta, joka erottaa eri käyttäjäryhmien käyttöliittymät toteuttavan ja kontrolloivan koodin omiin toisista riippumattomiin, mutta samalla myös toisistaan tietämättömiin osiin. Kutakin moduulia vastaa yksi verkko-osoite, johon menemällä avautuu käyttäjäryhmästä vastaavan moduulin etusivu (entry-point).

Järjestelmän kolmella eri käyttäjäryhmällä on seuraavat toiminnot käytettävissään.

- *Tavallinen käyttäjä:* tukipyynnön lähetys; tukipyynnön julkisten tietojen näyttäminen
- *Hallinnoija:* käyttäjätunnuksien listaus, luonti ja muokkaus; valintalistojen valintojen listaus, luonti ja muokkaus; sijoituksien muokkaus (kategorioiden käsittelijät ja käsittelijöiden yksiköt)
- *Tukipyyntöjen käsittelijä:* uuden tukipyynnön luonti, tukipyyntöjen tietojen muokkaus, vastausviestin lähettäminen tukipyyntöön; tukipyyntöjen listaaminen (aktiiviset ja passiiviset erikseen); haku tukipyynnöistä; työsuoritteiden listaaminen, luonti ja muokkaaminen; tukipyynnön tietojen tulostaminen; omien tietojen muokkaus; raporttien tulostus

Järjestelmää mallinnettaessa määritellään jokaiselle mahdolliselle, erilaiselle käyttötapaukselle oma käyttötapauskuvauksensa, joka koostuu tesktipohjaisen kuvauksen lisäksi vähintään yhdestä sekvenssikaaviosta ja kommunikaatiokaa-

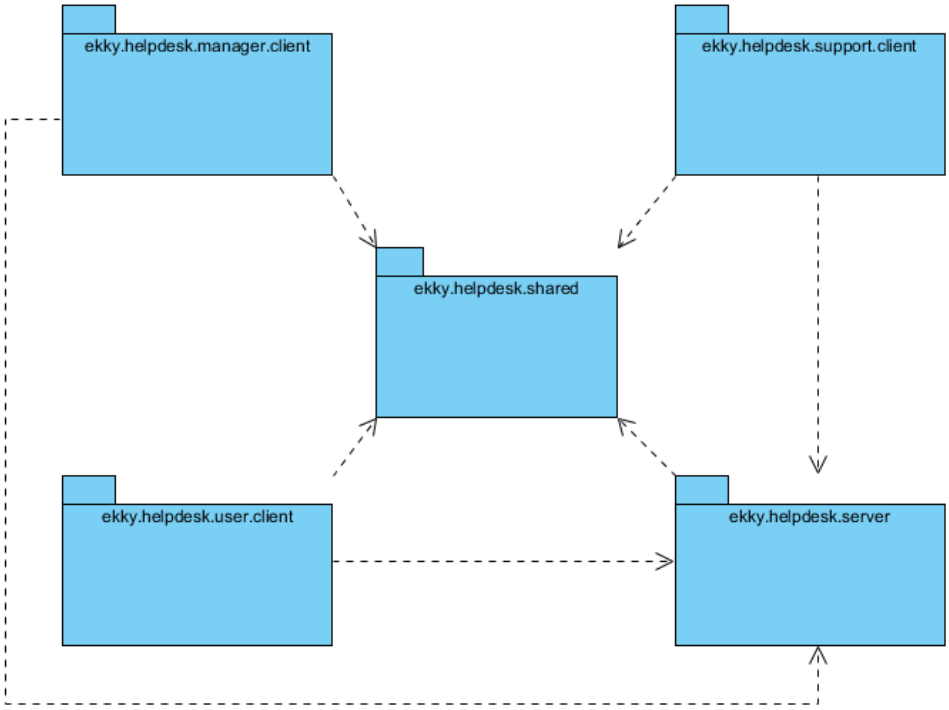
viosta. Koska kyseessä ei ole arkkitehtuurisesti erityisen kookkaasta järjestelmästä, pystytään erilaiset käyttötapaukset esittämään yhden käyttötapaukset yhteen kokoavan käyttötapauskaavion avulla (Kuva 7).



Kuva 7. Järjestelmään liittyvät käyttötapaukset

Jokaista kolmea käyttäjäryhmäkohtaista moduulia vastaa lähdekoodissa oma pakettinsa (esimerkiksi hallinnoijaa vastaa paketti *ek-ky.helpdesk.manager.client*), joiden lisäksi on yksi yhteinen paketti (*ek-ky.helpdesk.shared*) sekä paketti palvelimella ajettavalle koodille (*ek-ky.helpdesk.server*). Käytännössä jokainen käyttäjäryhmäkohtainen paketti on

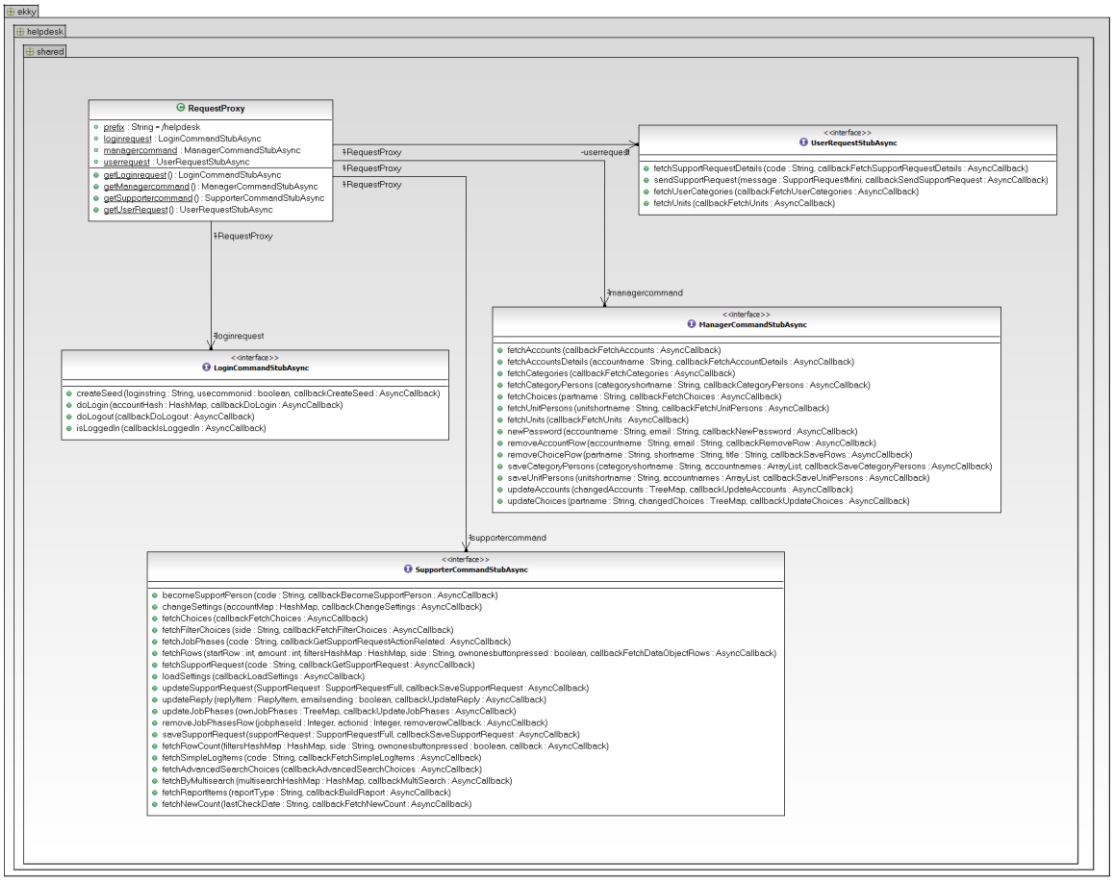
riippuvainen shared-paketista, koska etäkutsut palvelimelle kulkevat poikkeuksetta sen kautta. Näitä riippuvaisuuksia selventää kuva 8.



Kuva 8. Sovelluksen moduulirakenne kehitysvaiheessa

Näiden pakettien alle luotavissa alipaketeissa on jaottelu pyritty tekemään temaattisin perustein eli esimerkiksi tukipyyntöjen listaamiseen ja käsittelyyn liittyvät Java-ohjelmointikieliset luokat sijaitsevat paketissa *ekky.helpdesk.support.client.multiplesupportrequests*.

Kaikki tietoliikenne sovelluksen asiakaspään ja palvelimen välillä kulkee shared-paketissa sijaitsevan RequestProxy-luokan kautta, jonka voi ajatella muodostavan yhteisen kanavan, kaikelle verkon yli tapahtuvalle viestiliikenteelle. Kukin moduuleista käyttää omaa "metodinippuaan" (Kuva 9), mikä ei kuitenkaan johdu sovelluskehyksen asettamista rajoitteista, vaan sekin on suunnittelullinen valinta. Sisäänkirjautumiselle (ja uloskirjautumiselle), joka hallinnoijalta ja käsittelijältä vaaditaan, on selkeyden vuoksi oma metodinippunsa.



Kuva 9. Järjestelmän käyttämät etämetodit per moduuli.

Palvelinpuolen kontrolliluokkaan on kehitystyön aikana kertynyt lähes 20 erilaista metodia. Niiden nimeämiskäytäntö on kehitystyön aikana hieman yhtenäistetty, jotta jonkin tietyn metodin löytäminen muiden joukosta olisi nopeampaa. Useimmat metodit hakevat dataa palvelimelta, joten oli luonnollista lisätä niiden alkuun yhteinen sana 'fetch'. Näistä etäkutsuttavista metodeista on kerrottu tarkemmin alaluvussa 4.3 ("Etämetodien kutsuminen").

3 KÄYTTÖLIITTYMIEN SUUNNITTELUSTA

Koska kyseessä on graafisten käyttöliittymien kautta käytettävästä tietojärjestelmästä, on järjestelmän käytettävyyden kannalta tärkeää kiinnittää erityistä huomiota siihen miten informaatio esitetään ja käyttöliittymän komponentit asetellaan.

3.1 Käyttöliittymien suunnittelun teoriaa

Käyttöliittymien näyttöjen suunnittelun ja toteutuksen tueksi ei vielä projektin alkuvaiheilla oltu käyty läpi käyttöliittymien suunnitteluun tai psykologiaan liittyvää kirjallisuutta tai tutkimuksia. Tästäkin huolimatta ne säännöt, joita käyttöliittymien suunnittelussa noudatettiin, olivat hyvin linjassa sen suhteen mitä kirjallisuudessa suositellaan. Käyttöliittymien suunnittelu ja elementtien asettelu on kokonaisuudessaan opinnäytetyön tekijän tekemää. Erityisen tarkka pyrittiin olemaan siinä, ettei mikään näytöistä sisältänyt niin paljon elementtejä tai havainnoitavissa olevia asioita, että ne vaatisit käyttäjän kognitiiviselta kapasiteetilta (Saariluoma 2004, 69) liian paljon.

Kirjallisuudessa (muun muassa Reason 1990, 31), jossa käsitellään aivojen tiedonkäsittelyä, ei yleensä ole jätetty mainitsematta, että ihmisen työmuistin koko on "7 toisiinsa liittymätöntä asiaa plus miinus kaksi". Enempien asioiden tietoisessa mielessään pitäminen vaatii havainnoitavien kohteiden ryhmittelyä hahmolakien (Saariluoma 2004, 74) mukaisiin ryhmiin kuten samanväristen, toisiaan lähellä olevien, samanmuotoisten tai samassa linjassa olevien ryhmiin.

Ryhmittelyn huomioimisen lisäksi oli katsottu tärkeäksi olla kuormittamatta käyttäjän muistia sillä, että hän joutuisi pitämään mielessään mitä oli hetki sitten tehnyt yhdellä näytöllä, suorittaessaan jotain toimintoa toisella näytöllä, sillä se haittaisi tarkkaavaisuuden ylläpitämistä ja vahvojen muistisääntöjen syntymistä (Reason 1990), sekä se voisi aiheuttaa ahdistuneisuuden tunteiden kytkeytymisen havaintoihin (Saariluoma 2004, 79). Saariluoma mainitsee Käyttäjäpsykologia-kirjassaan mahdollisuuden sille, että käyttäjä voi prosessoida ahdistuksen kohteita hitaammin kuin kohteita, joiden kohdalla käyttökokemus on ollut positii-

vinen. Tämä on eräs näkökulmista, jota ei sellaisenaan ajateltu järjestelmän käyttöliittymiä suunnitellessa.

James Reasonin kirja Human Error tuo esille lukuisia aspekteja, jotka liittyvät käyttäjien tekemien virheisiin. Hän sijoittaa nämä yleisessä virheidenmallintamissysteemissään (Generic Error Modeling System, GEMS) kolmeen kategoriaan (1990, 53): taito-pohjaiset virheet (skill-based errors), sääntöpohjaiset virheet (rule-based errors) ja tietämystason virheet (knowledge-based errors).

Taito-tason virheitä olisivat esimerkiksi tilanteet, joissa rutiinin muutos ei johda pysyvään muutokseen, vaan tavataan palata takaisin vanhoihin taipumuksiin, erityisesti tarkkaavaisuuden herpaantuessa sekä tilanteet joissa hetki sitten ymmärsi, että jotain ei voi tehdä, mutta tekee epähuomiossa saman virheen uudelleen, sekä tilanteet, joissa kaksi erillistä havainnoitavaa kohdetta sotkeutuu toisiinsa (kirjassa esimerkkinä kissanruoan laittaminen teepannuun).

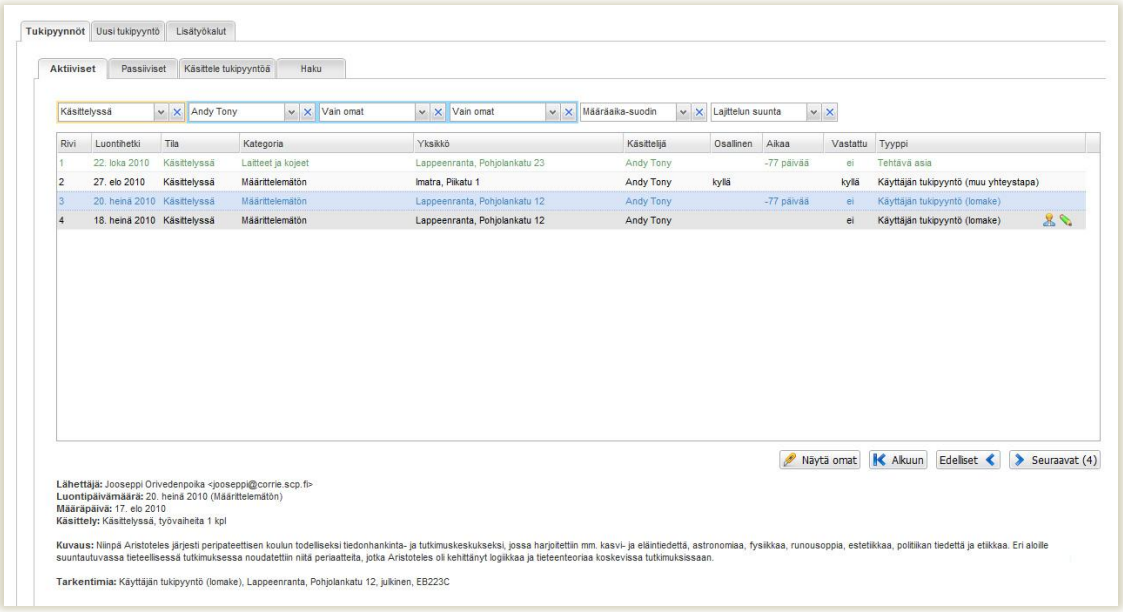
Virheiden luokittelujärjestelmänä Reasonin virheidenmallintamissysteemi voi olla hyväkin työväline, mutta tiettyjen virhetyyppien kohdalla on helppo nähdä yhteyksiä jokaiseen kolmesta kategoriasta. Esimerkiksi sääntö-tason virhe "informaation ylikuormitus" voi tietyissä tilanteissa liittyä läheisesti tietämystason virheeseen "poissa mielestä, poissa tietoisuudesta". Muina tietämystason virheiden potentiaalisina aiheuttajina on mainittu toimintojen kompleksisuuden asteen vertailun vaikeus (esiintyisi vaihtoehtoisen reitin tai toimintamallin valinnassa), sekä liika (kasaantuva) yliluottavaisuus itseen.

Käyttöliittymissä esiintyvien sanojen osalta tuntui luontevalta pyrkiä välttämään samankuuloisten sanojen tai paljon samaa merkitsevien käsitteiden käyttöä. On tutkittu (Reason 1990, 31), että foneettisesti samankaltaiselta kuulostavat sanat voivat myös visuaalisessa muodossa esitettynä heikentää työmuistin toimintaa, joten päätöstä pyrkiä yksiselitteisyyteen voidaan pitää perusteltuna.

3.2 Valikoima käsittelijän näyttöjä

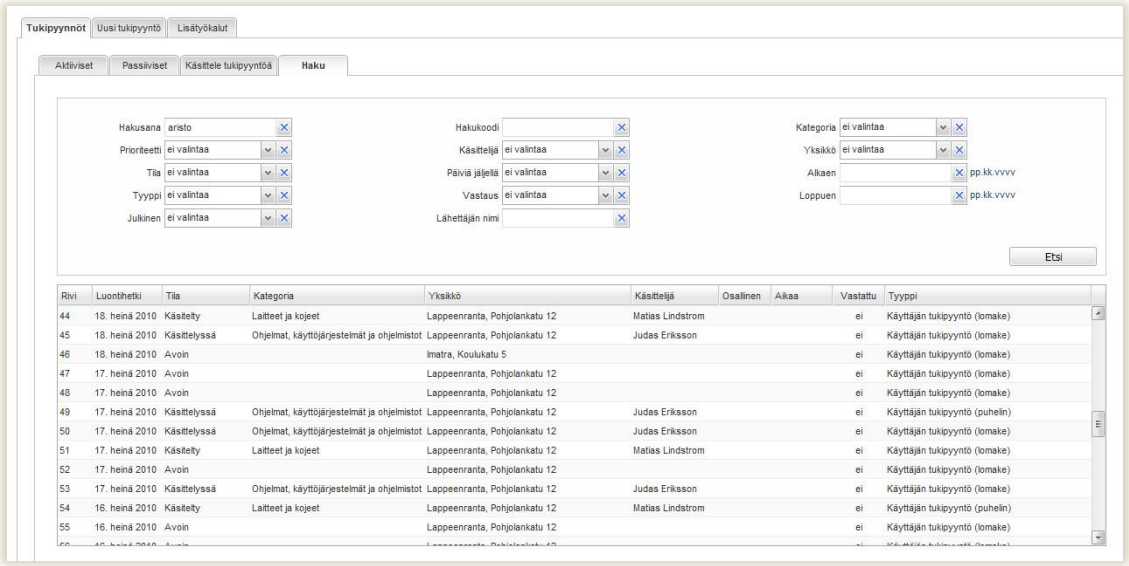
Tukipyynnöt on jaoteltu niiden tyypin mukaan kuuluvaksi joko aktiivisiin tai passiivisiin. Aktiviisia tukipyyntöjä (Kuva 10) ovat muun muassa tyypiltään avoimina tai käsiteltävinä olevat, mutta tarvittaessa erilaisia tyyppejä voidaan luodaa li-

sääkin. Tukipyyntöjä voidaan selailla, suodattimien tukemana, omilla ruudukoillaan,. Lisäksi tukipyyntöriviä kerran klikkaamalla saa siitä esiin esikatselutiedot ja tuplaklikkaamalla sitä pääsisi muokkaamaan. Tukipyyntörivien yllä liikkuu hiiren pointterin liikkeitä seuraten rollover-palkki, jossa on erilaisia toimintaikoneita.



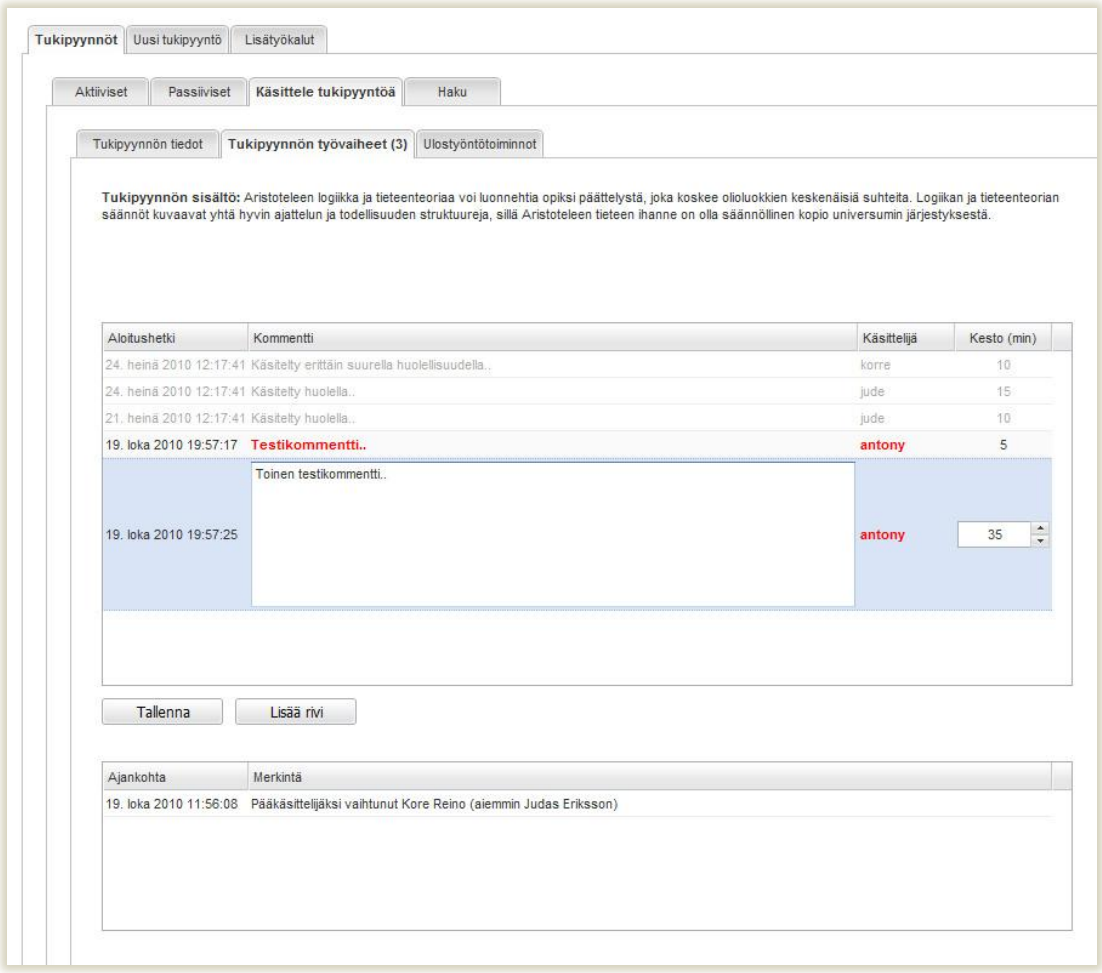
Kuva 10. Aktiivisten tukipyyntöjen listaus (suodattimia käytössä)

Haku-välilehdellä (Kuva 11) käsittelijä voi käyttää tietynlaisten tukipyyntöjen hakemiseen monipuolisempia suodattimia ja hakuehtoja. Käytännössä asetaan valintalistoista halutut ehdot ja täytetään halutut tekstipohjaiset rajaukset, sekä mahdollisesti asetetaan myös aikarajaus, joka kohdistuu tukipyynnön luomispäivään.



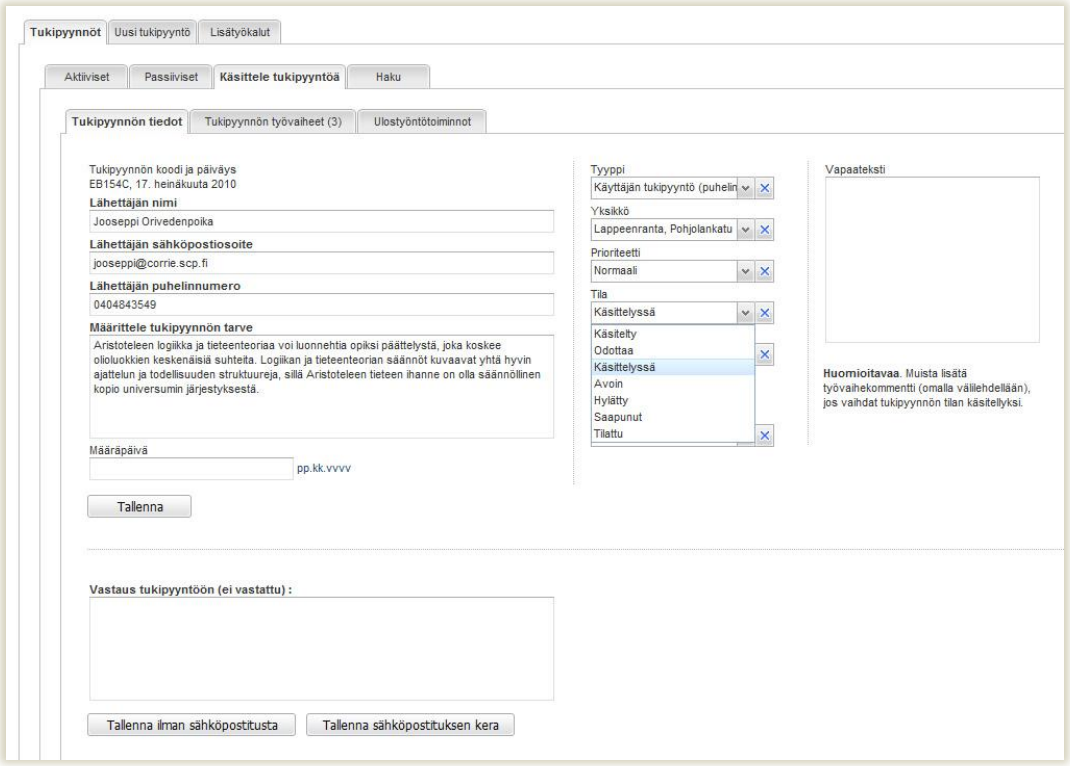
Kuva 11. Tukipyyntöjen haku (hakuehto käytössä)

Käsittelijä voi lisätä tai muokata tukipyyntökohtaisesti työsuoritetietoja (vain omiaan voi muokata), joille tallentuu lisäysajankohta, sekä käsittelijän itse asettama kesto ja kommentti. Tätä havainnollistaa kuva 12. Kommentti-kenttä voi sisältää monirivistä tietoa.



Kuva 12. Tukipyyntöön liittyvien työvaiheiden kommenttien muokkausta

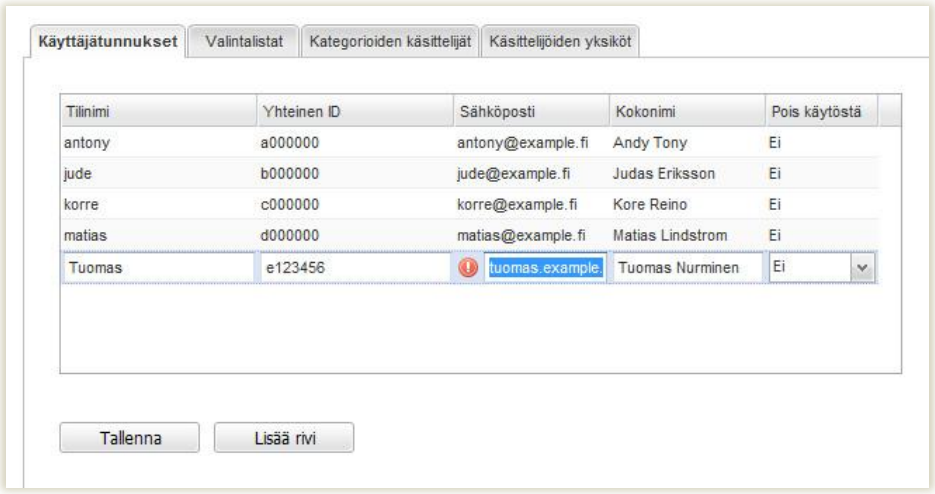
Käsittelijä voi muokata tukipyynnön tietoja (Kuva 13) muokkaamalla tietokenttiä ja asettamalla valintalistoista haluamansa valinnat. Lisäksi voidaan asettaa Julkinen-attribuutti, joka määrittää sen, voiko tukipyyntöä nähdä käsittelijän käyttöliittymien ulkopuolelta eli käyttämällä tiettyä sovittua verkko-osoitteen muotoa, jossa on annettu parametrina tukipyynnön koodi. Vapaateksti-kenttä on lisätty toteutuksen loppuvaiheilla. Huomionarvoista tässä on ryhmittelyn vaikutus lomakkeen käytettävyyteen.



Kuva 13. Tukipyyntö haettuna muokattavaksi

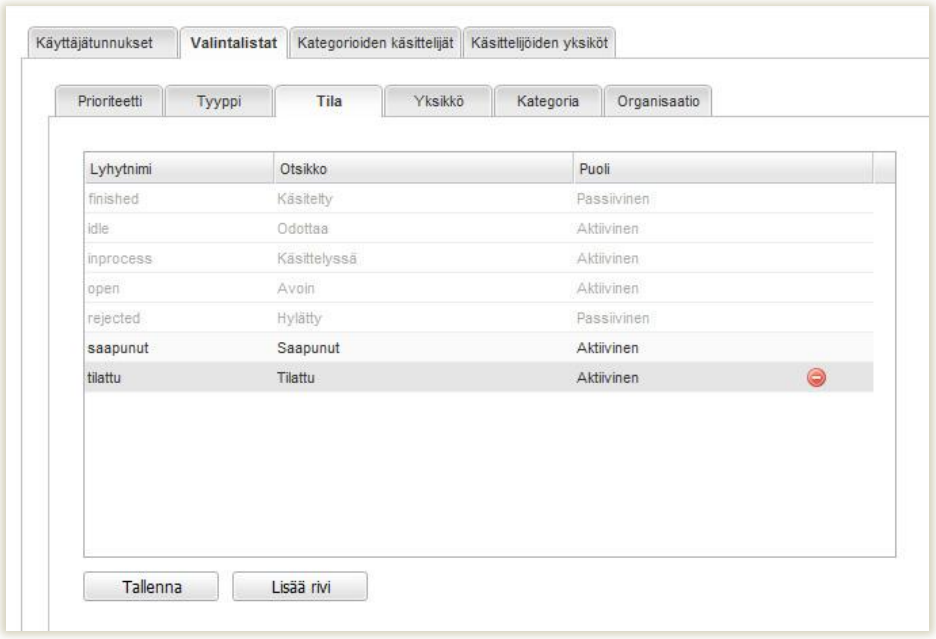
3.3 Valikoima hallinnoijan näyttöjä

Hallinnoija voi omassa käyttöliittymässään luoda ja muokata käsittelijöiden käyttäjätunnuksia (Kuva 14). Tarvittaessa käyttäjätunnuksen voi asettaa myös pois käytöstä olevaksi.



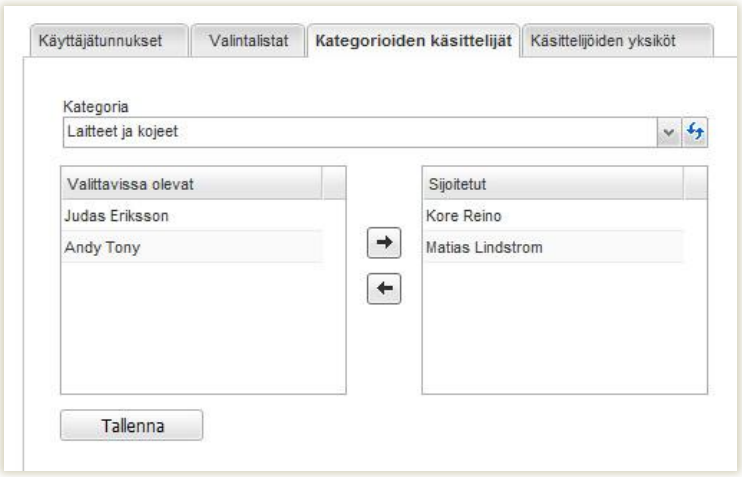
Kuva 14. Käyttäjätunnuksien luonti ja muokkausta

Tukipyyntölomakkeilla (käyttäjän ja käsittelijän käyttöliittymissä) eräinä lomake-elementteinä valintalistoja, joilla voidaan asettaa tukipyynnölle erilaisia tarkentimia. Näitä ovat muun muassa kategoria, yksikkö, prioriteetti, tyyppi ja tila. Nämä ovat sellaisia, joita voidaan tarvittaessa luoda lisää, sekä osan näistä voi asettaa pois näkyvistä tietyissä yhteyksissä. Osa näistä valintalistoihin päätyvistä kohteista on sellaisia, että niitä ei voi muuttaa, eikä poistaa. Muokkaaminen on kaikkien viiden osalta samantapaista (Kuva 15), niissä ollen kuitenkin attribuuttien osalta pieniä eroja. Vaadittavan lyhytnimi-attribuutti on järjestelmän sisäisesti käyttämä, eikä se näy missään, mutta otsikko-attribuutti näkyy.



Kuva 15. Valintalistan Tila-valintojen listaus (rollover-palkissa poista-ikoni)

Hallinnoija voi asettaa tietylle käsittelijälle tietyt niin sanotut "omat kategoriat" (Kuva 16), minkä vaikutus ilmenee käsittelijän käyttöliittymässä, käsittelijän ollessa omilla tunnuksillaan kirjautuneena ja valitessaan suodattimista "Näytä omat kategoriat" tai klikatessaan painonappia "Näytä omat". Yksiköiden kohdalla pätee sama.



Kuva 16. Käsittelijöiden asettamista kategorioille

3.4 Lomakkeiden virtaavuus – keskustelu käyttäjän kanssa

Kirjassa "Forms that Work: Designing Web Forms for Usability" (Jarrett ja Gaffney 2008) verrataan verkossa täytettävän lomakkeen käyttöä ihmisen kanssa keskusteluun, joka voi keskeytyä esimerkiksi sen vuoksi, että esitetään vaikea kysymys (Hamill 2010). Muita syitä lomakkeiden täyttämisessä esiintyviin vaikeuksiin on lukuisia, joista useat liittyvät elementtien asemointiin, ryhmittelyyn, nimeämiseen ja lomakkeen tuttuuden asteeseen. Myös "tuntemattoman pelko" (Dawson 2010) eli tietämys siitä, että on vastuussa omista valinnoistaan, voi johtaa lomakkeen täyttämisen vaikeuteen, jopa sen täyttämättä jättämiseen.

Uuden tukipyyntöjenkäsittelyjärjestelmän käyttöliittymissä lomakkeet ovat aina kokonaisuudessaan esillä, jokaisen niistä ollessa tarkoitettu käytettäväksi johonkin tiettyyn rajattuun tarkoitukseen. Useimmat käyttäjät halunnevat silmäillä esillä olevan lomakkeen kertaalleen läpi ymmärtääkseen sen tarkoituksen ja laajuuden, joten mitä pienemmälle alueelle suhteessa koko sivuun lomake mahtuu, sen parempi (Penzo 2006a). Jos lomake levittäytyisi kovin laajalle alueelle, voisi silmän sakkadiliikkeitä monitoroivissa tutkimuksissa ilmetä samantapaista, mitä on havaittu tekstipainotteisten verkkosivujen kohdalla eli tietyt osat havaittavasta kohteesta jäisivät erittäin vähälle huomiolle (katso muun muassa Jakob Nielsen's Alertbox 2006 ja Hudson ym. 2005).

Silläkin on paljon merkitystä, onko lomake sen käyttäjälle jo ennestään tuttu. Ensikäyttäjät silmäilisivät lomaketta paljon pidempään kuin kokeneemmat käyttäjät, joiden katse kiinnittyy heti niihin kohtiin, joiden he tietävät olevan oleellisia aikomuksensa läpiviemisen kannalta (Penzo 2006a).

Lomakkeen kompaktisuus vaikuttaa siihen, kuinka nopeaa sen visuaalinen navigointi on, kuten myös täytettävien kenttien selkeänimisyys (Penzo 2006a). Selkeä ja kuvaava kentän nimi auttaa käyttäjää tunnistamaan kentän tarkoituksen ja vähentää käyttäjän tarvetta vakuuttaa itselleen (Penzo 2006a), että hän on syöttämässä kenttään sitä tietoa, mitä hänen halutaan syöttävän.

Matteo Penzo (2006b) kertoo UXmatters-verkkojulkaisussa koejärjestelystä, jossa analysoitiin kenttien nimikkeiden sijainnin vaikutusta sakkadiaikoihin eli siihen kuinka kauan käyttäjän katse viipyy jossain kohdin lomaketta ennen kuin käyttäjä ymmärtää, mitä hänen on tarkoitus tehdä ja huomaa mitkä kenttien nimikkeet ja kentät kuuluvat yhteen.

Sakkadiaikoihin vaikutti jopa kymmenkertaisesti pelkästään se, että kenttien nimikkeet sijoitettiin kenttien vasemmalta puolelta (vasemmalle tasattuna) niiden yläpuolelle (Penzo 2006b). Testikäyttäjinä olleista noviisikäyttäjistä ja kokeneimmista käyttäjistä kaikki havaitsivat saman tien yhteyden kentän ja sen yläpuolella nimikkeen välillä. Olennaista tällöin oli vain se, että kentän ja nimikkeen välillä ei saanut olla liikaa tyhjää väliä, koska tällöin käsitys samaan ryhmään kuuluvuudesta hämärtyi.

Hyvinkin pienillä seikoilla voi olla merkitystä mahdollisten käyttäjässä esiintyvien hämmennystilojen ennaltaehkäisemiseksi. Jos täytettävän kentän pituus on huomattavan paljon pidempi kuin mitä voisi olettaa siihen olevan tarkoitetun kirjoitettavan, käyttäjästä voi tuntua, että hän ei ole ymmärtänyt kentän tarkoitusta oikein (Appleseed 2010). Tämä pätee erityisesti sellaisiin kenttiin, joiden nimikkeenä ei ole mikään tavanomainen kuten "nimi" tai "otsikko".

Sitäkin voi pitää todennäköisenä tapahtumana, että käyttäjän täytettyä lomakkeen lähes valmiiksi asti, mutta ollessaan juuri sillä hetkellä liian tietämätön siitä, mitä johonkin jäljellä olevaan kenttään pitäisi vastata, hän voi vastata siihen vähemmällä tarkkuudella tai suorastaan huolimattomasti. Tukipyyntöjenkäsitte-

lyjärjestelmän tapauksessa käyttäjän lomakkeen tarkoitus on saada ohjattua käyttäjä antamaan niin tarkka kuvaus tukipyynnön kohteesta ja sijainnista kuin mahdollista, sillä tukipyynnön käsittelyn kannalta olisi haitallista, jos tukipyynnön kohdetta ei voisi paikallistaa annettujen tietojen pohjalta. Myös raportointi- ja tilastointitarpeita varten eksakti ja yksiselitteinen tieto on käyttökelpoisempaa ja helpompaa käsitellä automatisoidusti kuin suurpiirteinen tekstuaalinen kuvailu.

Ensikäyttäjillä voi heille uusien lomakkeiden edessä olla tarve varmistella useampaan kertaan annettujen tietojen oikeellisuus, mutta tästäkin huolimatta heillä saattaa olla jopa halu (tai taipumus) olla lukematta lomakkeen yhteydessä olevaa ohjeistusta. Tämä asettaa lisää haasteellisuutta lomakkeen suunnittelulle – varsinkin jos ohjeisiin haluttaisiin sisällyttää jotain lomakkeen täyttämisen kannalta olennaista informaatiota (Addicott Web 2009).

Lomakkeiden suunnittelussa joudutaan usein tasapainottelemaan sopivan korostuksen ja liikaa huomiota vievien elementtien välillä. Joidenkin kenttien täyttäminen voidaan haluta asettaa pakolliseksi, mutta sitä ei haluta tehdä liian korostetusti. Toisaalta, osaa käyttäjästä saattaisi miellyttää, ehkä vain kyseisellä hetkellä, erityisen hyvin erottuvat pakollisiin kenttiin viittaavat merkinnät. Pakollisia kenttiä voi merkitä esimerkiksi jollakin merkillä kuten tähdellä (*) tai paksuntamalla kentän nimikkeen.

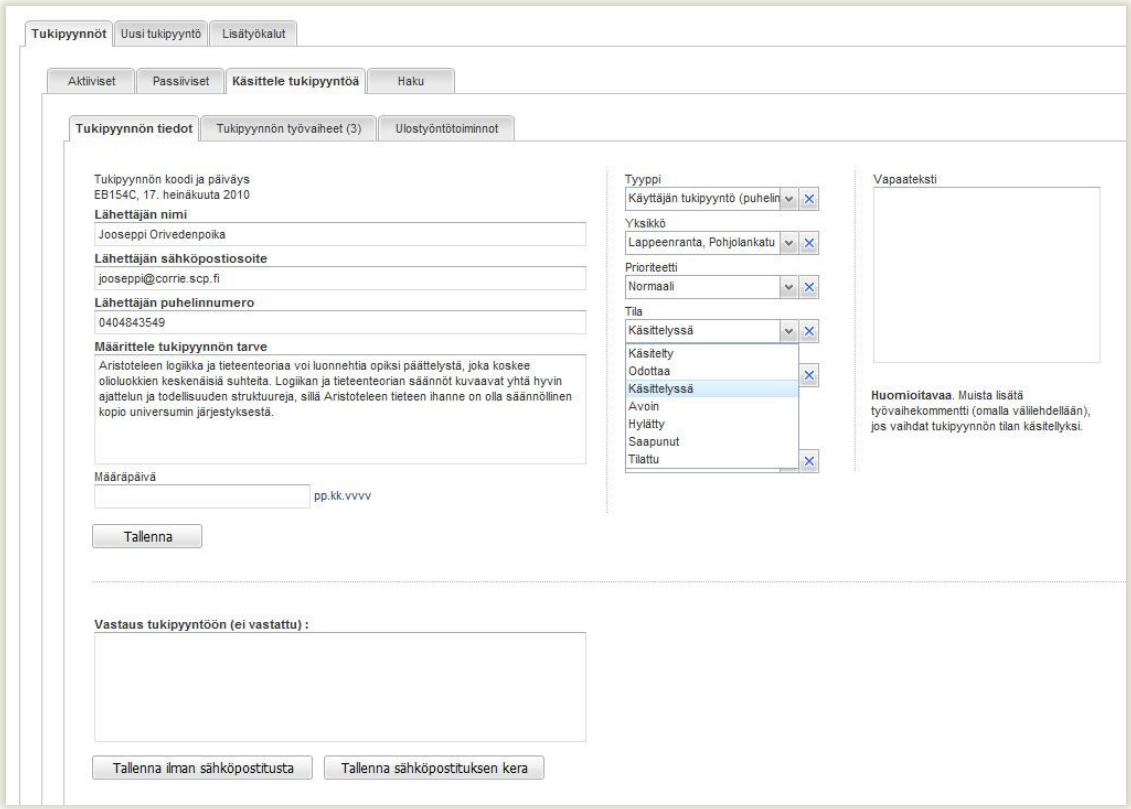
Nimikkeen paksunnos ei aina ole hyvä korostuskeino, sillä se heikentää luettavuutta ja paksunnettu nimike saattaa sekoittua lomakkeen ympäristössä tai muualla lomakkeella esiintyviin visuaalisiin elementteihin. Matteo Penzon *ad-hoc*-tutkimus (2006b) osoitti myös sen, että nimikkeiden paksunnoksesta seurasi jopa 60 %:n nousu sakkadiajoissa (katseen siirtyminen nimikkeestä sitä vastaavaan kenttään).

Toisaalta Luke Wroblewski, joka on muun muassa kirjoittanut kirjan "Web Form Design: Filling in the Blanks", havaitsi tietynlaisen lomakkeen kohdalla olevan suositeltavaa lisätä nimikkeiden visuaalista painoarvoa (eng. visual weight) paksuntamalla niitä, koska muuten ne saattavat joutua kilpailemaan huomiosta kenttien itsensä kanssa (Wroblewski 2005). Kyseisessä tapauksessa kenttä oli upotetun näköinen: yläreuna ja vasen reuna mustalla värillä, ja alareuna ja oikea reuna harmaalla.

Cxpartners on tehnyt omia tutkimuksiaan (Chui 2009), myös Penzon ja Wroblewskin kokeiluihin tutustuneena, joissa ilmeni, että kaikki ihmiset eivät käsitä tietyllä tavalla pakolliseksi merkittyä kenttää samalla tavoin (joku ei ymmärtänyt tähden merkitystä, ja joku toinen ei ymmärtänyt kursiivin merkitystä). Suositukseksi jäikin, että optionaaliset kentät merkittäisiin kenttien sisään sanalla "optionaalinen", harmaalla värillä.

Aihetta käsittelevässä sakkaditutkimuksessa (Penzo 2006b) tuodaan usein esille se huomio, että valintalistat ovat herkimmin käyttäjän huomion kiinnittävä yksittäinen lomake-elementti. Sellaisen kerrottiin olevan aina (sijaintiriippumattomasti) se elementti, jonka käyttäjä ensinnä lomakkeelta huomaa.

Tukipyyntöjenkäsittelyjärjestelmän käsittelijän käyttöliittymässä oleva tukipyyntöjen muokkaukseen tarkoitettu lomake (Kuva 17) sisältää runsaasti erilaisia lomake-elementtejä, joiden sijoittelussa on pyritty ottamaan huomioon muun muassa tietty ilmavuus, reunojen tasaus ja ryhmittely. Pakollisten kenttien osoittaminen on tehty paksunnoksella ja täytettävät kentät on pakattu melko tiiviiksi ryhmäksi. Kokonaisuus on käsittelijöiden itsensä kertoman mukaan toimiva, eikä demokäytön aikana ole kerrottu tapahtuneen virheitä esimerkiksi virhetulkintojen tai tietojen vääriin kenttiin merkitsemisien vuoksi.



Kuva 17. Tukipyyntö haettuna muokattavaksi

Ongelmalliseksi loogisuuden kannalta muodostui sen osoittaminen, että kentät "Lähettäjän sähköpostiosoite" ja "Lähettäjän puhelinnumero" ovat keskenään vaihtoehtoiset eli riittää, kun kirjoittaa tiedon toiseen niistä. Käsittelijän, joka on vain pienen käyttäjäryhmän käytettävissä, on vielä mahdollisuuksien rajoissa hyväksyä se, että "se pitää vain muistaa", mutta käyttäjän käyttöliittymässä asia pitäisi ratkaista jotenkin toisin.

Tallenna-painikkeen painamisen jälkeen epävalideista tai puuttuvista tiedoista kerrotaan tietynlaista ikonia symbolina käyttäen, joten sinänsä ei ole vaaraa, että tietokantaan asti päätyisi epäkelpoja tukipyyntöjä, mutta toisaalta siinä rikotaan Jacob Nielsenin erästä käyttöliittymäsuunnittelun "heuristista" periaatetta (Nielsen 2005): virheiden ennaltaehkäisyä. Sama ongelma on läsnä muissakin kohdin tukipyyntöjenkäsittelyjärjestelmän eri käyttöliittymiä ja kehityksen jatkovaiheissa olisi hyödyllistä ratkaista ne jollain käyttäjää häiritsemättömällä tavalla.

Erikoisimpana piirteenä lomakkeella on Määräpäivä-kentälle asetettu maski (pp.kk.vvvv), joka pakottaa lomakkeen käyttäjän syöttämään kyseiseen tietokenttään pelkkiä numeroita. Numeroiden paikkaa osoittavat apuviivat "__.__.____" tulevat esille vasta siinä vaiheessa, kun klikataan Määräpäivä-kentän kohdalla, jottei ilmenisi tarpeetonta visuaalisella informaatiolla kuormittamista (Wroblewski 2005). Pisteitä ei tarvitse kirjoittaa erikseen, vaan ne täydennetään automaattisesti. Maski-ominaisuus sisältyy SmartGWT:n lomakekenttien vakio-ominaisuuksiin.

Tällä tavoin ehkäistään se, että lomakkeen käyttäjä kokisi turhautuneisuutta (Crescimanno 2005) siitä, että joutuisi noudattamaan jotain tarkkaa kaavaa päivämäärän ilmaisemiseksi (ei tarvitse etsiä näppäimistöltä muita merkkejä kuin numeroita), sekä varmistutaan siitä, että tieto päätyy tietokantaan tietyssä muodossa (Forman 2001).

Tukipyyntöjenkäsittelyjärjestelmän käyttöliittymissä on pyritty kiinnittämään huomiota yleiseen esteettiseen olemukseen, sillä ensivaikutelmilla on erittäin suuri merkitys sen suhteen, kuinka mielellään käyttöliittymiä käytetään. Tutkimuksen mukaan käytettävyyden parantaminen ei tehnyt vetovoimaltaan vaisua sivustoa yhtään mielekkäämmäksi käyttää, vaikka käytettävyyden parannus tekikin jonkin tarpeen suorittamisen helpommaksi testatulla sivustolla (Phillips & Chaparro 2009).

Yksittäisenä lomakkeiden käytettävyyttä parantavana tekona voisi olla painikkeiden korostaminen jollain värillä. Tutkimuksen (Wroblewski ja Etre 2007) mukaan moni testihenkilöistä koki värillisten painikkeiden käytön tehneen niiden havaitsemisen helpommaksi. Toisaalta joukossa oli myös testihenkilöitä, joiden mielestä värilliset painikkeet saivat käyttäjän varmistamaan ylimääräisen kerran, että on varmasti painamassa oikeaa painiketta.

SmartGWT:n IButton-tyyppisten painikkeiden väriä ei voi muuttaa pelkästään värikoodia vaihtamalla, vaan täytyisi muokata kuvatiedostoja, joista painike varsinaisesti kootaan, mutta sopivasti värjättyjen painikkeiden käyttäminen voisi hyvinkin olla toimiva ja selkeä ratkaisu. Sitä tosin on hankalampi määrittää, mikä on kenenkin mielestä sopivaa, koska ihmisten mieltymyksissä on aina jonkin verran eroavaisuuksia.

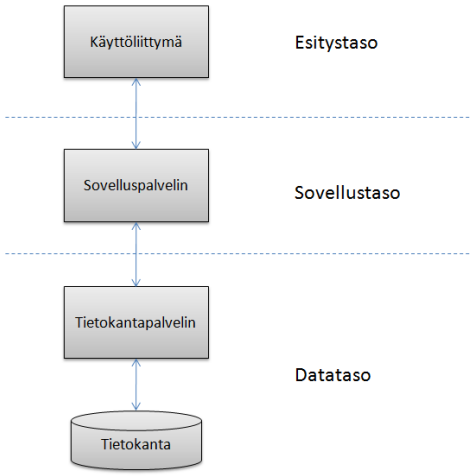
4 TOTEUTUKSEN YKSITYISKOHTIA

Seuraavissa alaluvuissa nostetaan tarkasteltavaksi muutamia koko järjestelmään vaikuttavia yksityiskohtia.

4.1 Tietokannasta

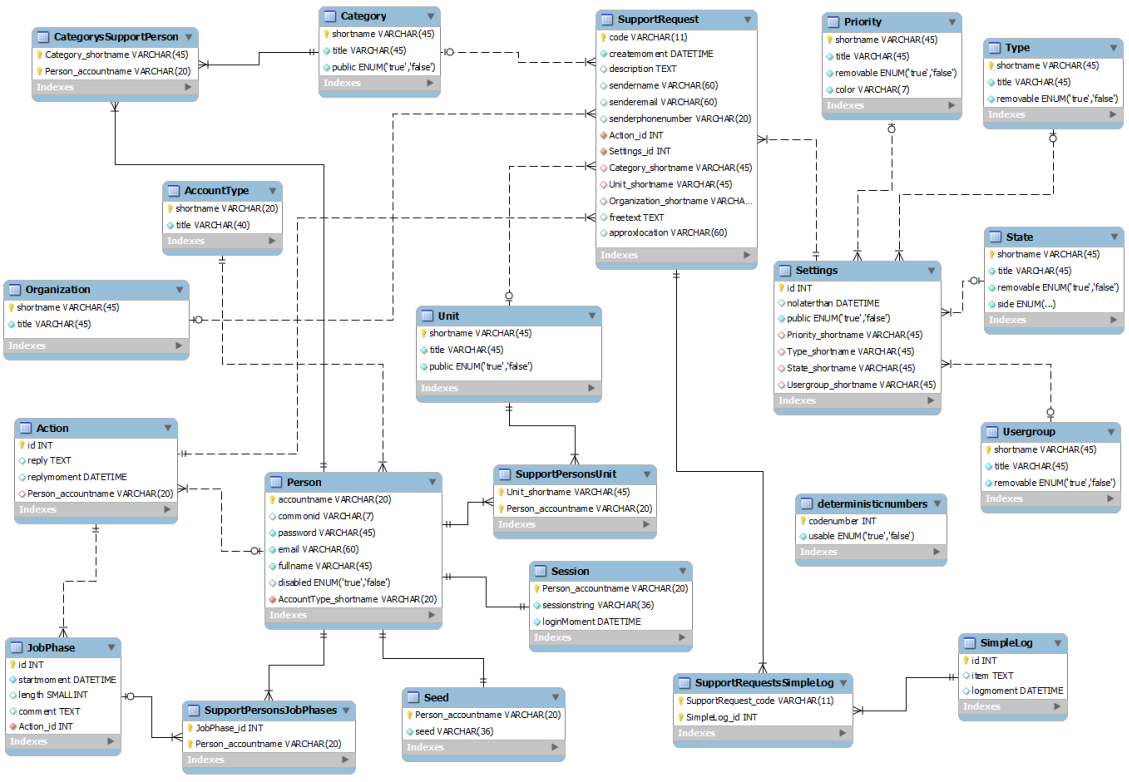
Räätälöityjen järjestelmien (Hovi ym. 2005, 15) tietokannat, kuten tässä opinnäytetyössä toteutetun tukipyyntöjenkäsittelyjärjestelmän tietokanta, suunnitellaan tarkasti juuri tietylle käyttäjäryhmälle. Niiden rakenne tehdään räätälöidysti, ne ovat selkeitä, niiden taulujen ja sarakkeiden nimet nimetään selkeästi tutuille termeillä, sekä niissä on pyritty hyvään ohjelmointimukavuuteen. Tämä kuvaus täsmää hyvin toteutettuun järjestelmään.

Tietokantapalvelin itsessään on osa kolmitasorakennetta (eng. 3-tier, kuva 18), jonka muut osat ovat esitystaso, jonka toiminnasta vastaavat enimmäkseen SmartGWT:n komponentit, ja sovellustaso, joka koordinoi yhteyksiä tietokantaan ja käsittelee esitystasolta saapuvat käskyt. Kolmitasorakenteessa esitystaso ei koskaan ole suoraan yhteydessä tietokantaan, vaan aina sovellustason kautta. Kyse on siis operatiivisen järjestelmän tietokannasta eli tietokannasta, josta luetaan ja siihen tallennetaan tietoa reaaliajassa, käyttöliittymissä suoritettujen toimintojen pohjalta.



Kuva 18. Tietokantapalvelin osana kolmitasorakennetta

Tasorakenteeseen sisältyy idea, että jokaisella kerroksella sijaitseva järjestelmän osa voidaan periaatteessa vaihtaa toiseen vastaavanlaiseen muista kerroksista riippumatta. Täten tietokantakin voitaisiin vaihtaa jonkin toisentyyppiseen (valintaperusteista lisää alaluvussa 2.3, "Tietojärjestelmän laitteisto- ja ohjelmistoympäristöstä" ja 6.6, "Ohjelmistojen lisenssit ja niiden yhteensovittaminen"). Käytännössä tämän mahdollistaa se, että järjestelmä käsittelee SQL92-standardin implementointiin perustuvaa MySQL-tietokantaa JDBC-ohjelmointirajapinnan (Java Database Connectivity) kautta. Ilman tätä rajapintaa jouduttaisiin tietokannan vaihdoksen yhteydessä (esimerkiksi MySQL:lästä Microsoft SQL Serveriin) muuttamaan tietokantatransaktioista vastaavaa ohjelmointikoodia. Tukipyyntöjenkäsittelyjärjestelmän tietokannan tauluille käytetään vain yhtä tietokantaa.



Kuva 19. Järjestelmän käyttämän tietokannan relaatiomalli

Ennen kuin tietokannan relaatiomallin mukainen rakenne (Kuva 19) oli saavutettu, oli käytävä läpi tietokannan suunnittelun muita vaiheita (muun muassa käsiteanalyysi, käsitekaavion tekeminen ja normalisointi). Kirja "Tietokantojen suunnittelu ja indeksointi" kutsuu näiden vaiheiden muodostamaa kokonaisuutta suunnitteluputkeksi (Hovi ym. 2005, 24). Vaiheet eivät seuraa toisiaan perätys-

ten, vaan työtä tehdään "spiraalin omaisesti tai iteratiivisesti" (Hovi ym. 2005, 25). Näin on käytännössä tapahtunutkin, sillä itse järjestelmän suunnittelun ja toteutuksen aikana ideat jalostuivat sekä uusia ideoita syntyi, mikä vaati täydentämään tietokannan rakennetta tai tauluja tietyltä osalta.

Käytännössä muutokset eivät vaatineet purkamaan relaatiomallia tai tekemään sitä kokonaan alusta uudelleen, mikä kertoo vähintään melko hyvästä ensimmäisten vaiheiden läpikäymisestä ja asiakkaan tarpeiden kuuntelusta.

Ensimmäisessä asiakastapaamisessa toteutettavaan järjestelmään kohdistuvista tarpeista ja odotuksista sai alustavan käsityksen, mutta vasta toisella kerralla pystyi olemaan valmistautuneempi sellaisten kysymysten esittämisellä, joilla pyrittiin kartoittamaan käsitteitä, joilla tietokannan tauluja nimetään. Tätä vaihetta kutsutaan käsiteanalyysiksi. Tämä toinen tapaaminen tapahtui asiakkaan työtiloissa, mikä mahdollisti heidän toimiensa seuraamisen heidän omissa ympäristöissään, antaen opinnäytetyön tekijälle omakohtaisen yleistuntuman käyttäjistä, heidän toimistaan ja käyttöympäristöistään (Hyysalo 2006, 100).

Käsiteanalyysissä kuvataan pääpiirteissään kolmenlaisia asioita: käsitteitä, tietoja ja yhteyksiä (Hovi ym. 2005, 35). Tukipyyntöjärjestelmän tapauksessa yhteys on esimerkiksi käsitteiden 'tukipyyntö' ja 'kategoria' välillä sekä käsitteiden 'kategoria' ja 'käsittelijä' välillä. Monilukuisuudella täsmennetään käsitteiden välistä suhdetta: yhdellä tukipyynnöllä voi olla vain yksi kategoria, mutta yksittäisellä kategorialla voi olla useita käsittelijöitä.

Se miten monilukuisuutta kuvataan kaaviossa, riippuu siitä, mitä notaatiomallia on käytetty (voidaan ilmaista numeroin tai kuvioin). MySQL Workbenchissä, jota käytettiin tietokannan relaatiomallin tuottamisessa, on käsitteiden välisiä suhteita kuvaamaan valittavissa muun muassa notaatiomallit IDEF1X ja Information Engineering, jota myös harakanvarpaiksi (Crow's Foot) kutsutaan niiden ulkonäön vuoksi. Kuvassa oleva relaatiomalli käyttää Information Engineering -notaatiota.

Projektin myöhemmissä vaiheissa ilmeni asiakkaan puolelta halukkuutta ottaa käyttöön monimutkaisempia käsitteiden välisiä suhteita kuin mikä oli alun perin katsottu riittäväksi. Kuvassa oleva relaatiomalli noudattaa alkuperäisen konsen-

suksen mukaista rakennetta. Uusien ehdotusten mukaan olisi olemassa kategorioita, jotka olisivat aina sidoksissa johonkin tiettyyn yksikköön, mutta myös kategorioita, jotka eivät liittyisi mihinkään yksikköön erityisesti.

Kun noita kahta yksinkertaiselta kuulostavaa tarvetta ajattelee sellaisenaan, niin ne kuulostavat siltä kuin niiden voisi ajatella tulevan mieleen ja huomioiduksi hyvinkin helposti, minkä lisäksi ne olisivat varsin helposti huomioitavissa tietokannan rakenteessakin. Käytännössä tällaista ei projektin alkuvaiheilla huomioitu ja projektin loppuvaiheilla muutos olisi vaikuttanut käyttöliittymien näyttöjen ja yleisen käytettävyyden suunnittelemiseen uusiksi moniltakin osin. Katso tarkemmista perusteluista käyttöliittymien suunnitteluun luvusta 4 ("Käyttöliittymien suunnittelun filosofia") ja alaluvusta 2.2 ("Asiakkaan tarpeiden kartoittamisesta ja siinä onnistumisesta").

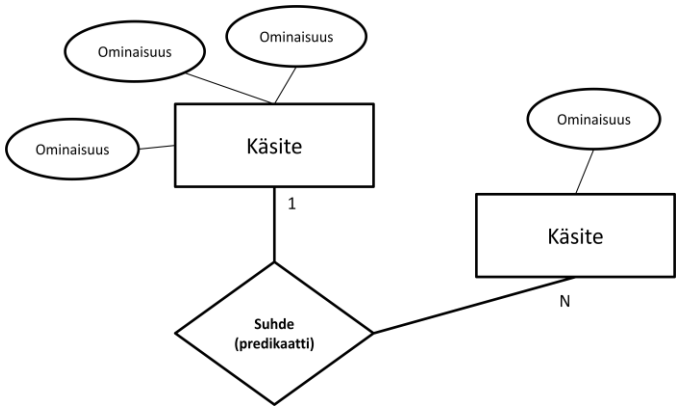
Järjestelmän kehitystyötä saatetaan jatkaa Projektityö-kurssin puitteissa, mutta varsinaisen ongelman voi nähdä syntyneen mahdollisesti sen vuoksi, että käsiteanalyysin jälkeen ei suoritettu riittävän tarkkaa tarveanalyysiä eli tarkistettu käsitemallia testaamalla sitä tiedossa olevilla tietotarpeilla (Hovi ym. 2005, 80). On tulkinnanvaraista, voiko tämän määrittää jonkun tietyn henkilön syy, mutta nähtävissä on ainakin kaksi mahdollista aiheuttajaa, joista toinen on se, että opinnäytetyön tekijä ei saanut pyytämiään esimerkkejä aiempaan tukipyyntöjärjestelmään syötetyistä tukipyynnöistä. Toinen voi liittyä siihen, että asiakkaan puolella tapahtui miehistön vaihdos heinäkuun alussa. Tarve relaatioiden muutokselle tuli esille vasta elokuun lopulla, jolloin kehitystyötä oli jatkunut jo lähes kaksi kuukautta.

Eräänä tietokannan muodostamisperusteena oli pyrkimys luettavuuteen ja hyvään nimeämiskäytäntöön. Taulujen nimissä ei saanut esiintyä monitulkintaisuutta, eikä käytettyjen käsitteiden välillä saanut olla päällekkäisyyttä. Tietokannan relaatiomallista piti saada nopeasti selkoa sellaisenkin, joka ei sitä aikaisemmin ollut nähnyt.

Pyrkimys kolmannen normaalimuodon mukaisuuteen, joka sisältää muun muassa pyrkimykset tiedon toisteisuuden välttämiseen, tiedon päivityksen tekemiseen vain yhteen paikkaan ja taulujen sarakkeiden (tietoalkiot) olemiseen funktionaalisesti riippuvaisia vain perusavaimesta (jokaista tietoalkiota kohden olisi

vain yksi perusavain, esimerkiksi yhtä sähköpostiosoitetta kohden vain yksi käyttäjätunnus, joka toimisi perusavaimena).

Tietokannan relaatiomallia edelsi suunnitteluputken mukaisesti käsitemalli, joka oli käytännössä ruutupaperille piirretty. Tämä vaihe suoritetaan yleensä vaihtoehtoisesti joko UML:n luokkakaavioita käyttäen tai "salmiakkikuvioiduin" ER-kaavioin, joista jälkimmäisestä esimerkki kuvassa 20.



Kuva 20. ER-kaavio muodostuu vähintään elementeistä käsite, suhde ja ominaisuus, sekä monilukuisuusmerkinnöistä

ER-kaavio on erittäin nopeakäyttöinen työväline, vaikkakin luettavuuden kannalta se voi paikoitellen olla epäkäytännöllinen, sillä siinä ei esimerkiksi käytetä nuolia kuvaamaan sitä, miten päin käsite-suhde-käsite -yhteys luetaan. Suhdetta kuvaavia predikaatteja voi olla yllättävän hankala keksiä ja niissä päätyykin helposti käyttämään esimerkiksi verbiä "on", kuten lauseessa "monella persoonalla on yksi tilityyppi". Toisinpäin luettuna se olisi "yhdellä tilityypillä on monta persoonaa". Tuollaisessa tapauksessa voisikin olla kuvaavampaa käyttää predikaattina "voi käyttää", jolloin lause kuulostaisi molemminpäin suhteellisen järkevältä.

Näitä predikaatteja ei varsinaisesti käytetä missään myöhemmissä vaiheissa, mutta niistä on kuitenkin apua, jos järjestelmän tietotarpeista keskustelee jonkun toisen kanssa. Luokkakaavioissa käsitteiden välisissä yhteyksissä voi määrittää roolin, jonka käsite ottaa jonkin tietyn yhteyden tapauksessa. Käsitteiden persoona ja tilityyppi välillä voisi tällöin olla roolit "käytettävä" (tilityyppi) ja "käyttäjä" (persoona). Varsinainen relaatiomalli tehdään ER-kaavion tai luokkakaavion pohjalta.

Mainitun normalisoinnin vastakohtana on denormalisointi, joka tarkoittaa tahallista tiedon toistamista jonkin perustellun syyn nojalla. Syynä voi olla taulujen yhdistely luettavuuden tai tietokantakyselyissä tarvittavien taulujen liitosten vähentäminen. Luettavuus viittaa tässä sekä tietokannan relaatiomallin luettavuuteen, että mahdolliseen tarpeeseen lukea tiedolla täytetyn tietokannan taulun rivejä sellaisenaan.

Taulussa SupportRequest on käytetty denormalisointia. Se sisältää lähettäjän yhteystietoihin liittyviä sarakkeita, joissa tulee varmuudella olemaan toistuvia tietoja. Ne voisi niin halutessa sijoittaa omaankin tauluunsa ja jättää niiden tilalle jonkin yksittäiseen käyttäjään viittaavan koodin. Sähköpostisoite ei yksilöisi käyttäjiä, koska kaikilla ei olisi sitä. Johtuen hiukan myös ohjelmointimukavuudellisista tekijöistä, mutta varsinaisesti siitä, että käyttäjiä ei yksilöidä lähetysvaiheessa minkään tunnistusmekanismin avulla, on tukipyynnön lähettäjän tiedot annettu olla kyseisessä taulussa. Tilankäytön kannalta aiheutuva toisteisuus on niin vähäistä, ettei sillä ole mitään merkitystä.

Järjestelmä on toteutettu siten, että monivaiheiset tietokantatransaktiot (SQL-lauseiden sarjat) eivät koskaan aiheuta pysyviä muutoksia, jos yksikin transaktion vaiheista epäonnistuu. Tietokantaan ei sallita talletettavan mitään, mikä voisi aiheuttaa SQL-injektion mahdollisuuden. JDBC-ajuriin itseensä sisältyy omat tarkistusrutiinit, mutta tämän lisäksi kaikki tietokantaan kohdistuva data kuljetetaan tarkistuslogiikan kautta (ohjelmointikoodissa ParameterChecker-luokka).

Kehitysvaiheessa käytetty MySQL:n InnoDB-tietokantamoottori tarjoaa niin sanotun doublewrite-moodin yksittäisten tietokannan muokkauskertojen onnistumisen varmistamiseksi, mutta varsinainen täysvarmistus on aiottu tehdä dumppaamalla (viittaa komentoriviltä ajettavaan mysqldump-käskyyn) koko tietokanta ajoittain jonnekin talteen. Tämä on asiakkaan esittämä ehdotus (katso myös alakohta 2.5, "Asiakkaan tarpeiden kartoittamisesta ja siinä onnistumisesta").

MySQL:n tukemien metodien lisäksi ei ollut tarvetta luoda omia funktioita tai proseduureja. Indeksejä tauluille ei ole asetettu niiden perusindeksien lisäksi, mitä MySQL Workbench generoi käytettäessä Forward Engineer -toimintoa, joka luo varsinaiset tietokannan taulujenluontilauseet. Käytännössä se luo in-

deksit perusavaimille ja viiteavaimille. Viiteavaimilla varmistetaan tiedon eheys (poistettaessa rivejä ei jää orpoja tietoja ja päivitettäessä tietoa päivittyy myös viitattuun tauluun).

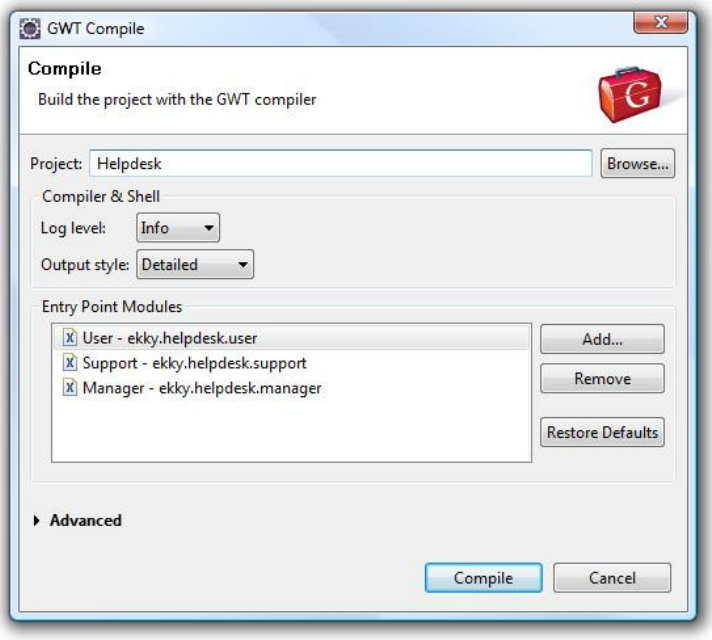
```
CREATE  TABLE IF NOT EXISTS `helpdesk`.`JobPhase` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `startmoment` DATETIME NOT NULL ,
  `length` SMALLINT NULL DEFAULT 0 ,
  `comment` TEXT NULL ,
  `Action_id` INT NOT NULL ,
  PRIMARY KEY (`id`) ,
  INDEX `fk_JobPhase_Action1` (`Action_id` ASC) ,
  CONSTRAINT `fk_JobPhase_Action1`
    FOREIGN KEY (`Action_id` )
    REFERENCES `helpdesk`.`Action` (`id` )
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB;
```

4.2 Valmiin järjestelmän käsittelystä

SmartGWT-sovelluskehyksessä kaikki ohjelmointikoodi kirjoitetaan Java-ohjelmointikielellä, mutta vietäessä lähdekoodi automaattitoimintaisen kääntäjän läpi, muodostuu lopputuloksena ajettava sovellus, jonka palvelinpuolen koodi on edelleen Javaa, mutta asiakaspuolella (selaimen puoli) ajettava ohjelmointikoodi on konvertoitu JavaScriptiksi. Samankuuloisista nimistään huolimatta näillä kahdella ohjelmointikielellä ei ole paljonkaan yhteistä, minkä vuoksi aivan kaikkia Javassa käytettävissä olevia luokkakirjastoja ei voida käyttää, mutta suurta osaa pyritään emuloimaan (Google 2010c) JavaScript-koodissa.

Käytännössä oli huomattavissa, että emuloituja Java-ohjelmointikielen metodeita ja tietorakenteita on moniin tarpeisiin niin paljon, ettei ohjelmointivaiheessa ollut edes tarpeen edes ajatella JavaScriptiä. Tietyissä tarpeissa, kuten tietoa salatessa (sisäänkirjautuminen), oli kehitettävä emuloinnista muodostuvan rajoitteen kiertävä ratkaisu, koska MessageDigest-luokka, joka tarjoaisi MD5- ja SHA1-salausalgoritmin käyttöön, oli käytettävissä vain palvelinpuolen koodissa. Käytännössä ongelma ratkaistiin etsimällä Internetistä vapaasti käytettävissä oleva SHA1-algoritmin Java-kielinen implementaatio, joka käytti vain käytettävissä olevia metodeita ja tietotyyppejä.

Käyttäjäryhmäkohtaisten moduulien käytön johdosta lähdekoodin kääntämistä ei tarvitse tehdä koko sovellukselle kerrallaan, vaan sen voi tehdä myös moduulikohtaisesti. Kääntämisen voi suorittaa Eclipse-kehitysympäristöön asennetun Google Web Toolkit -pluginin sisältämän graafisen käyttöliittymän tarjoavan kääntäjän (Kuva 21) avulla.



Kuva 21. SmartGWT-sovellukset käännetään GWT:n kääntäjällä

Kääntämisen jälkeen jokaista moduulia vastaa oma hakemistonsa (*ekky.helpdesk.manager.Manager*, *ekky.helpdesk.support.Support* ja *ekky.helpdesk.user.User*), jotka sisältävät lukuisia isokokoisia tiedostoja, joiden nimestä ei voi suoraan päätellä mitään (esim. *1B350FC43C25D589AC767C9.cache.html*), mutta näitä tiedostoja ei ole tarkoitettukaan muokattavaksi.

Kutakin moduulia varten on oma käynnistystiedostonsa (HTML-tiedosto), johon voi niin halutessaan lisätä esimerkiksi tyylimäärittelyjä (CSS), joihin voidaan viitata myös SmartGWT:n komponenteista käsin. Tukipyyntöjenkäsittelyjärjestelmän tapauksessa täydentäviä tyylimäärittelyä tehtiin eräässä vaiheessa Internet Explorerilla esiintyneiden visuaalisten poikkeavuuksien (komponenttien asemointi, reunukset, marginaalit, jne.) korjaamiseksi. Käytännössä kaikki nuo poikkeavuudet korjaantuivat (katso alakohta 7.4.1, "Toiminnallisuustestaus ja

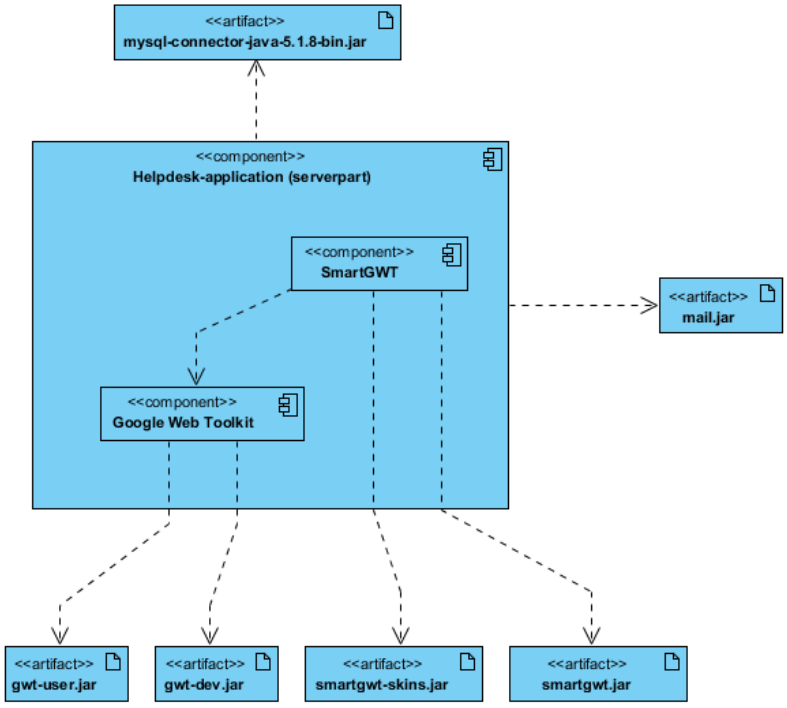
käyttöliittymien testaus") sillä, että käynnistäviin HTML-tiedostoihin lisättiin seuraava META-tagi:

<meta http-equiv="X-UA-Compatible" content="IE=7">

Tämä ratkaisuvaihtoehto tuli ilmi käyttöliittymien testauksen yhteydessä, mutta sitä voi pitää suositeltavana vaihtoehtona vain siinä tapauksessa, että Internet Explorerin versiota 7 uudempien selainversioiden tarjoamia ominaisuuksia ei ole tarpeen hyödyntää. SmartGWT:n tulevien versioiden kohdalla tarve kyseisen tagin käytölle voi olla jo poistunut ja komponentit näyttävät kaikilla yleisimmillä selaimilla samanlaisilta ilman ylimääräisiä korjaustoimenpiteitä.

Muita käännöshakemistoon muodostuvia hakemistoja on *WEB-INF*-hakemisto, johon sijoitetaan server- ja shared-pakettien luokkia vastaavat käännetyt .class-tiedostot. Nämä ovat aitoa Javan tavukoodia.

Sovellus tarvitsee toimiakseen SmartGWT- ja Google Web Toolkit -komponentit neljän .jar-tiedoston muodossa (*smartgwt.jar*, *smartgwt-skins.jar*, *gwt-user.jar* ja *gwt-dev.jar*), mutta näiden (Kuva 22) lisäksi ei monia muita artefakteja tarvitakaan.



Kuva 22. Sovellus tarvitsee toimiakseen muutamia artefakteja (.jar-tiedostoja)

Kuvassa 22 on lähdetty olettamuksesta, että tietokantana käytetään MySQL:ä, jolloin JDBC-rajapinta voidaan toteuttaa käyttämällä MySQL Connector/J:tä (*mysq-connector-java-5.1.8-bin.jar*, versio 5.1 tai uudempi). Sähköpostin lähettämiseen käytetään JavaMail API:tä (*mail.jar*, versio 1.4.3 tai uudempi). Nämä kaksi .jar-tiedostoa sijoitetaan joko Tomcatin *common/lib* -hakemistoon tai sovelluksen *WEB-INF/lib* -hakemistoon.

Kun järjestelmän sovellusosa ensimmäisen kerran siirretään palvelimelle (esimerkiksi WinSCP:llä), riittää, että kaikki kääntäjän käännöshakemistoon luomat tiedostot kopioidaan Tomcatin *webapps*-hakemistoon tai muualle, josta se näkyy ulospäin. Kehitysympäristön kääntäjä tuottaa sovelluksen ohjelmointikoodin, hakemistorakenteen sekä liittää mukaan useimmat tarpeelliset kirjastot.

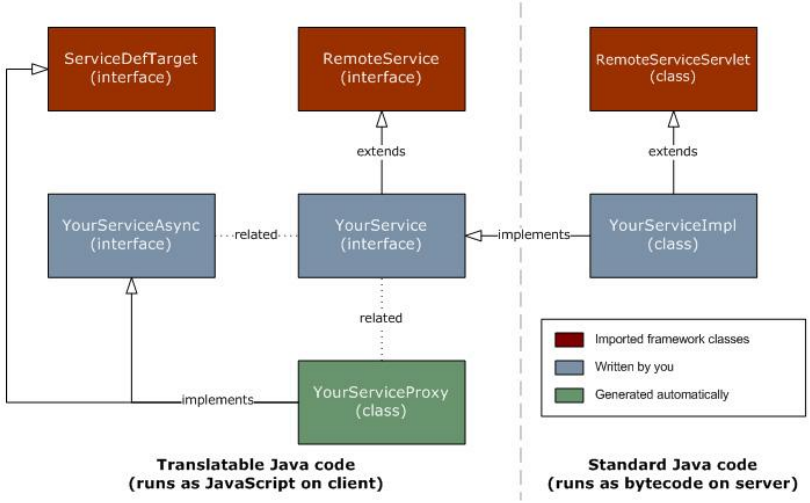
Myöhemmin kun tiedostoja siirretään vähäisemmissä erin paikallisesta kehitysympäristöstä verkkoon, on yksittäisten moduulien kohdalla käytännössä pakko siirtää kerralla kaikki isokokoiset, tiedostonimessään sekalaisesti kirjaimia ja numeroita kantavat tiedostot. Odotteluaikaa, jota isokokoisten tiedostojen siirtelystä kertyy, kasaantui projektin aikana paljonkin. Jokainen odottelukerta tuntui aina yhtä puuduttavalta, sillä lähes kaiken muun pystyi tekemään niin nopeasti kuin tekijän oma tehokkuus antoi myöten.

Palvelinpuolen tiedostoja voi siirtää yksitellenkin, koska niiden nimistä voi päätellä mitä Javan luokkaa mikäkin niistä vastaa. Sama pätee myös moduulien käynnistystiedostoihin (esimerkiksi *support.html*).

4.3 Etämetodien kutsuminen

Etämetodien käyttäminen on järjestelmän toiminnan kannalta olennaista, sillä järjestelmää ei käytännössä voi käyttää ilman asiakas- ja palvelinpuolen yhteentoimivuutta. Osa toiminnallisuudesta prosessoidaan palvelimella ja tietokannan käsittelytarpeet asettavat myös oman ehdottoman vaatimuksensa yhteyskanavan olemassaololle. Etämetodeiden kutsut ovat tilattomia (eng. stateless) eli palvelin kohtelee niitä kaikista muista kutsuista riippumattomista (kutsu itsessään sisältää parametreina kaiken tarvittavan etämetodin käyttämiseksi).

Asiakaspuolen luokista ei voi kutsua suoraan mitään palvelinpuolen luokkaa, vaan sellaisen kutsun on aina kuljettava tynkärajapintojen kautta. Nämä tynkärajapinnat toimivat etäkutsujen metodien rajapintakuvauksina, ollen toistensa vastinpareja. Jos tynkärajapintojen käyttöön ei aiemmin ole tutustunut, on todennäköistä, että niiden ymmärtäminen ei tapahdu hetkessä. Kun niiden toimintaidean lopulta hahmottaa, niiden käyttö alkaa tuntua luontevalta, eikä mikään seikka tunnu jääneen irralliseksi ja selittämättömäksi. Kuvassa 23 on kaavamainen esitys sovelluksen kommunikoimisesta palvelimen kanssa.



Kuva 23. Kaavamainen esitys sovelluksen kommunikoimisesta palvelimen kanssa etäkutsu-mekanismin (Google 2010b) välityksellä

Käytännössä palvelinpuolen luokka implementoi palvelimenpuoleisen tynkärajapinnan, asiakaspuolen kutsuessa asynkronisesti palvelimen metodeja kuten paikallisia metodejakin, sillä erotuksella, että lisäksi asetetaan kutsun jälkeen suoritettava takaisinkutsumetodi (eng. callback-method). Otettakoon esimerkiksi etämetodin kutsu hallinnoijan moduulista. Oletetaan, että hallinnoija oli käyttöliittymänsä kautta halunnut hakea tietyn käsittelijän tiedot käyttäjätunnuksen perusteella. Tämä vaatii käytettäväksi RequestProxy-luokan staattista metodia getManagercommand, jotta saadaan yhteys "kanavaan", jota pitkin etämetodia fetchAccountDetails kutsutaan.

```
RequestProxy.getManagercommand().fetchAccountsDetails("antony", callbackFetchAccount-
Details);
```

Huomioitavaa on, että metodin nimeä tullaan käyttämään samassa muodossa neljässä eri paikassa. Ensinnäkin sitä käytetään paikallisessa kutsussa, minkä lisäksi myös asiakaspuolen tynkärajapinnassa ManagerCommandStubAsync:

```
public interface ManagerCommandStubAsync {
  void fetchAccountsDetails(String accountname, AsyncCallback<LinkedHashMap<String,
String>> callbackFetchAccountDetails);
}
```

sekä sen palvelinpuolen vastineessa ManagerCommandStub:

```
public interface ManagerCommandStub extends RemoteService {
  LinkedHashMap<String, String> fetchAccountsDetails(String accountname) throws IllegalAr-
gumentException;
}
```

minkä palvelimen luokka ManagerCommandController implementoi:

```
public class ManagerCommandController extends RemoteServiceServlet implements Mana-
gerCommandStub {
  @Override
  public LinkedHashMap<String, String> fetchAccountsDetails(String accountname) throws Il-
legalArgumentException {
        // metodin sisältö
  }
}
```

Palvelinluokka voisi implementoida muunkin rajapinnan, mutta suunnitelullisena valintana on ollut sijoittaa etäkutsut moduulikohtaisiin "nippuihin". Etäkutsujen asynkronisuudesta johtuen sovelluksen toiminta jatkuu riippumatta siitä, missä vaiheessa yksittäisen kutsun käsittely palvelimella on.

Kutsun päätteeksi suoritetaan takaisinkutsu-metodi:

```
private AsyncCallback<LinkedHashMap<String, String>> callbackFetchAccountDetails = new
AsyncCallback<LinkedHashMap<String, String>>() {
  public void onFailure(Throwable caught) {}
  public void onSuccess(LinkedHashMap<String, String> accountDetailsMaps) {}
```

```
};
```

Yhteyskäytävänä toimivassa RequestProxy-luokassa olevaan määrittelyyn ei ole tarpeen puuttua lisättäessä uusia etäkutsumetodeja, vaan sen voi antaa olla tällaisenaan:

```
private static ManagerCommandStubAsync managercommand = (ManagerCommandStubA-
sync) GWT.create(ManagerCommandStub.class);
static { ((ServiceDefTarget) managercommand).setServiceEntryPoint(prefix +
"/ekky.helpdesk.manager.Manager/managercommand"); }
```

Todennäköistä on, että tynkärajapintojen vastinparillisuus hämmentää aluksi, koska ei ole ilmiselvää, miten nämä kaksi rajapintaa "näkevät" toisensa. Lopuksi oli tehtävä oleellinen Java Servlet -spesifikaatioon (JSR 154) kuuluva määrittely, joka tehdään lisäyksenä sovelluksen *war*-hakemiston tiedostoon *web.xml* (WEB-INF -hakemistossa). Ilman tätä palvelin ei tunnistaisi palvelupyyntöä sellaiseksi, joka pitää käsitellä:

```
  <servlet>
    <servlet-name>ManagerCommandController</servlet-name>
    <servlet-class>ekky.helpdesk.server.ManagerCommandController</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ManagerCommandController</servlet-name>
    <url-pattern>/helpdesk/ekky.helpdesk.manager.Manager/managercommand</url-
pattern>
    <url-pattern>/ekky.helpdesk.manager.Manager/managercommand</url-pattern>
  </servlet-mapping>
```

Edellä esitelty etäkutsun malli käyttää Google Web Toolkitin RPC-mallia (Remote Procedure Protocol) ja sen sijaan olisi valita myös SmartGWT:n tarjoaman RPCManager-luokan etämetodien käyttämiseen. Valinta näiden kahden välillä on tehty tottumuksellisista syistä.

Huomioitavaa on se, että oman, itse luodun luokan instanssin palauttamiseksi verkon yli on luokan implementoitava Serializable-rajapinta, mutta muuta erityistä ei tarvita. Javan omat luokat (String, Integer, ym.) ja tietorakenteet (Linked-

HashMap, Set, ym.) siirtyvät verkon yli "sellaisenaan" – itse asiassa ne konvertoidaan JavaScriptiksi, jotta selain pystyy käsittelemään niitä. Esimerkiksi seuraavassa on vastauksena etäkutsuun saatu tekstimuotoinen, tietyn syntaksin mukainen vastausviesti, jonka SmartGWT osaa käsitellä:

```
//OK[10,2,9,2,8,2,7,2,6,2,5,2,4,2,3,2,4,1,["java.util.HashMap/962170901","java.lang.String/20040
16611","accountname","antony","email","antony@example.fi","commonid","a526456","fullnam
e","Andy Tony"],0,5]
```

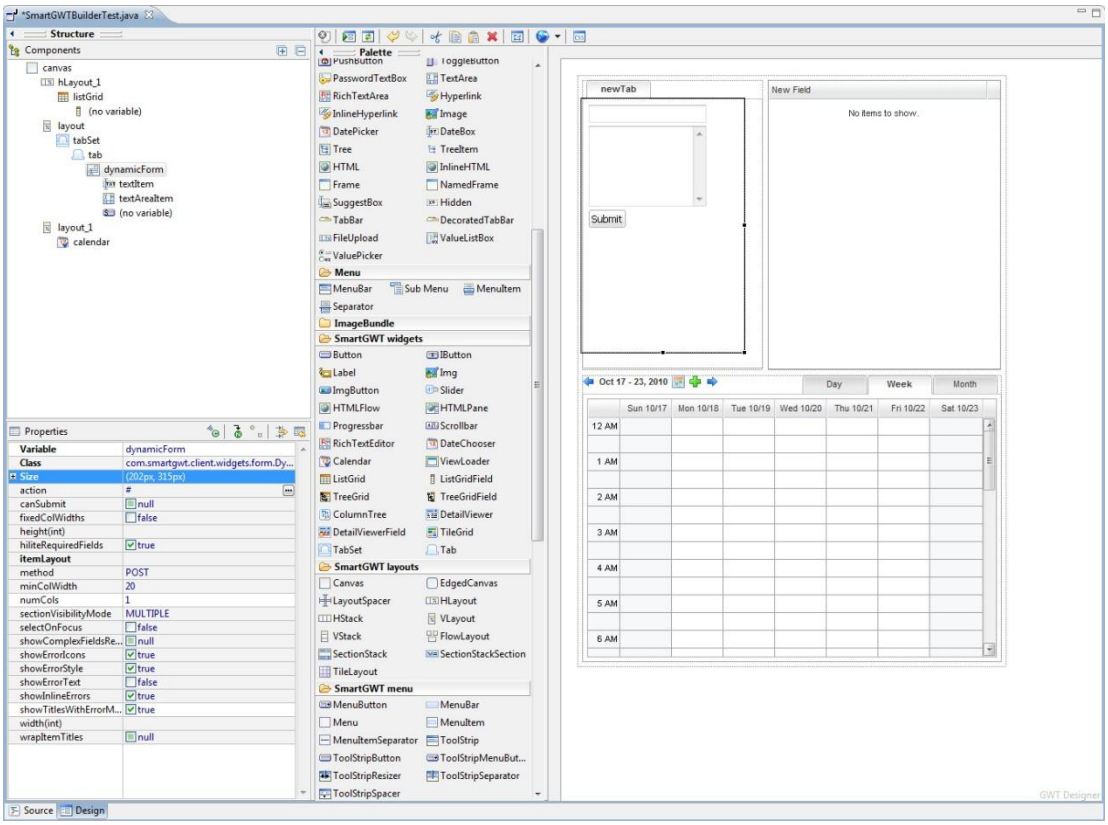
4.4 Sovelluksen mallintamisesta

Sovelluksen mallintamista voi tarkastella useasta eri näkökulmasta kuten sen visuaalisen ulkoasun mallintamisesta ja järjestelmän sisäisen rakenteen mallintamisesta (ohjelmistoarkkitehtuurin eri abstraktiotasot huomioiden). Seuraavassa tuodaan esiin muun muassa käyttöliittymien näyttöjen mallintamista ja sitä kuinka toteutusvaiheen Java-luokat osallistuvat järjestelmän toiminnallisuuden luomiseen.

4.4.1 Näyttöjen mallintamisesta ja tuottamisesta

Projektin alkuvaiheilla, samoihin aikoihin kun tutustuttiin SmartGWT:n enemmän toimintoja ja ominaisuuksia tarjoaviin komponentteihin, tehtiin kokeiluja ehdotuksia käyttöliittymien näytöiksi. Tehtyjen näyttökuvien avulla oli helpompi päästä alkuun erilaisista ratkaisuvaihtoehdoista asiakkaan kanssa keskusteltaessa. Käytännössä noita näyttökuvia ei tehty Instantiationsin kehittämällä GWT Designerillä (Kuva 24), joka sisälsi SmartGWT-laajennuksen (widget kitin), vaan esimerkkinäytöt variaatioineen luotiin kokonaan ohjelmointikoodia kirjoittaen. GWT Designer on osa WindowBuilder Pro -Eclipse-pluginia.

Valinta olla käyttämättä tarjolla ollutta visuaalista editoria johtui siitä, että se tuntui liian hitaalta ja se lisäili ohjelmointikoodiin aaltosulkeita, jotka olivat avuksi editorille itselleen, mutta olivat koodaajan kannalta häiritseviä. Elementtien asemointi oli editorissa helpompaa, mutta muilta osin SmartGWT:n komponentit vaikuttivat olevan siinä määrin helppoja käsitellä ohjelmointikoodissa, ettei merkittävää tarvetta editorin käyttöön edes esiintynyt.

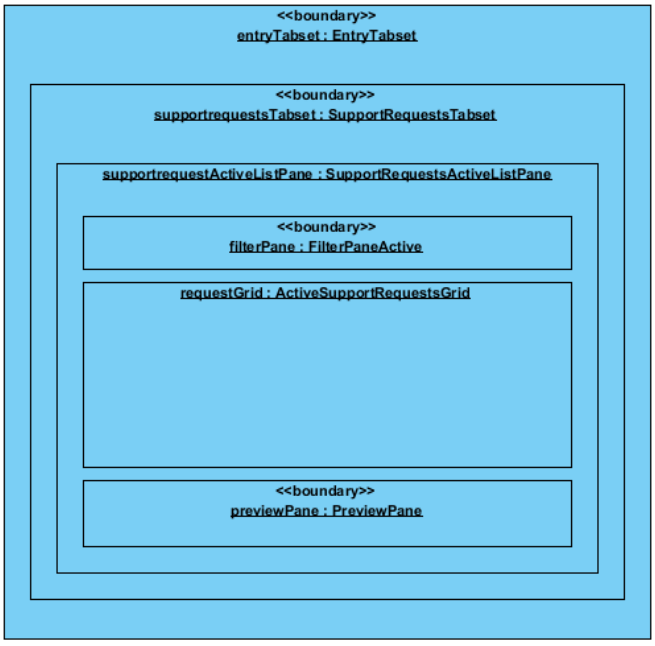


Kuva 24. WindowBuilder Prohon sisältyvä GWT Designer mahdollistaa myös SmartGWT-pohjaisen sovelluksen visuaalisen luomisen

Kyseiseen editoriin liittyen toteutui ennakoimaton riski: Google osti itselleen Instantiationsin tuotteet, mikä tarkoitti käytännössä sitä, että tämä työkalu olisi muutenkin ollut saavuttamattomissa. Elokuun alkupäivinä oli Instantiations-yrityksen verkkosivuille asetettu ilmoitus, jossa kerrottiin, että SmartGWT GUI Builder ei enää olisi saatavilla, koska Google on ostanut itselleen useimmat kyseisen yrityksen tuotteista. Kesti syyskuun 16. päivään asti ennen kuin Google lopulta julkaisi (Google Web Toolkit Blog 2010) Google-brandatun WindowBuilder Pron, jonka osana GWT Designer on. Uusi GWT Designer sisältää "widget toolkitit" myös SmartGWT:tä varten. Ohjelmassa itsessään ei kuitenkaan ollut tapahtunut sellaista muutosta, joka olisi tehnyt siitä nopeamman käyttää. Jokainen tehdyn toiminnon peruutus (undo) tai jonkin komponentin ominaisuuden muutos aiheutti edelleen vähintään sekunnin mittaisen hidasteen. Tulevissa ohjelman versioissa tämä ongelma saattaa poistua, sekä sen ominaisuudet voivat täydentyä.

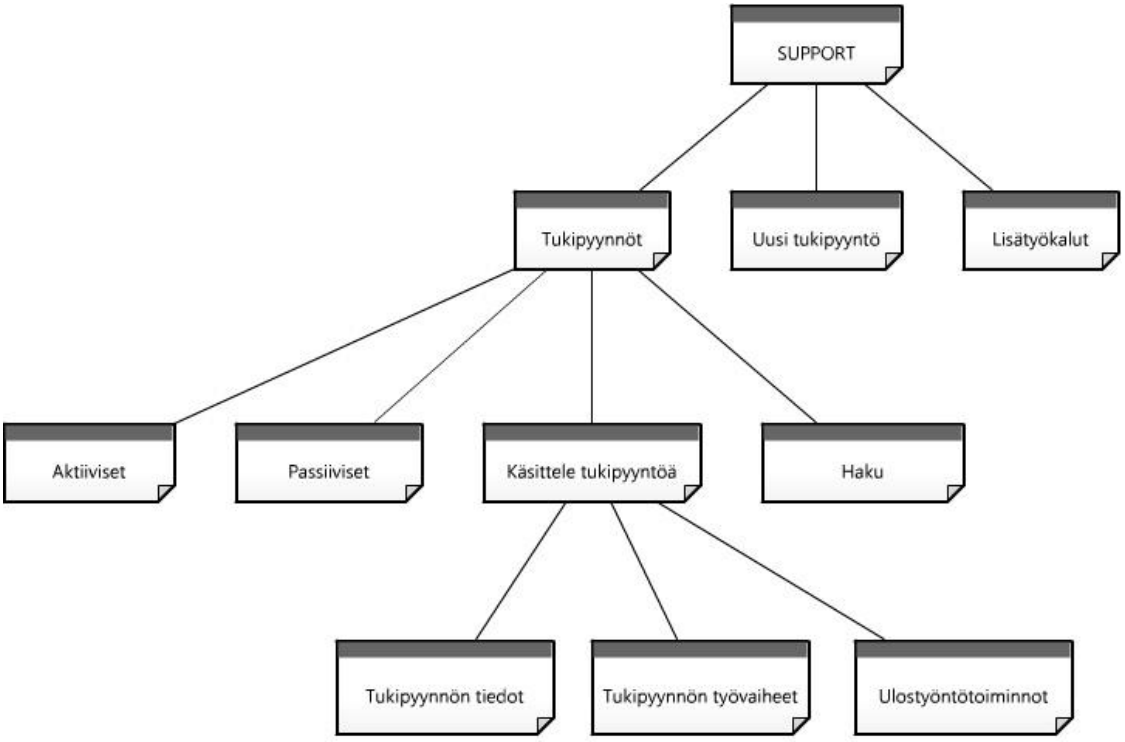
Käyttöliittymien näyttöjen mallintamisessa onkin käytetty (Toiminnallinen määrittely -dokumentissa) UML:n objektikaavioita. Kuvaustapa ei ole minkään tietyn standardin mukainen, mutta UML:n tarjoamia elementtejä hyödyntäen on voitu kuvailla käyttöliittymien sisältämien komponenttien ja elementtien asemointia. Kukin nelikulmainen objekti vastaa ohjelmointikoodissa käytettyä luokkaa (sen instanssia nimeä).

Luodut objektikaaviot (esimerkkinä kuva 25) ovat pelkistyksiä, mikä tarkoittaa tässä sitä, että jokaista yksittäistä muita komponentteja sisällään pitävää paneelia ei oteta mukaan käyttöliittymiä kuvaaviin kaavioihin. Osa toiminnallisuuden kannalta tärkeistä SmartGWT:n tarjoamista pienkomponenteista, kuten painonapit (IButton), on joissain tapauksissa otettu mukaan. Pelkistämisen hyvä puoli on siinäkin, että objektikaavioita ei tarvitse päivittää jokaisen pienen muutoksen jälkeen (niiden tuottamiseen ei ole käytetty minkäänlaista automatiikkaa). Seuraavassa esimerkkinä Aktiiviset-välilehden olennaiset luokat ja niiden instanssit.



Kuva 25. Aktiiviset-välilehden kuvailemiseen käytetty objektikaavio

Objektikaaviot on nimetty välilehtien korvakkeissa esiintyviä nimiä vastaavasti. Välilehdistä muodostuu hierarkia, jonka kuvaamiseen on luotu karttakuva (Kuva 26) käyttäen SaaS-palvelua Lovely Charts. Karttakuvaa hyödynnetään myös käyttöliittymien ja toiminnallisuuden testaamisessa.



Kuva 26. Lovely Chartilla tehty karttakuva, josta on luettavissa Käsittelijän välilehtien korvakkeissa esiintyvät sanat

Merkittävänä vaikutteena sille, minkälaiseksi käyttöliittymät ovat muodostuneet, on ollut se, että SmartGWT-komponenteista, niiden tarjoamista ominaisuuksista ja järjestelmän itsensä tietosisällöstä saa lukumääräisesti, visuaalisesti ja toiminnallisesti niin monia erilaisia kombinaatioita, että niihin on osittain päädytty sellaisten kokeilujen tulosten kautta, joiden käytettävyydestä ei aluksi ollut aivan täyttä varmuutta. Ruudukko-komponentti (grid) on eräs tällaisista, mistä esimerkki kuvassa 27.

52	14. heinä 2010	Avoin		Lappeenranta, Pohjolankatu 12		ei	Käyttäjän tukipyyntö (lomake)
53	14. heinä 2010	Käsittelyssä	Ohjelmat, käyttöjärjestelmät ja ohjelmistot	Lappeenranta, Pohjolankatu 12	jude	ei	Käyttäjän tukipyyntö (lomake)
54	14. heinä 2010	Avoin		Lappeenranta, Pohjolankatu 12		ei	Käyttäjän tukipyyntö (lomake)
55	13. heinä 2010	Avoin		Lappeenranta, Pohjolankatu 12		ei	Käyttäjän tukipyyntö (lomake)
56	13. heinä 2010	Avoin		Imatra, Koulukatu 5		ei	Käyttäjän tukipyyntö (lomake)

Kuva 27. Osa käytössä olevaa SmartGWT:n ruudukkokomponenttia

Kun ensimmäisiä prototyyppejä tukipyyntöjä listaavasta ruudukosta tehtiin, ei ensimmäisenä tullut mieleen, että grid-komponentissa olisi käytettävissä niin sanottu rollover-palkki, johon voi sijoittaa myös ohjelmoitavia toimintaikoneita. Sekin selvisi vasta myöhemmin kuinka tiettyä ehtoa vastaavan rivin tietyt sarak-

keet voi muuttaa ei-editoitavissa -tilaan, muiden ruudukon solujen ja kaikkien muiden rivien pysyessä editoitavissa hiiren klikkauksella.

Kokeilujen kautta ja mieltymyksien jalostuessa alkoivat käyttöliittymät saada pysyvämpää ja vähemmän muutoksille altista olemusta. Tämän muutoksille alttiuden vuoksi tuntui turhalta työltä käydä liian aikaisin kirjaamaan toiminnalliseen määrittelyyn ylös käyttötapauksien kuvauksia ja piirtää toimintaa kuvaavia kaavioita – tilanne olisi ollut toinen, jos projekti olisi ollut toteuttamassa useampi kuin yksi henkilö. Lopulta kaikki käyttötapaukset kuitenkin saivat kuvauksensa. Seuraavassa esimerkinomaisesti katkelma käsittelijän käyttötapauksesta "Muokkaa tukipyynnön tietoja".

Nimi
Muokkaa tukipyynnön tietoja

Kuvaus
Käsittelijä muokkaa tukipyynnön tietoja muokkaamalla tietokenttiä ja asettamalla valintalistoista haluamansa valinnat. Lisäksi voidaan asettaa Julkinen-attribuutti, joka määrittää sen, voiko tukipyyntöä nähdä käsittelijän käyttöliittymien ulkopuolelta eli käyttämällä tiettyä sovittua url-osoitteen muotoa, jossa on annettu parametrina tukipyynnön koodi.

Alkutila ja alkuehdot
Täytyy olla sisäänkirjautunut käsittelijän käyttöliittymään. Täytyy olla joka Aktiiviset, Passiiviset tai Haku-välilehdellä.

Tyypillinen käyttötapauksen kulku
Tuplaklikataan jotain tukipyyntörivijä ruudukossa tai klikataan kerran rollover-palkista "Valitse muokattavaksi" -ikonia. Tämän jälkeen siirrytään automaattisesti "Käsittele tukipyyntöä" -välilehdelle, jolle on haettu palvelimelta tukipyynnöt tiedot. Tietokenttiin muokataan halutusti, sekä muokataan mahdollisesti myös valintalistojen valintoja, asetetaan mahdollisesti pääkäsittelijä, ehkä määräpäiväkin ja ehkä asetetaan myös julkiseksi. Vapaateksti-kenttään voi kirjoittaa mitä tarpeelliseksi ja täydentäväksi koetaan. Lopuksi painetaan Tallenna-nappia, jolloin tukipyyntö tallentuu tietokantaan.

Virhe 1
Tietokenttien validointi ei läpäisty, joka johtuu aina siitä, että kentissä on epäkelpoa tietoa.

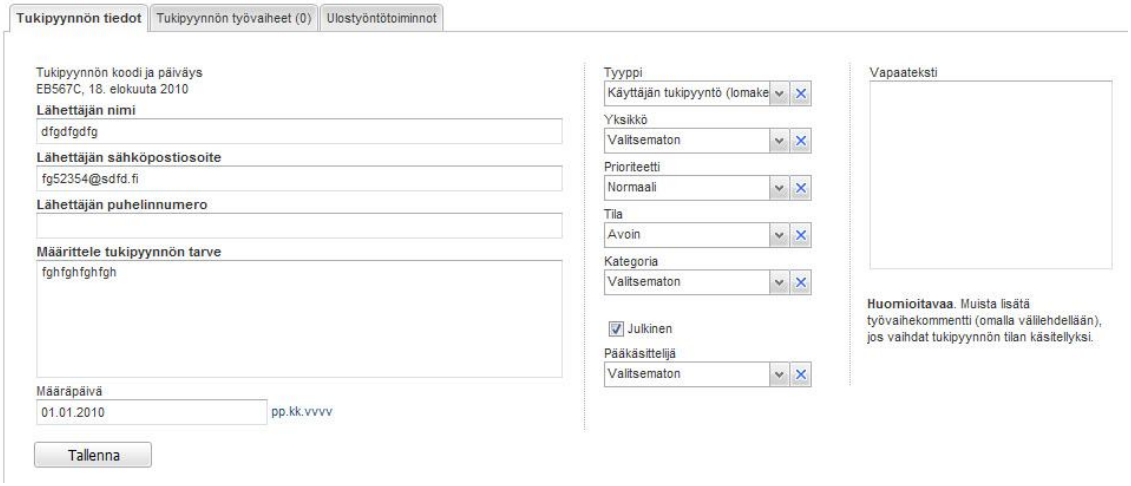
Virhe 2
Palvelimeen ei saada yhteyttä tukipyynnön tallentamiseksi.

Erityisvaatimukset
Täytyy hyväksyä, että tukipyynnön koodia ja alkuperäistä tukipyynnön luontipäivää ei voi muuttaa.

Lopputila ja jälkiehdot
Tukipyynnön uudet tiedot ovat tallentuneet tietokantaan. Lisäksi, jos on vaihdettu pääkäsittelijää, kirjautuu lokiruudukkoon ("Tukipyynnön työvaiheet"-välilehdellä) tieto siitä, että pääkäsittelijä on muuttunut.

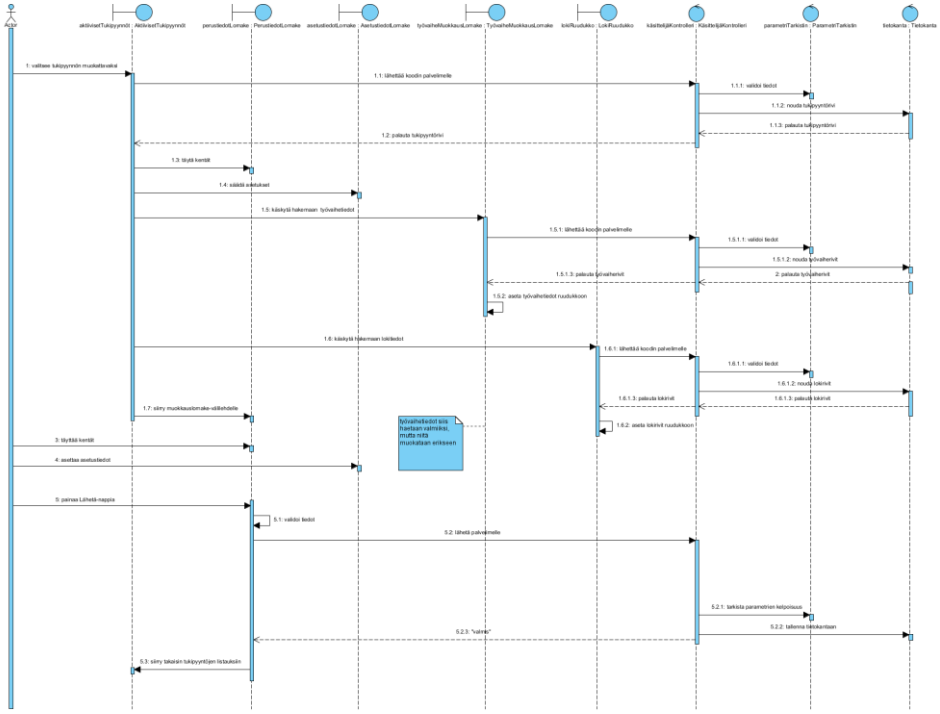
Käyttöliittymän osa, johon käyttötapaus viittaa (Kuva 28), on ollut alttiina muutamille muutoksilla, joista useimmat ovat saaneet alkunsa tarpeesta lisätä jo-

honkin tietokannan tauluun uusi attribuutti. Käyttötapaus sisältää tekstuaalisen kuvauksen lisäksi sen verran selventäviä kuvia kuin on tarpeen.



Kuva 28. Tukipyynnön muokkaamiseen tarkoitettu lomake käsittelijän käyttöliittymässä

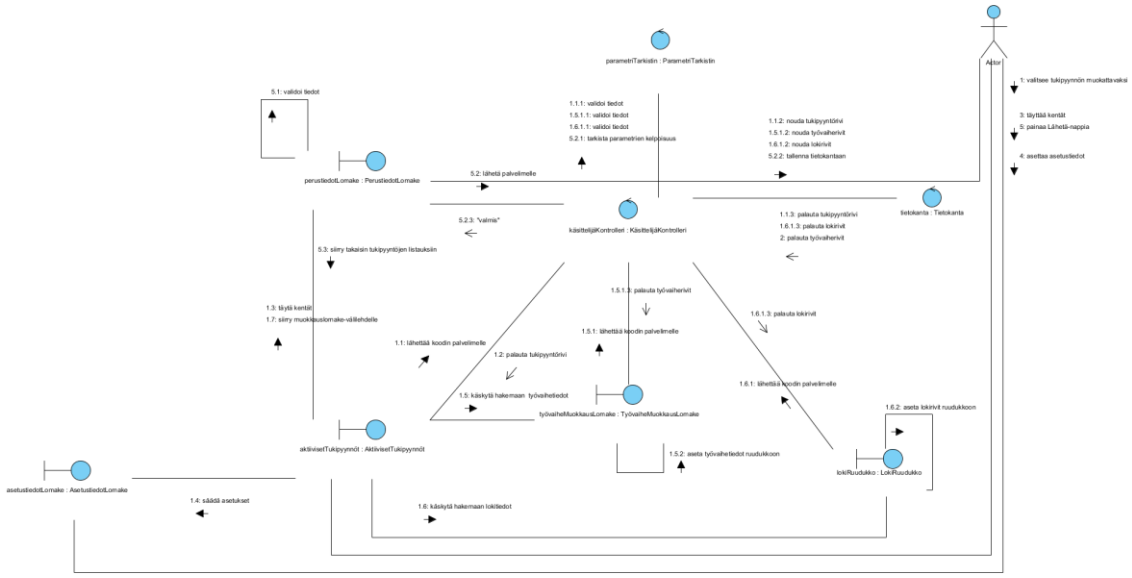
Käyttötapauskuvausten yhteydessä on myös kuvausta tänydentävinä kaavioina sekvenssi- (Kuva 29) ja kommunikaatiokaavio (Kuva 30).



Kuva 29. Käyttötapauksen sekvenssikaavio

Useimmissa tapauksissa sekvenssikaavio on hyödyllisempi kuin kommunikaatiokaavio, mutta joissakin tilanteissa kommunikaatiota on helpompi lukea.

Kommunikaaviot on tuotettu semiautomatisoidusti käyttäen mallinnusohjelma Visual Paradigm for UML:n synkronisointitoimintoa, minkä jälkeen kaavion elementtejä on sijoiteltu manuaalisesti. Kaikki UML kaaviot on tuotettu käyttäen kyseistä ohjelmaa. Kommunikaatikaaviossa ei voida esittää kaikkea sitä tietoa, mikä on sekvenssikaaviossa ja se kuvaakin vain, kuten kaavion nimikin antaa ymmärtää, olioiden välistä kommunikointia.



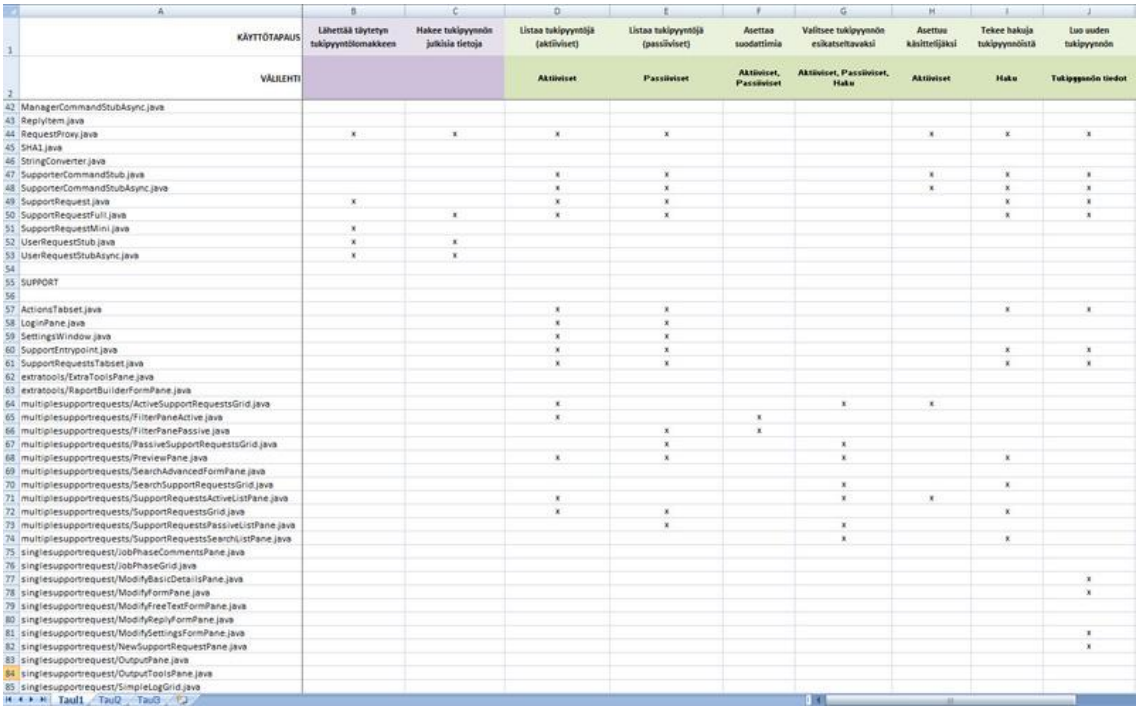
Kuva 30. Käyttötapauksen kommunikaatiokaavio

4.4.2 Välilehdillä käytössä olevien toimintojen tarvitsemat Java-luokat

Toteutuksen aikana koettiin käytännölliseksi luoda Excel-taulukkolaskentaohjelmalla matriisi (Kuva 31), josta ilmenee, mitä Java-luokkia mikäkin toiminto/välilehti käyttää. Jos järjestelmää kehittää jossain vaiheessa joku muu, voi tästä matriisista olla paljon hyötyä, ainakin aluksi, sillä sen avulla etsittävien kohteiden hapuilu vähenee merkittävästi. Kyseinen matriisi on tehty siinä vaiheessa, kun suurin osa toiminnallisista vaatimuksista oli jo toteutettu.

Puutteena tässä taktiikassa on se, että matriisi ei päivity itsekseen, vaan se on tehtävä manuaalisesti, mutta sitä ei ole helppo korvata ainakaan automaattitoimintaisia, lähdekoodin staattisia analysaattoreita käyttämällä (perustelu alaluvussa 4.4.3, "Olioiden instanssien paikallistettavuudesta"). Ajonaikaiseen toimintaan kytkeytyvällä dynaamisella analysaattorilla sen sijaan olisi mahdollista jäljittää esimerkiksi sitä, minkä luokkien instansseja mikäkin välilehti käyttää,

suorittamalla kyseisellä välilehdellä esiintyviä toimintoja ja antaen analysaattorin monitoroida ja lokittaa toimintaa.



	A	B	C	D	E	F	G	H	I	J
1	KÄYTTÖTAPAUS	Lähettää täytetyn tukipyyntölomakkeen	Hakee tukipyynnön julkisia tietoja	Listaa tukipyyntöjä (aktiiviset)	Listaa tukipyyntöjä (passiiviset)	Asettaa suodattimia	Valitsee tukipyynnön esikatseltavaksi	Asettuu käsittelijäksi	Tekee hakuja tukipyynnöistä	Luo uuden tukipyynnön
2	VÄLILEHTI			Aktiiviset	Passiiviset	Aktiiviset, Passiiviset	Aktiiviset, Passiiviset, Haku	Aktiiviset	Haku	Tukipyynnön tiedot
42	ManagerCommandStubAsync.java									
43	ReplyItem.java									
44	RequestProxy.java	x	x	x	x			x	x	x
45	SHA1.java									
46	StringConverter.java									
47	SupporterCommandStub.java			x	x			x	x	x
48	SupporterCommandStubAsync.java			x	x			x	x	x
49	SupportRequest.java	x		x	x				x	x
50	SupportRequestFull.java		x	x	x				x	x
51	SupportRequestMini.java	x								
52	UserRequestStub.java	x	x							
53	UserRequestStubAsync.java	x	x							
54										
55	SUPPORT									
56										
57	ActionsTabset.java			x	x				x	x
58	LoginPane.java			x	x					
59	SettingsWindow.java			x	x					
60	SupportEntrypoint.java			x	x				x	x
61	SupportRequestsTabset.java			x	x				x	x
62	extratools/ExtraToolsPane.java									
63	extratools/RaportBuilderFormPane.java									
64	multiplesupportrequests/ActiveSupportRequestsGrid.java			x			x	x		
65	multiplesupportrequests/FilterPaneActive.java			x		x				
66	multiplesupportrequests/FilterPanePassive.java				x	x				
67	multiplesupportrequests/PassiveSupportRequestsGrid.java				x		x			
68	multiplesupportrequests/PreviewPane.java			x	x		x		x	
69	multiplesupportrequests/SearchAdvancedFormPane.java									
70	multiplesupportrequests/SearchSupportRequestsGrid.java						x		x	
71	multiplesupportrequests/SupportRequestsActiveListPane.java			x			x	x		
72	multiplesupportrequests/SupportRequestsGrid.java			x	x				x	
73	multiplesupportrequests/SupportRequestsPassiveListPane.java				x		x			
74	multiplesupportrequests/SupportRequestsSearchListPane.java						x		x	
75	singlesupportrequest/JobPhaseCommentsPane.java									
76	singlesupportrequest/JobPhaseGrid.java									
77	singlesupportrequest/ModifyBasicDetailsPane.java									x
78	singlesupportrequest/ModifyFormPane.java									x
79	singlesupportrequest/ModifyFreeTextFormPane.java									
80	singlesupportrequest/ModifyReplyFormPane.java									
81	singlesupportrequest/ModifySettingsFormPane.java									x
82	singlesupportrequest/NewSupportRequestPane.java									x
83	singlesupportrequest/OutputPane.java									
84	singlesupportrequest/OutputToolsPane.java									
85	singlesupportrequest/SimpleLogGrid.java									

Kuva 31. Matriisi välilehtien ja luokkien välisistä yhteyksistä (ei kokonaisuudessaan)

Palvelinpuolella sijaitsee kontrolliluokkia, jotka implementoivat palvelimenpuoleisen, moduulikohtaisen tynkärajapinnan, sekä muita luokkia, joiden voidaan ajatella tarjoavan palveluita näille kontrolliluokille. Näitä luokkia ovat *ManagerCommandController, SupporterCommandController, UserCommandController, LoginCommandController, ParameterChecker, RandomGenerator, SessionChecker, SimpleLogger ja Mailer*.

Teknilliseen määrittelydokumenttiin sisältyi kaikkien näiden luokkien kuvaukset ja olennaiset tarkentavat tiedot. Seuraavassa on esimerkkinä ManagerCommandController-luokan kuvaus:

Luokan nimi
ManagerCommandController

Tyyppi
Kontrolleriluokka

Yleiskuvaus
Sisältää nipun metodeita, joita sovelluksen asiakaspuoli kutsuu tynkärajapintojen kautta.

Asiakkaat
Hallinnoijan moduulissa sijaitsevat luokat

Luokan metodeilla tyypillistä

Jokainen ManagerCommandController-luokan metodi alkaa parametrien tarkistuksella, johon käytetään luokkaa ParameterChecker. Jos parametreissä ilmenee virheitä tai epäkelpoisuutta, päätetään metodin suoritus siihen ja palauttamalla kutsujalle IllegalArgumentException-poikkeuksella, lyhyen selitetekstin kera:

```
if (!ParameterChecker.check_manager_fetchAccountsDetails(accountname)) {
  throw new IllegalArgumentException("invalidparameters");
}
```

Jos parametritarkistus meni läpi ok, jatketaan varmistumalla siitä, että kutsu on peräisin autentikoidulta käyttäjältä:

```
// is logged in?
SessionChecker sessionChecker = new SessionChecker();
HttpServletRequest request = this.getThreadLocalRequest();
if (!sessionChecker.isLoggedIn(request).get("accounttype").equals("manager")) {
  return null;
}
```

Tämän jälkeen suoritetaan varsinainen metodin runko-osa, joka tyypillisesti alkaa tietokantayhteyden alustamisella (käytetään DBConnection-luokkaa) ja jatkuu parametrien jatkokäsittelyn kautta yhden tai useamman tietokantatransaktion suorittamiseen. Jos tietokannan käsittelyssä ei ilmennyt virheitä ja toiminta kulki muutenkin kuten oli tarkoitettu, palautetaan metodin lopuksi etämetodin kutsujalle jokin vastausviesti, joka voi olla mitä tahansa serialisoitavissa olevaa tyyppiä, kuten Integer, String, HashMap tai jokin oma luokka.

4.4.3 Olioiden instanssien paikallistettavuudesta

Monien toimintojen toteuttamiseksi on koodissa jouduttu kuljettamaan luokkien instansseille ennalta tietoa siitä, mistä on saatavilla jonkin tietyn toisen luokan instanssi. Tällainen tarve ilmenee esimerkiksi sellaisessa tapauksessa, jossa yhdellä välilehdellä tehty toiminta aiheuttaa toiminnon, joka haluaa esivalmistella jonkin toisen välilehden toimintoja. Tuolloin voitaisiin tarvita tietoa siitä, mikä on jonkin olion (esimerkiksi välilehti, välilehdellä sijaitseva paneeli tai välilehdellä sijaitseva komponentti) instanssi tai mistä saadaan viite siihen.

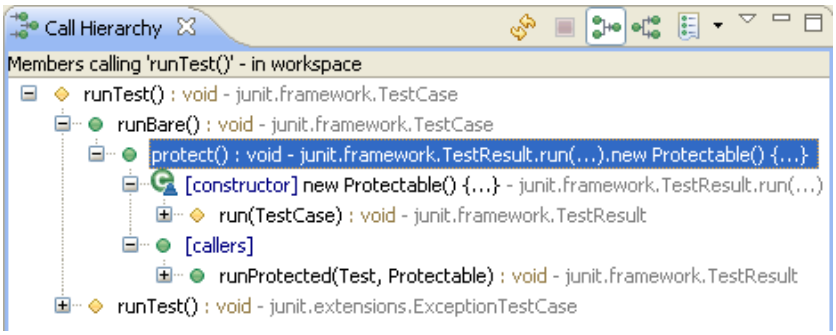
Toteutuksessa on lähdetty siitä ideasta, että jos tietoa kulloinkin käsittelevällä oliolla ei olisi tietoa siitä, missä jokin tietty toinen olio sijaitsee, koodia muutetaan sen verran, että tuo tieto voidaan sille saakka "valuttaa". Valuttaminen tarkoittaa tässä sitä, että tieto välilehtiä vastaavien instanssien muodostamissa hierarkioissa valutetaan ylemmiltä tasoilta alemmas – niin pitkälle kuin on tarve ja vain yhtä polkua pitkin. Tämän jälkeen koodissa voidaan ketjuttaa viittaukset esimerkiksi tähän tapaan:

```
getSupportrequestModifyPane().getReplyFormPane().fillReply(
  supportrequest.getReply(), supportrequest.getReplymoment()
);
```

Tämä aiheuttaa tietynlaista "spagettikoodimaisuutta", jollaista proseduaalista ja oliopohjaista ohjelmointiparadigmaa kehittäneet halusivat vältettävän. Käytännössä oliot joutuvat tarjoamaan niiden olemukseen kuulumattomia palveluita toimiessaan niille itselleen tarpeettoman tiedon säilyttäjinä. Jos järjestelmä olisi laajempi, tämä muodostuisi hankalaksi hahmottaa, mutta nykykoossaan näitä pystyy vielä seuraamaan. Järjestelmän jatkokehityksessä olisikin järkevää harkita jonkinlaisen välilehtihakemiston luomista.

Johtuen SmartGWT:n komponenttien moniroolisesta luonteesta (ovat usein yhtä aikaa kontrolli-, entiteetti- ja rajapintaluokkia) sekä näistä erityisistä teknisistä ratkaisuista, ei sovelluksen asiakaspuolelta ole helposti rajattavissa moduulikohtaisia paketteja tarkempia osia itsenäisiksi toiminnallisiksi yksiköiksi, sillä liian monet luokat ovat vähintään muutamin kuvaillunlaisin "sidoksin" kiinni muissa luokissa. Tämä pätee myös SmartGWT:n graafisista komponenteista periytettyihin luokkiin. Palvelinpuolella samaa ongelmaa ei ole (katso alakohta 6.8, "Palvelinpuolen luokat").

Niissä yhteyksissä, joissa seurataan ohjelmointikooditasolla sitä, mihin jokin näistä ketjutetuista metodeista pyrkii viittamaan, auttaa Eclipse-kehitysympäristön apuväline Call Hierarchy (Kuva 32). Sillä voi seurata, mistä jotain tiettyä metodia kutsutaan ja mitä metodeita valittu metodi itse kutsuu.



Kuva 32. Eclipse-kehitysympäristön Call Hierarchy -apuväline

Valitettavasti tämä ei toimi niin pitkälle, että voitaisiin seurata kutsuja asiakaspuolen luokista palvelinpuolen luokkiin ja takaisin, sillä Eclipse ei kykene tulkitsemaan sitä, miten sen pitäisi menetellä tynkärajapintojen välillä (etäkutsujen yhteydessä) olevan "näkymättömän yhteyden" suhteen. Tätä ongelmaa käsitellään myös alaluvussa 4.4.4 ("Sovelluksen automatisoidusta takaisinmallintamisesta kaavioiksi").

Eclipsen ominaisuuksiin yleensä on hyödyllistä tutustua, sillä ne tekevät ohjelmointityöstä sitä mielekkäämpää, mitä paremmin nuo ominaisuudet saa sovitettua mukaan työskentelymetodeihinsa. Esimerkkinä mainittakoon metodin määrittelyyn siirtyminen klikkaamalla Ctrl-näppäin pohjassa metodin nimeä (koodissa), sekä Open Resource -toiminto, jonka saa pikaisesti esille painamalla näppäinyhdistelmää Ctrl + Shift + R. Kirjoittamalla avautuvaan dialogiin muutaman kirjaimen etsitystä resurssista (esimerkiksi luokan nimi), päivittyy samassa yhteydessä oleva lista hakuehtoja vastaavista resursseista reaaliajassa. Lisää Eclipse-vinkkejä löytyy Eclipsen omasta dokumentaatiosta, joka on saatavilla osoitteesta: http://help.eclipse.org.

Tukipyyntöjenkäsittelyjärjestelmän toimintaan tutustuttaessa on hyödyllistä käydä läpi toiminnalliseen määrittelyyn sisältyviä käyttötapauksia, jotka sisältävät käyttötapauksiin liittyviä sekvenssikaavioita – siitäkin huolimatta, että niissä on käytetty vain järjestelmän määrittelyvaiheen (analyysivaiheen) pseudoluokkia.

4.4.4 Sovelluksen automatisoidusta takaisinmallintamisesta kaavioiksi

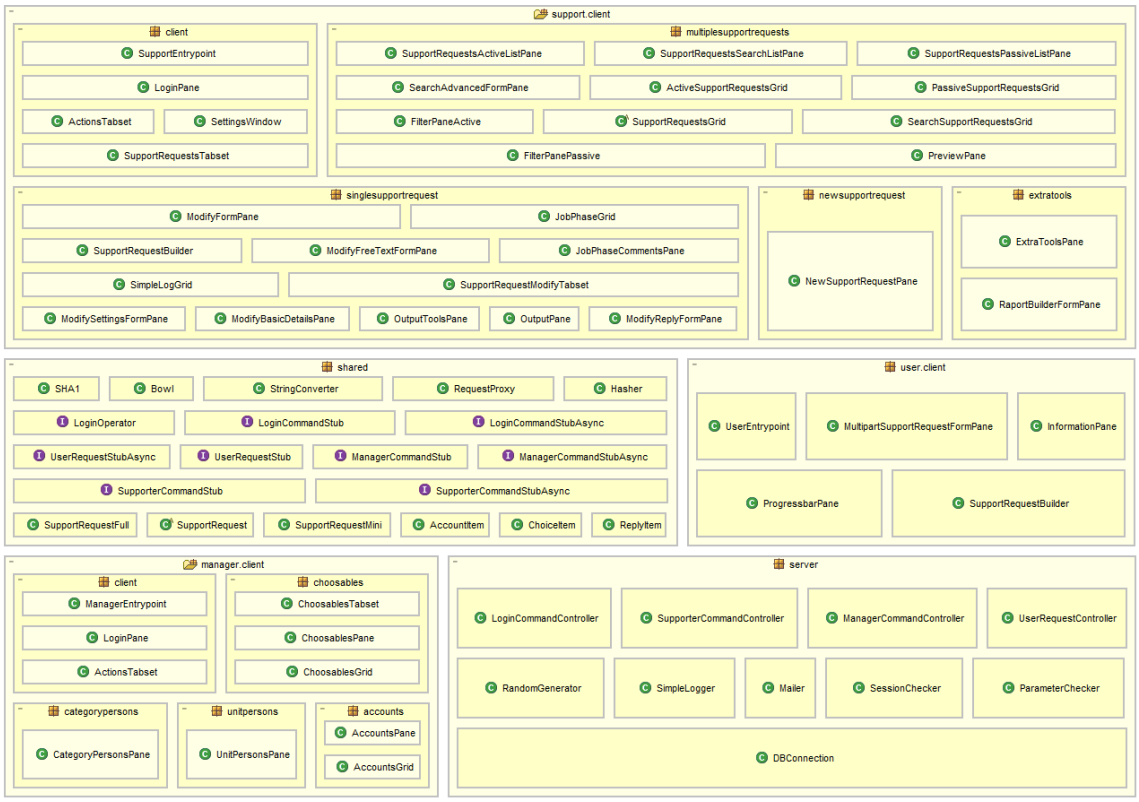
Yksikään kokeilluista takaisinmallinnus-toiminnon tarjoavista ohjelmista (muutamia poimintoja laajasta tarjonnasta) ei kyennyt mallintamaan GWT:n RPC-

kutsuja ja täten sovelluksen toiminnan mallintamista ei voi automatisoida, vaan suuri osa mallintamisesta jouduttaisiin tekemään manuaalisesti.

Mallinnusongelma syntyy kahdesta seikasta:

- Käyttäjän tekemä toiminta voi aiheuttaa yhden tai useamman asynkronisen etämetodikutsun luomisen, joista kukin voi takaisinkutsumetodin päätteeksi käynnistää uuden etämetodikutsun. Kukin etäkutsu alkaa asiakaspuolella, kulkien tynkärajapintojen kautta palvelinpuolelle. Koska tätä tynkärajapintojen kautta kulkemista ei ole voinut mallintaa automaattisesti, olisi muutoksia tehtäessä muistettava aina erikseen päivittää asiaan liittyvät kaaviot. Tämä voisi muodostua työlääksi muistettavuuden kannalta ja olla täten virhealtista.
- Ohjelmien takaisinmallinnustoiminnot eivät tyypillisesti "ymmärtäneet" esimerkiksi sitä, että välilehtisäilöön (TabSet) oli sijoitettu välilehtiä (Tab), joilla on kullakin sisältönä Canvas-tyyppinen luokan instanssi, kuten HLayout-paneeli. Jotkin niistä saattoivat tehdä "valistuneen arvauksen", jota seurasi, että yksi ohjelma saattoi käyttää assosiaatioviivaa yhteyden kuvaamiseen, toinen yleistystä, kolmas jotain muuta ja neljäs ei nähnyt yhteyttä ollenkaan.

Takaisinmallinnusohjelmia on lukuisia erilaisia ja olisi erittäin hyödyllistä ja kehitystyötä ratkaisevasti helpottavaa, jos niiden avulla voisi mallintaa tämänkin sovelluksen kokonaisuudessaan. Sovelluksen versio 1.0 koostuu 76 Java-luokasta, joiden pohjalta on muodostettu kuvassa 33 oleva malli, joka sisältää ne kaikki. Kuva on tuotettu käyttäen Structure 101 for Java -ohjelmaa.



Kuva 33. Kaikki sovellun käyttämät luokat

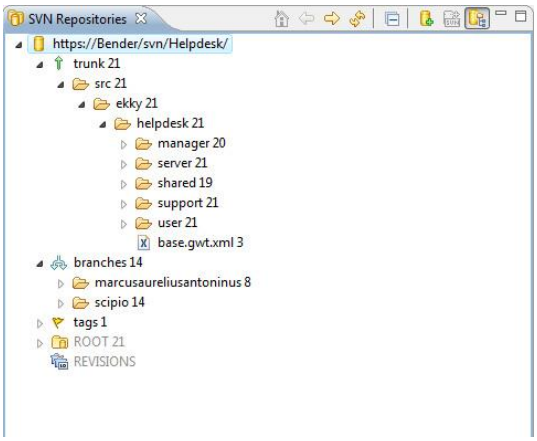
4.5 Pieniä yksityiskohtia

Seuraaviin alalukuihin on valikoitu muutama järjestelmän toteutukseen liittyvä yksityiskohta.

4.5.1 Versionhallinta

Versionhallinta otettiin kehitystyöhön mukaan vasta opinnäytetyön loppuvaiheilla eli siinä vaiheessa, kun tukipyyntöjenkäsittelyjärjestelmän versio 1.0:n toiminnallisuuksista lähinnä raportointi oli toteuttamatta. Lähdekoodista ja kaikesta muusta projektiin liittyvästä materiaalista otettiin varmuuskopiot jokaisen muutamia tunteja kerrallaan kestäneen työjakson päätteeksi, mutta sellaista ei voi kutsua varsinaiseksi versionhallinnaksi. Käyttöönotettu versionhallintajärjestelmä pohjautuu Apache Subversioniin, jonka käyttämisen helpottamiseksi ja osatoimintojen automatisoimiseksi asennettiin Eclipseen Subversive-plugin. Apache Subversioniin pohjautuvia levityspaketteja on useita erilaisia. Näistä valittiin käyttöön VisualSVN Server Standard Edition.

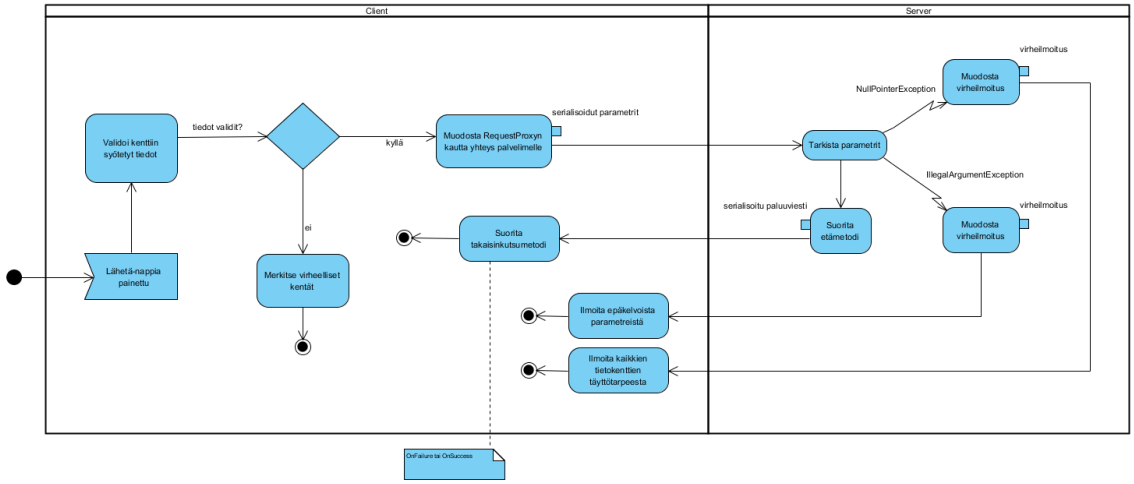
Versionhallinnan käyttöönoton tarkoitus oli erityisesti siinä, että tällä tavoin voidaan pitää yllä kahta tai useampaa ohjelmistokehityslinjaa, joiden välillä voi suorittaa myös toiminnallisuuden yhdistämisiä tai eriyttämistä. Konventiona on, että ohjelmiston päälinjan nimi on 'trunk', ja erilliset tuotelinjat ovat hakemiston 'branch' alla. 'Tags' sisältää tilannetallenteita jostain kehityksen vaiheesta, joille voidaan antaa jokin versionumeroa kuvaavampi nimi. Kuvassa 34 on esillä Eclipsen versionhallintanäkymästä se osa, jossa nämä eri kansiot ovat käsiteltävissä. Uusien versioiden luomiseen vaaditaan aina erillinen käskytys Commit-toiminnolla.



Kuva 34. Tukipyyntöjenkäsittelyjärjestelmän versionhallintaa Eclipse-kehitysympäristöön asennetulla Subversivellä

4.5.2 Virhe- ja poikkeusmenettelyt

Virheiden käsittelyä ja poikkeusmenettelyä on tukipyyntöjenkäsittelyjärjestelmän tapauksessa käytännössä kolmenlaista. Yksi liittyy lomakkeilla ja ruudukoissa annettujen tietojen validointiin, toinen palvelinpuolella tapahtuvaan yksityiskohtaiseen syötteiden tarkistukseen (parametritarkistus) ja kolmas vakavampiin virhetilanteisiin kuten yhteyden katkeamiseen tietokantaan tai tiedon tallennuksen epäonnistumiseen. Kuvan 35 aktiviteettikaaviolla on pyritty osoittamaan missä yhteydessä mitäkin tarkistusta käytetään.



Kuva 35. Aktiviteettikaavio tietokenttien validoinnista ja tietojen tarkistuksesta

Lomakevalidointi tapahtuu liittämällä kuhunkin haluttuun lomakkeen kenttään oma validaattorinsa, joka voi säännöllisiin lausekkeisiin verraten validoida annetun syötteen kelpoisuuden.

```
TextAreaItem descriptionItem = new TextAreaItem("description", "M&auml;&auml;rittele tukipyynn&ouml;n tarve");
RegExpValidator descriptionValidator = new RegExpValidator();
descriptionValidator.setErrorMessage("Ep&auml;kelpo kuvausteksti");
descriptionValidator.setExpression("^[(a-zA-z\u00c0-\u00ff\u0100-\u017f\u0180-\u02000-9\\-\\*\\+=\\/\\.,\\(\\)\\[\\] _%\\|\\^&;:<>\\@\\?\\!\\'\\\"|\n)]{1,3000}$");
descriptionItem.setValidators(descriptionValidator);
```

Vääränlaisesta syötteestä seuraa tietojen lähettämisyrityksen jälkeen se, että vääränlaista syötetietoa sisältävän kentän viereen ilmestyy ikoni ilmaisuksi syötetiedon korjauksen tarpeesta. Tämä validointi tapahtuu kokonaisuudessaan asiakaspuolella eli siinä yhteydessä palvelimelle ei lähetetä mitään. Tietoturvasyistä kaikki palvelinpuolen rajapintametodien kautta vastaanottamat syötteet tarkistetaan yksityiskohtaisesti. Tästä ei ole poikkeuksia. Tämä pätee lomakkeilla annettuihin tietoihin, sisällön hakemiseen ruudukoihin, ruudukoiden muuttuneiden sisällön tallentamiseen jne.

```
public static boolean check_user_fetchSupportRequestDetails(String code) {
  Pattern pCode = Pattern.compile("^[a-z0-9]{3,11}$", Pattern.CASE_INSENSITIVE);
  Matcher mCode = pCode.matcher(code);
  return mCode.matches();
}
```

Tarkistuksen osoittaessa syötteen olevan epäkelpoa, suoritetaan hallittu tilanteen jatkokäsittely palauttamalla metodin kutsujalle tietoa syötteen epäkelpoudesta. Vain validi tieto tallentuu tietokantaan ja vain valideilla hakuparametreilla voidaan hakea tietoa tietokannasta.

Tilanteessa, jossa ilmenee vakava virhe kuten yhteyden tietokantaan olemattomuus, tulee kutsuvan metodin asettama asynkroninen callback-metodi saamaan tiedon kutsun epäonnistumista, jolloin se suorittaa implementoimansa AsyncCallback-rajapinnan onFailure-metodin sisältämän toiminnon tai sarjan toimintoja. Joissakin tapauksissa käyttäjän saa nähtäväkseen dialogi-ikkunan, joka kertoo millä tavoin suoritettu toiminto epäonnistui ja mahdollisesti myös miksi. Vakavista virheistä jää myös merkintä Tomcatin logiin, mutta tietokantaan tai muualle levylle ei erikseen kirjoiteta lokimerkintää virheen ilmentymisestä.

4.5.3 Järjestelmään kirjautumisesta ja salasanan salaamisesta

Kirjautuminen järjestelmään tapahtuu SHA1-salauksen ja siemennyksen tukemana, mikä käytännössä tarkoittaa sitä, että salasanaa ei missään vaiheessa lähetetä selväkielisenä verkon yli. Siemennys viittaa suunnittelumalliin *double hashing* (lainaukseen lisätty korostukset)*:*

> *With double hashing, the server generates **a one-time random seed**. The browser then hashes twice: first, it hashes **the password** to yield what's hopefully stored on the database. But instead of sending that, the browser **combines** it with the one-time seed to form a new hash. This new hash is sent to the server. The server then pulls out the stored hash from the database and combines it with the original one-time seed to form a new hash, which **must match the hash that was uploaded**. This works because in both cases, the initial password has been passed through the same two hash functions. In the browser, the user's attempt is passed through a fixed hash function and the result is immediately passed to a new hash function with one-time seed. And in the server, the database already holds the result of hashing the real password using the fixed hash function. (Mahemoff 2006)*

Kirjautuneena pysyminen perustuu sessioihin, joiden kelpoisuus tarkastetaan jokaisen palvelimeen kohdistuvan etäkutsun yhteydessä. Käyttäjätunnuksien salasanat tallennetaan tietokantaan SHA1-muodossa. Sovelluksen asiakaspuolen salauksen toteuttamiseksi Google Web Toolkit -pohjaisessa SmartGWT-sovelluksessa oli tarpeen etsiä sellainen SHA1-algoritmin toteuttava Java-luokka, joka pystyy kiertämään GWT:n asettaman rajoitteen (katso alaluku 4.2, "Valmiin järjestelmän käsittelystä"). Tähän tarpeeseen löytyikin eräs Java-luokka, jonka tekemiseen on moni osallistunut. Sen lähdekoodi sisältää seuraavat tekijätiedot:

```
SHA1.java - An implementation of the SHA-1 Algorithm
This version integrated into Freenet by Ian Clarke (02-02-2000)
(i.clarke@dynamicblue.com) from a previous public domain version by
Chuck McManis (cmcmanis@netcom.com) which was public domain
Tweaked by Mr.Tines<tines@windsong.demon.co.uk> for pegwit, June 1997
- added method 'frob()' which wipes the contents, so as to match
 the bizarre behaviour of Pegwit's double barreled hashing.
```

4.5.4 Loki-toiminnallisuuden toteutuksesta

Toteutettaessa loki-toiminnallisuutta tukipyynnön sisällön muutoksien ja asetuksien vaihtamisten ylöskirjaamiseksi, ajauduttiin muutaman kokeilun kautta huomioon, että ehtolauseiden totuusarvojen vertailulle on myös vaihtoehto. Seuraavaa ei kannata soveltaa liian laajalti ja on tulkinnanvaraista onko se asianmukaisesta tietovirtojen hallitsemistaan huolimatta hyväksyttävissä oikeaoppisten ohjelmointikäytäntöjen mukaiseksi. Vaihtoehtona olisi ollut kirjoittaa useita rivejä if-ehtoja, AND- ja OR-operaattoreita käyttäen, mutta siitä olisi seurannut luettavuuden menetys.

Käytännössä tarpeena oli vertailla tallennettavan tukipyynnön tietoja tietokannassa samalla tukipyynnön koodilla jo olevan tukipyynnön tietoihin, jotta lokiin voitaisiin kirjoittaa täsmällinen muutosta kuvaava merkintä. Jos joidenkin tietojen osalta oli eroavaisuutta, merkittiin siitä tieto lokiin, yhdessä päiväleiman ja tiedon muuttajan nimen kanssa. Tietokannan tauluun SimpleLogger lisättäisiin tuolloin esimerkiksi seuraavanlainen kuvausteksti:

Andy Tony muutti tukipyynnön määräpäivää (aiemmin '18. elo 2010', nyt '19. elo 2010')

Vertailu itsessään, tietoalkio kerrallaan, vaikutti aluksi helpolta tehtävältä, sillä String-tyyppisiä tietoja voitiin verrata helposti String-luokan equals-metodin avulla. Ongelmaksi muodostui pian se, että vertailun kohteina ei saanut olla NULL-tyyppinen arvo, sillä siitä seuraisi heti NullPointerException-poikkeama ja ohjelman suorituksen keskeytys – paitsi jos sen ottaisi mukaan hallittavien poikkeuksien joukkoon. NULL tarkoittaa ei-mitään (sillä ei ole edes pituutta).

Jos NULL-tyyppinen tieto ei aiheuttaisi poikkeamaa, olisi ehtolauseen voinut muodostaa XOR-operaattorin (ks. Taulukko 2.) avulla ("ehdottomasti vain jompikumpi"). Eli tällöin lokiin olisi talletettu merkintä vain siinä tapauksessa, että uudessa ja vanhassa tiedossa olisi eroa; jos molemmat olisivat NULL tai molemmat sisältäisivät samansisältöistä tekstiä, ei lokiin tallettavaa metodia (put) suoritettaisi.

Taulukko 2. XOR-operaattorin käytön hahmottelua

Uusi tieto	**Vanha tieto**	**Totuusarvo**
null	null	false
jotaintietoa	null	true
null	jotaintietoa	true
jotaintietoa	jotaintietoa	false

Lisäksi olisi pitänyt olla vielä yksi ehtolause, joka tarkistaa, että uusi ja vanha *jotaintietoa* olisivat erilaisia. Eroavana tietona voisi olla esimerkiksi kategorian (lyhyt)nimi. Kaavamaisesti ilmaistuna pitäisi muodostaa if-lauseet seuraavan kaavan pohjalta, jos käytettäisiin pelkästään if-ehtoja:

(A XOR B) OR ("uusi jotaintietoa eroaa vanhasta jotaintiedosta")

Seuraavassa esimerkkikoodissa asia tehdään aivan toisin, hyväksikäyttämällä poikkeuksia. Siinä tarkistetaan ensin, että molemmat vertailut kohteet eivät ole NULL, eivätkä ne ole täsmälleen samatkaan. Jos nämä ehdot pätevät, ei lokiin kirjoittamista tarvitse tehdä. Jos ehdot eivät päde, suoritaan ehtolauseen else-osio, jossa tehdään tahallinen poikkeutus, minkä jälkeen suoritus siirtyy poikkeuksen hallitsevaan koodiin. Siellä tapahtuu varsinainen tiedon talletus lokiin.

Lopputuloksena syntyi helppolukuista koodia, jossa ei ole myöskään turhaa toisteisuutta.

```
try {
  if ((currentSupportrequest.getOrganization() == null && supportrequest.getOrganization()
== null)
  || (currentSupportre-
quest.getOrganization().contentEquals(supportrequest.getOrganization()))) {
        // nothing to do
        } else {
          String str = null;
          str.equals("npe"); // will trigger NullPointerException
        }
  } catch (NullPointerException e) {
        put(changerFullname + " muutti organisaatiota (aiemmin '" + currentSupportre-
quest.getOrganizationTitle() + "', nyt '" + getDetail("organization", supportre-
quest.getOrganization(), supportrequest.getCode()) + "')", supportrequest.getCode());
}
```

4.5.5 Tukipyyntöjen koodien deterministisyydestä

Kuten on tarpeen yksilöidä erilaisia kuitteja, esineitä ja muita kohteita, oli myös tukipyynnöt yksilöitävä jollakin tapaa. Tietojärjestelmän itsensä kannalta sillä ei ole merkitystä käytettäisiinkö yksinkertaista numerosta yksi alkavaa laskuria, joka kasvaa aina yhdellä uutta tukipyyntöä kohden, vai onko numerot generoitu jotenkin toisin. Tukipyyntöjenkäsittelyjärjestelmän tapauksessa on valittu tukipyyntöjen koodien generoimiseksi pseudosatunnaisten numeroiden käyttö. Pseudosatunnaisuus tarkoittaa tässä sitä, että samoilla lähtöasetuksilla saadaan aina tietyt samat luvut.

Sellaisen satunnaislukujen joukon tuottaminen, jossa ei olisi yhtään samaa lukua, on matemaattisesti haastavaa, mistä johtuen Java-kielen Random-luokan random-metodilla ei saada pelkästään erilaisia lukua, vaan mukaan tulee aina jonkin verran samoja lukuja. Tämän vuoksi sillä arvotut luvut onkin sijoitettu SortedSet-tietorakenteeseen, joka ei hyväksy itseensä lisätyn kuin yhden kappaleen jonkin tietyn tietotyypin mukaista arvoa (tässä tapauksessa Integer-tyyppisiä numeroita). Kun seuraavan esimerkin mukaisen ohjelman ajaa ja tarkistaa sen jälkeen kuinka monta erilaista lukua SortedSet-tietorakenteeseen

kertyi, niitä ilmenee olevan 48826 kpl eli 50000 arvontakierroksen jälkeen löytyi parin prosentin verran duplikaatteja.

```
Random random = new Random(1000);
SortedSet<Integer> values = new TreeSet<Integer>();

for (int i = 1000000; i < 1050000; i++) {
  int r = random.nextInt(i);
  if (r > 0) {
        values.add(r);
  }
}

System.out.println(values.size());
```

Olennaista on se, että tätä pientä ohjelmaa voisi samoilla asetuksilla ajaa useita kertoja, kertyvien lukujen määrän ollessa aina sama. Lisäksi kaikki arvotut luvutkin olisivat samoja. Jos tukipyyntöjenkäsittelyjärjestelmää varten tarvitaan joskus enemmän kuin 48826 satunnaista lukua, niitä voidaan aina luoda lisää. Käytännössä luvut sijoitetaan omaan tietokannan tauluunsa, josta niitä voidaan tarvittaessa ottaa käyttöön ja "yliviivata" jo käytetyt luvut. Integer-luvuille maksimiarvo on 2147483647, joka muodostuu kaavasta 2^{31} - 1.

Vähemmän pseudomaisten satunnaismerkkijonojen tuottamiseen olisi mahdollista käyttää Javankin tukemaa UUID-standardia, mutta siihen pohjautuvat merkkijonot ovat tarpeeseen nähden liian pitkiä ja tarpeettoman kryptisen näköisiä, kuten tällaisia: 9b7f0184-c4d0-40f1-855b-ea0e26ad4958.

5 TESTAAMINEN

Antonia Bertolino on artikkelissaan "Software Testing Research: Achievements, Challenges, Dreams" (2007) määrittänyt testaustekniikoiden tutkijoiden ja kehittäjien näkemyksiin perustuvan teknologisen tiekartan, jossa hän kuvailee neljää äärimmäistä, mutta saavuttamatonta unelmaa. Nämä unelmat ovat: universaali testiteoria, testipohjainen mallinnus, täysin automaattinen testaus, sekä tehokkuusmaksimoitu testien suunnittelu ja toteutus.

Universaalilla testiteorialla hän tarkoittaa koherenttia ja eksaktia kehystä, jota vasten testaajat voisivat olemassa olevia testaustekniikoita verrata ja sitä kautta ymmärtää niiden suhteelliset vahvuudet ja rajoitukset, ja tulla johdatetuksi valitsemaan parhaiten sopivimman yksittäisen testaustekniikan tai testaustekniikoiden yhdistelmän. Eräänä haasteena tällaisen teorian luomisella hän näkee sen, että erilaisten testauksen lähestymistapojen rajoitteiden ymmärtämiseksi on tehtävä vielä runsaasti tutkimustyötä.

Tehokkuusmaksimoinnin saavuttamisen hankalaksi tekevistä syistä hän mainitsee tietojärjestelmien jatkuvasti kasvavan monimutkaisuuden. Tämän tukipyyntöjenkäsittelyjärjestelmän tapauksessa kyse on varsin pienestä tietojärjestelmästä, mutta jo sellaisenkin kattava testaaminen vaatisi satoja työtunteja.

5.1 Kriittiset testit

Tukipyyntöjenkäsittelyjärjestelmän versiota 1.0 varten laadittiin testaussuunnitelma, jonka mukaisesti on käyty läpi tietyt kriittiset testit, joiden oli välttämätöntä mennä läpi hyväksyttävästi. Kyseiset testit keskittyivät toiminnallisuuden, käyttöliittymien ja suorituskyvyn testaamiseen, sillä näiden testien tulosten perusteella oli mahdollista analysoida SmartGWT:n sopivuutta tällaisen tietojärjestelmän kehittämiseen.

Ennen näiden alle kymmenen kriittisen testin suorittamista, jotka itsessään sisälsivät lukuisia yksittäisiä testitapauksia, kirjoitettiin 29-sivuinen testaussuunni-

telma, luotiin jokaista testiä varten testiohjeistus, testiraportin pohja, sekä tarvittaessa generoitiin valmiiksi tietyissä testeissä käytettävät syötetiedot.

Näiden testien lisäksi muodostettiin lukuisia testivinkkejä erilaisista testityypeistä, jotka soveltuvat luodun järjestelmän testaamiseen. Nämä testivinkit on kokeiltu käytännössä ja ne sisältävät ohjeistusta kyseisten testityyppien soveltamiseen SmartGWT-pohjaisille sovelluksille yleensä. Testityypeistä on käsitelty muun muassa yksikkötestausta, suorituskykytestausta, staattista analyysia ja erilaisten mittareiden käyttöä.

Tämän opinnäytetyön tekemisen ohella on käyty Ohjelmistotestaus-kurssia, joka täydensi suuressa määrin opinnäytetyön tekijän aiempaa tietämystä testaamisesta. Testaus on ohjelmistotuotannon osa-alue, jota ei pidä jättää erilliseksi, toteutuksen loppuvaiheilla suoritettavaksi. Mitä aikaisemmin testausta alkaa suunnitella (systemaattisesti), sitä tarkempia arvioita pystyy tekemään testaukseen kuluvasta ajasta, sekä sitä aikaisemmin tullaan löytämään ongelmakohdat ja mahdolliset, koko kehitystyön pysäyttävät esteet.

5.1.1 Kriittisten testien suorittamisesta ja testiohjeistuksesta

Testejä suoritettiin kehitysympäristössä, sekä myös (tai pelkästään) Java-hosting -palveluita tarjoavan tahon (Daily Razor) laitteistolla. Daily Razor tarjosi kuukausivuokralla käyttöön Linux-ympäristön ja kohdejärjestelmää vastaavan sovelluspalvelimen (Tomcat 6.0). Palvelu oli alun perin hankittu demokäyttöön. Linuxin kernelin versio oli 2.6.27.18-22 ja sovelluspalvelin oli privaatti-tyyppinen eli se oli konfigurointia myöten vain yhden Daily Razorin asiakkaan eli testaajan käytössä.

Vuokrapalvelun hallintapaneelin kautta pääsi hallinnoimaan MySQL-pohjaista tietokantaa phpAdminilla, mutta tarvittaessa tietokantaa saattoi käsitellä myös SSH-yhteyksien kautta, MySQL:n komentorivipohjaisia käskyjä käyttäen (tätä tarvetta ei esiintynyt). Testauksessa käytettiin selaimia Firefox, Internet Explorer, Opera, Safari ja Chrome.

Testit vaativat tiettyjen esivalmistelujen suorittamisesta ja tiettyjen ehtojen täyttymistä:

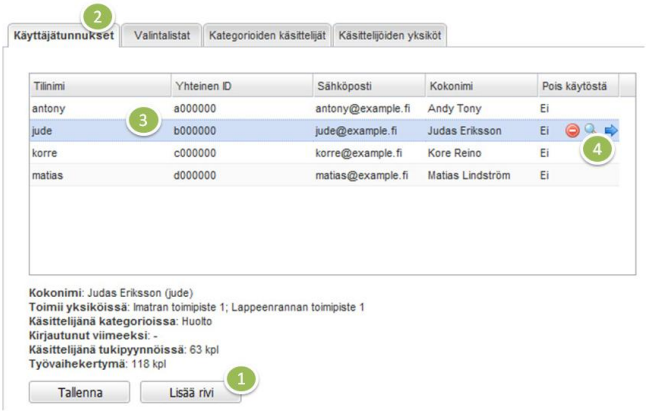
- selaimessa täytyi olla JavaScript-tuki päällä ja evästeitä täytyy voida tallentaa.
- välityspalvelimien käyttö ei ollut suositeltavaa
- erityisiä selainplugineita (esimerkiksi Java-tuki) ei tarvittu, mutta tietyissä testeissä oli tarpeen asentaa Firefoxiin Selenium- ja FireBug-lisäosat (eng. addon)

Testaussuunnitelmaan kirjoitettiin tietyistä huomioitavista asioista, jotka olisivat saattaneet vaikuttaa testien tuloksiin. Käytettävyys- ja toiminnallisuustestien aikana testaajan oman käyttöjärjestelmän ylimääräiset taustaprosessit oli asetettava normaalitasolle. Testaajan täytyi sammuttaa koneeltaan muutkin mahdolliset testin aikana tarvitsemattomat ohjelmat, kuten P2P-verkkoja käyttävät ohjelmat ja ohjelmat, jotka voisivat aiheuttaa testien aikana ylimääräistä tietoliikennettä. Jos oli oletettavaa, että testien aikana jokin testikäyttäjän käyttöjärjestelmän tausta-ajossa ajettava prosessi olisi allokoitu käynnistymään testien aikana, oli joko testauksen ajankohta ajoitettava toisin tai kyseisen häiriötekijän päällekytkeytyvyys estettävä. Myös virustorjuntaohjelmat oli syytä kytkeä pois päältä testien ajaksi.

Kutakin testiä varten luotiin oma kuvaus-, pöytäkirja ja raportointitiedostonsa, sekä mahdolliset liitetiedostot ja syötteenä käytettävä data. Esim. testejä 4.0 – 4.3 varten oli käytettävissä tiedostot:

testidata 4.1 - 4.3 - testisyötteet.txt
testidata 4.1 – 4.3 - validointikohteet - hallinnoija.jpg
testidata 4.1 - 4.3 - validointikohteet - käsittelijä.jpg
Testausohjeet 4.1 - 4.3 - käyttöliittymän testaus.docx
Testausraportti 4.1 - 4.3 - käyttöliittymän testaus.xlsx
testauspöytäkirja 4.0 - 4.3 - käyttöliittymän testaus.docx

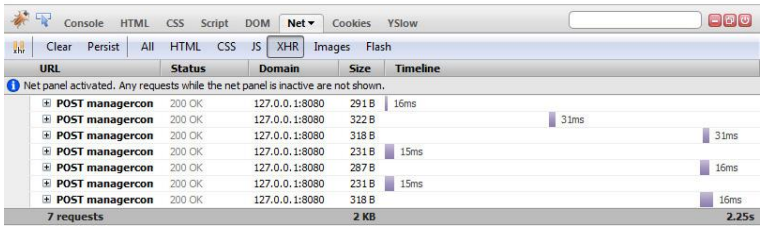
Testausohjeeseen liittyvässä testidata-tiedostossa saattoi olla täydentäviä ohjeita liittyen siihen, mihin tarkalleen olisi kiinnitettävä huomiota esimerkiksi käyttöliittymää testattaessa. Ohjeiden eri vaiheisiin saattoi olla liitettynä kuva, joka oli tarpeen ohjeen tekstiosuudessa viitattujen kohteiden paikallistamiseksi. Tälläinen oli esimerkiksi kuva 36.



Kuva 36. Käyttöliittymän testaamiseksi oli luotu yksityiskohtaiset ohjeet

Testausohjeet- ja testausraportti-dokumentit käyttivät omaa mallidokumenttiaan dokumentin tekstin runkona. Testausohjeet sisälsivät alaotsikot: Tunnistetiedot, Tarkoitus, Esivalmistelut, Testausohje ja Raportointiohje. Testausohjeissa saatettiin kuvailla pitkästikin mitä ja miten jotain testataan:

> *Tässä testissä ei kirjata ylös eksakteja arvoja kuvastamaan sitä aikaa, joka jonkin verkon yli tietoa hakevan toiminnon suorittamiseen kuluu, vaan vain sitä, pysyvätkö nämä ajat tietyissä rajoissa. Lokaalisti ajat ovat pääsääntöisesti muutaman kymmenen millisekunnin luokkaa. Verkossa olevaa palvelinta käytettäessä aikaa kuluu 150 – 200 millisekuntia enemmän (latenssi).*

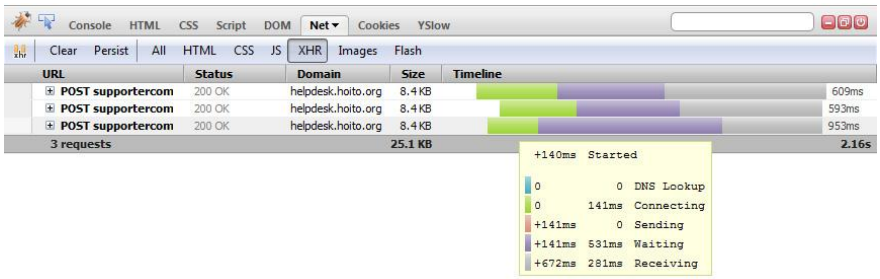


Kuva 37. Vasteaikojen mittausta FireBugilla

> *Testiä FireBugilla suoritettaessa on valittava FireBug-paneelista välilehti Net ja sieltä suodattimeksi XHR-mittari eli se, joka mittaa verkkotoimintoihin kuluvia aikoja. Testissä tarkastellaan pelkästään AJAX-pyyntöjä eli ei mitata sitä kuinka paljon koko sivun lataus kestää. Käyttöliittymien tultua kertaalleen ladatuiksi, eivät ne sen jälkeen hae verkosta kuin pieniä määriä tietoa kerrallaan.*

Lisäksi saatettiin antaa havainnollisia esimerkkejä siitä, minkälainen näkymä testaustilanteessa tulee olemaan esillä, mistä esimerkkeinä kuvat 37 ja 38. Esimerkiksi vasteaikojen tarkkailemiseen tarkoitetusta Firefoxin FireBug-lisäosasta on liitetty mukaan tekstuaalisen selvityksen lisäksi muutamia kuvakaappauksia.

Joissakin tapauksissa tiedonhaun kokonaisaika voi tuntea suurehkolta, mutta olennaisin tarkasteltava ajanjakso on odotteluaika (Waiting) eli se aika, joka palvelimelta kestää prosessoida pyyntö. Alla on esimerkinomaisesti ajettu samaa hakuehtoa kolme kertaa peräkkäin, normaalin käsittelyajan ollessa n. 270 millisekuntia (viimeinen haku kesti kuvan mukaan jopa 541 millisekuntia). ***Tarkastele siis erityisesti violettia palkkia.***



Kuva 38. Vasteaikojen mittausta FireBugilla

5.1.2 Testien suoritusjärjestyksestä ja ei-testattavista ominaisuuksista

Taulukko, johon oli merkitty testikoodeittain jokainen testi, joka alun perin oli aiottu suorittaa kattavasti, sisälsi sarakkeet: koodi, testityyppi, selite, kollektiivinen tulos, testiympäristö, pakollinen, toistokerrat, selaimet, työaika (h), korvattavissa teorioinnilla, sekä sarakkeen "pöytäkirjan ja raportin kirjoittaja voivat olla eri henkilöitä". Seuraavassa taulukossa (Taulukko 3) on esillä vain osa näistä sarakkeista. Luettavuuden vuoksi taulukosta on poistettu myös muutama rivi. Suoritettujen testien suoritusjärjestyksellä ei ollut merkitystä, mutta testikohtaisesti saattoi olla asetettuna järjestys testitapausten läpikäymiselle.

Testityyppi sinänsä ei viittaa mihinkään sellaiseen testiin, joka olisi aina sama erilaisten sovellusten kohdalla, vaan se on aina sovelluskohtaisesti muodostettu. Turvallisuuden ja tietokannan testaus päätettiin siirtää myöhempään, testiä 1.0 lukuun ottamatta, koska ne eivät suoranaisesti liittyneet SmartGWT:n testaamiseen (liittyi kolmitasorakenteen sovellusosaan eli palvelimenpuoleiseen osaan sovelluksesta, SmartGWT:n vaikuttaessa enimmäkseen esitystasolla) ja toisekseen ne on aiottu suorittaa kattavasti projektin jatko-osassa (projektisuunnitelmaan tehtiin ohjausryhmän palaverissa muutos, jonka nojalla järjestelmän versiosta 1.0 eteenpäin kehitystyö saattaa jatkua Projektityö-kurssin puitteissa).

Taulukko 3. Muutamia kriittisiksi luokitelluista testeistä

Koodi	Testityyppi	Selite	Kollektiivinen tulos	työaika (h)
1.0	Tietokannan testaus	Tietokannan luominen tauluineen ja testisisällön syöttäminen	Onnistui / Ei onnistunut	1
2.0	Toiminnallisuustestaus	Välilehdillä erotettu toiminnallisuus (hallinnoija)	Toiminnallisuudet toteutettu / Lieviä puutteita / Vakavia puutteita	2
4.2	Käyttöliittymän testaus	Lomakkeiden tekstikenttien validoinnin toimivuus (käsittelijä)	Kaikki ok / Puutteita	1
4.3	Käyttöliittymän testaus	Ruudukoiden solujen validoinnin toimivuus (käsittelijä)	Kaikki ok / Puutteita	1
5.0	Käyttöliittymän testaus	Lomakekomponenttien ja muiden elementtien asemoituvuus eri selaimilla.	Kaikki ok / Muutamia lieviä poikkeavuuksia / Vakavaa sekavuutta esiintyy	3
6.0	Käyttöliittymän testaus	Viiveellisyys (välilehdillä siirtyminen, ruudukoiden päivitys)	Hyvät / Tahmaisuutta esiintyy / Käyttökelvoton	4
7.0	Turvallisuuden testaus	Parametritarkistuksesta vastaavan koodin katselmointi	Kaikki ok / Vähäisiä puutteita / Vakavia puutteita	6
9.0	Suorituskykytestaus	Vasteaikojen havainnointi	Normaali / viivettä / hitautta	3

Testit pystyi suorittamaan yksikin henkilö, mutta testaamisen nopeuttamiseksi olisi useamman testaajan käyttö ollut toimiva ratkaisu esimerkiksi eri selaimilla testattaessa. Testeistä useimmat eivät vaatisi erityistä ohjelmistotuotantoalan koulutusta, vaan riittäisi, että noudattaa testeissä kerrottua suoritusjärjestystä ja testikohtaista ohjeistusta, sekä osaisi tietokoneen ja yleisempien toimisto-ohjelmien käytön perusteet.

Testejä 7.0 – 7.1 (Turvallisuuden testaus) ei pystyisi suorittamaan ilman Java-ohjelmointikielen hyvätasoista tuntemusta ja säännöllisten lausekkeiden perusteiden hallitsemista. Eniten suorituskykyä vaativia toimintoja ovat tukipyyntöjen listaaminen ja hakujen tekeminen tukipyynnöistä. Kokonaisaika muodostuu verkon latenssista, yhteyden muodostamiseen kuluvasta ajasta, tiedon prosessoinnista palvelimella, tiedon siirtämisestä verkon yli, tiedon prosessoimisesta asiakaspäätteelle ja tiedon esittämisestä ruudulla. Tätä testattiin testeissä 9.0 – 9.1. Testitulokset kirjattiin ylös raporttiin, joka sisälsi alaotsikot: johdanto, ristiriidat ja poikkeamat, kattavuustarkastelu, tulokset, arviointi ja hyväksyminen.

Käsitys SmartGWT:n käyttökelpoisuudesta tukipyyntöjenkäsittelyjärjestelmän toteuttamiseen selkiytyi erityisesti toiminnallisuus-, suorituskyky- ja käyttöliittymätestien systemaattisen, eri selaimella suoritetun testaamisena tuloksena. Pelkästään niiden perusteella ei kuitenkaan voida saada asiasta täyttä varmuut-

ta, sillä tarkemmassa analyysissä on otettava huomioon esimerkiksi ohjelmistokehitysympäristön tarjoama tuki SmartGWT-pohjaisten sovellusten kehittämiselle. Näistä muista aspekteista kerrotaan tarkemmin tämän opinnäytetyön loppupäätelmissä (luvussa 7).

5.1.3 Tuloksia toiminnallisuustestauksesta ja käyttöliittymien testauksesta

Toiminnallisuustestauksen ja käyttöliittymätestauksen erotteleminen omiksi testeikseen oli perusteltua, sillä tällä tavoin testaajan ei tarvitse jatkuvasti vaihdella kahden hiukan erilaisen orientaatiotilan välillä. Tämä tarkoittaa sitä, että toiminnallisuus- ja käyttöliittymätestejä tehtäessä tultiin molemmissa käyneeksi läpi samat käyttöliittymien näytöt, mutta niistä havainnoitiin eri asioita.

Olisi ollut tehokkuuden kannalta huono ratkaisu tarkastella ensin sitä onko jokin toiminto toteutettu ja heti sen jälkeen tarkastella samasta näkymästä ovatko komponentit visuaalisesti sen näköiset kuin pitääkin ja ovatko ne oikein asemoituneet – ja seuraavassa näkymässä sama uudelleen. Suoritteena se ei ole erityisen raskasta, mutta pidempään jatkuessaan sellaiseen testaukseen tuskastuu nopeasti, mikä voisi ilmetä väärinä havaintoina, testauskohteiden huomioimattomuutena ja muina testausprosessista poikkeamisina. James Reason sijoittaa (1990, 53) nämä yleisessä virheidenmallintamissysteemissään (Generic Error Modeling System, GEMS) kategoriaan taito-pohjaiset virheet (eng. skill-based errors). Inhimillisistä virheiden osuutta on selvitetty enemmän käyttöliittymien suunnittelun teoriasta kertomisen yhteydessä, alaluvussa 3.1 ("Käyttöliittymien suunnittelun teoriaa").

Testaajana toimi opinnäytetyön tekijä itse, sillä syksyllä alkaneella Ohjelmistotestaus-kurssilla, jota opinnäytetyön tekijä itse suoritti kesän ja syksyn aikana, ei ollut käytettävissä ketään sellaista testaajaa, joka olisi ollut perehtynyt Java-ohjelmointikieleen riittävästi (C#:iin ja PHP:hen kylläkin). On tapana, että järjestelmän tekijä itse ei testaisi tuotostaan, vaan sen tekisi joku muu, mutta muut kriittiset testit eivät olisi olleet heille riittävän haasteellisiakaan.

Mahdollisessa Projektityö-kurssin puitteissa suoritettavassa järjestelmän jatkokehityksessä määritellään monia testejä, joiden suorittaminen vaatii runsaasti

perehtyneisyyttä testausteknologioihin ja testaustyökaluihin. Osasta näitä testejä kerrotaan lisää luvussa 5.2 (''Soveltuvia testejä ja testityyppejä '').

Toiminnallisuustestaus oli nopea käydä läpi ja se suoritettiin kaikilla yleisimmillä selaimilla (Firefox, Internet Explorer, Safari ja Chrome). Testi oli tärkeää suorittaa eri selaimilla, koska niiden JavaScript-implementaatio saattaisi joiltain osin toimia eri tavoin (SmartGWT-pohjaiset sovellukset ovat ajettaessa JavaScript-pohjaisia, vaikka itse ohjelmakoodi Java-ohjelmointikielisenä kirjoitetaankin). Toiminnallisuustestauksen osalta voidaan todeta, että kaikki muut toiminnot oli toteutettu, suhteessa toiminnallisen määrittelyn dokumentaatiossa määriteltyyn, paitsi raportointi-toiminto.

Operalla toiminnallisuustestaus suoritettiin vasta käyttöliittymätestauksen jälkeen, sillä käyttöliittymätestauksen aikana selvisi syy sille, miksi Opera näytti pelkästään tyhjiä valkeita sivuja ja Internet Explorer 8:lla visuaalisia häiriöitä oli liikaa. Syynä oli se, että käyttöliittymän käynnistävän HTML-tiedoston alussa oli seuraavanlainen doctype-määritys (aina ensimmäisellä rivillä):

> *<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">*

Tämän kanssa Operalla ei näkynyt mitään ja IE 8 ja IE 9 beta osoittivat visuaalisten häiriöiden merkkejä (elementit asemoituivat poikkeavasti, sekä niiden marginaalit ja reunukset saattoivat olla väärin). Jos tuon määrityksen otti kokonaan pois, alkoi Operallakin toimia kuten oli tarkoitettu. Firefoxilla, Chromella ja Safarilla doctypen läsnäolemisella tai poissaolemisella ei ollut vaikutusta testin lopputulosten kannalta. Doctypen ideana on kertoa selaimelle, miten sen pitää käsitellä HTML-sivunkuvauskieltä ja miten selaimen pitäisi elementit ruudulle piirtää. SmartGWT:n oma suositus on (Isomorphic Software 2010b), että ei käytettäisi doctypeä ollenkaan. World Wide Web Consortium (W3C), joka on kansainvälinen yritysten ja yhteisöjen yhteenliittymä, ja joka ylläpitää ja kehittää verkkoteknologioiden suosituksia, pitää doctypen käyttöä suositeltavana (World Wide Web Consortium 2010). Ilman sitä selain "tekee arvauksen" siitä, miten sen pitäisi HTML-lähdekoodia käsitellä.

Käytännössä visuaalisten häiriöiden ongelma ratkesi yksinkertaisella tavalla, joka vaati vain sen, että käyttöliittymän käynnistävään HTML-tiedostoon lisätään seuraavanlainen meta-tagi:

<meta http-equiv="X-UA-Compatible" content="IE=7">

Tämä asettaa yhteensopivuus-moodin, joka mahdollistaa sen, että uudemmilla IE-selaimilla käytetään muun muassa elementtien asemointiin Internet Explorerin versio 7:ään rajoittuvia ominaisuuksia. Ratkaisuna tämä toimikin siinä suhteessa erinomaisesti, että visuaalisista häiriöistä ei jäänyt jäljelle yhtäkään, mutta samalla menetettiin Windowsin mahdollistama ClearType-ominaisuus (kirjasimien antialiasiointi). Koska kyse oli räätälöidyn tietojärjestelmän käyttöliittymästä, eikä tarvetta IE 7:ää uudempien selaimien tarjoamille ominaisuuksilla varsinaisesti ollut, oli ratkaisu sinänsä hyväksyttävissä. Seuraavassa taulukossa (Taulukko 4) on ote testaustuloksista:

Taulukko 4. Ote käyttöliittymätestauksen tuloksista

	FF	IE 7	IE 8	IE 8 (2)	IE 9 beta	IE 9 beta (2)	Opera	Safari	Chrome
Uusi tukipyyntö	ok	ok	4,5,7	ok	ok	ok	14	1	1
Lisätyökalut	ok	ok	ok	ok	ok	ok	ok	ok	ok
Aktiiviset	ok	11	7,9	ok	9,16	ok	ok	ok	ok
Passiiviset	ok	11	7,9	ok	9,16	ok	ok	ok	ok
Haku	ok	12	7,9	12	12	12	12	ok	ok

Mainitun meta-tagin käyttöönoton jälkeen jäljelle ei jäänyt jäljelle kuin yksi merkittävä poikkeama, joka esiintyi Safarilla ja Chromella (merkitty numerolla 1). Poikkeama juontui siitä, että nämä selaimet tukevat itsessään mahdollisuutta venyttää sellaista tekstikenttää, johon voi kirjoittaa monirivistä tekstiä. SmartGWT:n osaa kyllä reaaliajassa huomioida esimerkiksi selainikkunan koon muutoksen, mutta tekstikentän koon muuttamiseen se ei reagoi mitenkään. Tästä seuraa, että tekstikenttää laajennettaessa, eivät tekstikentän läheisyydessä olevat komponentit siirry samalla, vaan tekstikenttä voi mennä muiden komponenttien päälle tai alle.

Seuraavassa listassa muutamia testin aikana ylöskirjattuja havaintoja.

- 1: Venytettäessä monirivistä tekstiä sisältäviä kenttiä, eivät kentän alla tai sivulla olevat elementit liiku mihinkään, vaan kenttä tulee niiden päälle tai alle
- 7: Valintalistat ja niiden vieressä olevat ikonit hypähtelevät valittaessa
- 9: Suodattimien valittuna olemista kuvaava reunus on liian paksu
- 10: Väri-solujen värivalinta-nappi kohdistuu liian alas

5.1.4 Tuloksia suorituskykytestauksesta

Suorituskykytestaus on testi josta on johdettavissa erilaisia variaatioita kuten kuormitustestaus, stressitestaus ja konfiguraatiotestaus. Näistä on kerrottu tarkemmin alaluvussa 5.2.3 ("Suorituskykytestaus"). Seuraavassa kerrotaan kuormitustestauksesta, jolla testataan kuinka palvelu suoriutuu normaalista kuormituksesta.

Tuloksista ilmeni (Taulukko 5), että käsittelijän käyttöliittymän rollovereissa (ruudukoissa esiintyvä, koko rivin mittainen palkki, joka liikkuu hiiren osoittimen mukana ja voi sisältää toimintaikoneita) esiintyy Internet Explorer 8:lla lievää viiveellisyyttä, jota ei kuitenkaan esiinny hallinnoijan käyttöliittymissä. Tämä voi johtua siitä, että sarakkeita ja rivejä on ollut enemmän, sillä SmartGWT tekee eri toimintojen yhteydessä kaikkiin soluihin kohdistuvia tarkistuksia ja nopeassa rolloverin liikuttamisessa voi ilmetä tarkistuksien pinoutumista, joka ilmenee käyttäjälle toiminnan viiveellisyytenä.

Taulukko 5. Suorituskykytestauksen tulokset

KÄSITTELIJÄN KÄYTTÖLIITTYMÄ

FF (paikallinen)	**Rolloverit**			**Ruudukon selaus**			**Välilehdillä siirtyminen**		
	Nopea	Viivettä	Tahmea	Nopeaa	Viiveellistä	Tahmeaa	Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset	x			x					
Passiiviset	x			x					
/							x		
/Tukipyynnöt							x		
/Tukipyynnöt/Käsittele tukipyyntöä							x		

IE (paikallinen)	**Rolloverit**			**Ruudukon selaus**			**Välilehdillä siirtyminen**		
	Nopea	Viivettä	Tahmea	Nopeaa	Viiveellistä	Tahmeaa	Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset		x		x					
Passiiviset		x		x					
/							x		
/Tukipyynnöt							x		
/Tukipyynnöt/Käsittele tukipyyntöä							x		

FF (etä)	**Ruudukon selaus**		
	Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset		x	
Passiiviset		x	

IE (etä)	**Ruudukon selaus**		
	Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset	x		

Passiiviset				x		

HALLINNOIJAN KÄYTTÖLIITTYMÄ

FF (paikallinen)	**Rolloverit**			**Välilehdillä siirtyminen ja ruudukon päivitys**		
	Nopea	Viivettä	Tahmea	Nopeaa	Viiveellistä	Tahmeaa
Käyttäjätunnukset	x			-		
Valintalistat	x			x		

IE (paikallinen)	**Rolloverit**			**Välilehdillä siirtyminen ja ruudukon päivitys**		
	Nopea	Viivettä	Tahmea	Nopeaa	Viiveellistä	Tahmeaa
Käyttäjätunnukset	x			x		
Valintalistat	x			x		

FF (etä)				**Välilehdillä siirtyminen ja ruudukon päivitys**		
				Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset				x		
Passiiviset				x		

IE (etä)				**Välilehdillä siirtyminen ja ruudukon päivitys**		
				Nopeaa	Viiveellistä	Tahmeaa
Aktiiviset					x	
Passiiviset					x	

Vuokrapalvelinta käytettäessä voitiin havaita, että ruudukoiden selaus (uuden tiedon haku ja sen näyttäminen) kävi hitaammaksi verrattuna paikallisen palvelimen käyttämiseen. Toisaalta järjestelmä on tarkoitettukin käytettävän lokaalisti, eikä pitkien verkkoyhteysvälien kautta. Välilehdillä siirtyminen ei ollut erityisen paljon hitaampaa IE:llä kuin mitä se on Firefoxilla.

Tulokset olivat täsmälleen sen mukaisia kuin testiohjeessa oletettiin niiden olevan. Toisin sanoen palvelin ehtii prosessoimaan yksittäiset Ajax-pyynnöt riittävällä nopeudella. Tämä ei ollut testi, jossa mitattaisiin eksakteja arvoja (millisekunteja), vaan testitulokset perustuivat käyttötuntumanaan. Lokaalisti hakuajat pysyttelivät alle 100 millisekunnin, useimmiten paljonkin sen alle, muutamissa kymmenissä millisekunneissa. Tämä päti sekä hallinnoijan, että käsittelijän käyttöliittymiin.

Vuokrapalvelimella varsinaiset prosessointiajat käsittelijän käyttöliittymässä olivat tyypillisesti luokkaa 150 – 200 millisekuntia (tukipyyntöruudukot). Hallinnoijan käyttöliittymien osalta kaikki prosessointiajat sijoittuivat samaan aikaluokkaan.

Hakujen kohdalla lisäystä tuli keskimäärin korkeintaan muutaman kymmenen millisekunnin verran. Tukipyynnön hakeminen muokattavaksi aiheutti kolme peräkkäistä kutsua, joista kukin oli ajallisesti samaa luokkaa kuin tyypillinen yksittäinen kutsu. Poikkeuksellisen kauan kestäviä pyyntöjen käsittelyjä ei esiintynyt missään vaiheessa.

5.2 Soveltuvia testejä ja testityyppejä

Tutkimusten mukaan (Kushwaha 2008) ohjelmistoa kehitettäessä jopa 50 % kustannuksista voi olla omistettuna testaamiselle – erityisen kriittisissä ohjelmistoissa enemmänkin. Seuraavista testausvinkeistä jo ensimmäinen, yksikkötestaus, voisi pienenkin tietojärjestelmän tapauksessa muodostua tehtäväksi, joka veisi ajallisestikin enemmän aikaa kuin mitä kului testauksen kohteena olevan ohjelmakoodin tuottamiseen, mikä jo sisänsä kertoo paljon testaamisen merkityksestä ohjelmistotuotannossa. Erilaisia testitapauksia on mahdollista kehitellä lukematon määrä, joten testien suunnittelijan on osattava arvioida, mikä on riittävää ja pysytellä niissä rajoissa.

Seuraavat testivinkit on muodostettu ajatellen kohteena olevan ensisijaisesti toteutettu tukipyyntöjenkäsittelyjärjestelmä, mutta ne soveltuvat myös SmartGWT-pohjaisten sovellusten testaamiseen yleensä – ja tietenkin myös Google Web Toolkit -pohjaisten sovellusten testaamiseen, SmartGWT:n pohjautuessa siihen. Tässä luvussa esiteltyjen testityyppien ja testitapauksien ei ole tarkoitus osoittaa kaikkia soveltuvia sellaisia, vaan valikoiman kattavuuteen on vaikuttanut merkittävästi käytettävissä ollut aika. Erilaisia verkkosivujen ja -palveluiden testaukseen käytettävissä olevia testityökaluja on listattu esimerkiksi SoftwareQATest.com:n verkkosivuilla lähes 500 kappaletta eli pelkästään niiden lataamiseen ja asentamiseen kuluisi runsaasti aikaa.

5.2.1 Yksikkötestaus

JUnit

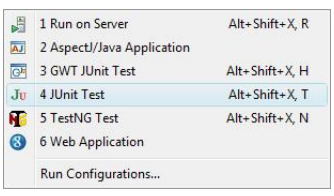
Koska suuri osa Google Web toolkit -pohjaisista sovelluksista on kirjoitettu kauttaaltaan Java-ohjelmointikielellä, on mahdollista yksikkötestata suuri osa sellaisesta käyttäen JUnit-testauskehystä. Yksikkötestauksella tarkoitetaan "hyvin matalan tason testausta, jossa varmistutaan metoditasolla siitä, että koodi toimii ohjelmoijan tarkoittamalla tavalla" (Sininen Meteoriitti). Käytännössä yhden metodin testaamiseen tarvitaan yhtä useampia parametrien variaatioita, jotta voidaan varmistua siitä, että testauksen kohde tuottaa tietynlaisella syötteelle juuri sellaisen lopputuleman kuin on tarkoitettukin.

Testivetoisessa kehityksessä (Test Driven Development, TDD) yksikkötestien kirjoittaminen edeltäisi testattavaa koodia, lyhyin ja toistuvin iteraatioin (Janzen 2005), mutta tukipyyntöjenkäsittelyjärjestelmän tapauksessa tällainen lähestymistapa ei olisi ollut kovinkaan suositeltava, koska todennäköisyys testauksen kohteen merkittävälle muuttumiselle kehityksen aikana oli korkea. Toiminnallisuuksien rajaamisen vaikeudesta kooditasolla on kerrottu tarkemmin alaluvusta 4.4.4 ("Sovelluksen automatisoidusta takaisinmallintamisesta kaavioiksi").

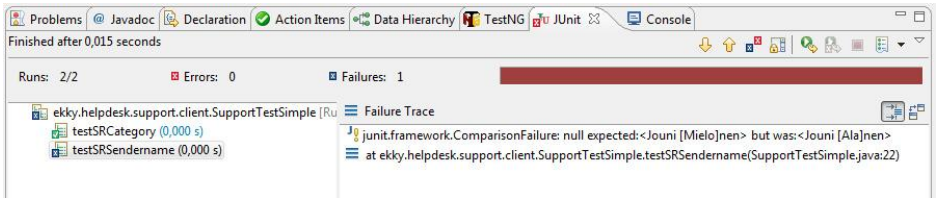
Java-pohjaiselle ohjelmakoodille voi yksikkötestejä suorittaa nopeasti suoraan Eclipse-kehitysympäristössä, joko yksi kerrallaan tai testijoukkoina. Varsinaista sovellusta ei siis käynnistetä, vaan kohteena ovat sovelluksen rakennusosina toimivat luokkien metodit. Seuraavan esimerkin on tarkoitus osoittaa, kuinka yksinkertaista testaus periaatteessa voi olla:

```
import junit.framework.TestCase;

public class SupportTestSimple extends TestCase {

  private SupportRequestMini supportrequest = new SupportRequestMini();
  private SupportRequestMini supportrequest2 = new SupportRequestMini();

  public void testSRCategory() {
        supportrequest.setCategory("tietokoneet");
        assertEquals("tietokoneet", supportrequest.getCategory());
  }
  public void testSRSendername() {
        supportrequest2.setSendername("Jouni Alanen");
        assertEquals("Jouni Mielonen", supportrequest2.getSendername());
  }
}
```

Testi ajettiin valitsemalla testattavan luokan kohdalla valikosta tai kontekstimenusta "Run As" ja sen alta "Junit Test" (Kuva 39).



Kuva 39. JUnit-testin käynnistäminen



Kuva 40. JUnit-testin tulokset, joista ilmenee, että toinen testitapauksista ei mennyt läpi.

Kyseinen testi kesti vain 0,0015 sekuntia. Testitapauksista toinen meni läpi, mutta toisen suorituksessa ilmeni virhe (Kuva 40), jonka syistä JUnit pyrkii kertomaan. Jos pyrkimyksenä on testata sitä, miten SmartGWT:n komponentit toimivat ajonaikaisesti (esimerkiksi testi, joka testaa saiko ruudukkokomponentti ladattua juuri ne tietorivit, jotka haluttiin), täytyy luokan periä GWTTestCase-luokka (sisältyy Google Web Toolkittiin). JUnit 3:a käytettäessä perii testattava luokka TestCase-luokan, mikä verrattuna GWTTestCasen käyttöön eroaa testien kirjoittamisen osalta vain yhdellä tapaa (Google 2010): luokan täytyy ylikuormittaa metodi getModuleName.

```
@Override
public String getModuleName() {
  return "ekky.helpdesk.support.Support";
}
```

Käytännössä testejä luotaessa voi joutua käärimään testattavan luokan vielä yhden luokan sisään, luoden kohdeluokasta samalla uuden instanssin, jotta olisi mahdollista periyttää joko luokka TestCase tai GWTTestCase. Java-ohjelmointikielessä ei ole mahdollista moniperiyttää luokkaa, joten jos oma luokka on jo periyttänyt SmartGWT:n komponenttiluokan, joka on todennäköisesti itsekin periyttänyt jonkin muun luokan, ei muuta vaihtoehtoa ole. Jos yrittää testata asiakaspuolen koodia eli JavaScript-koodiksi käännettävää koodia käyttämällä periytettävää luokkaa TestCase, GWTTestCasen sijaan, on seurauksena virheilmoitus:

> *Caused by: java.lang.UnsupportedOperationException: ERROR: GWT.create() is only usable in client code! It cannot be called, for example, from server code. If you are running a unit test, check that your test case extends GWTTestCase and that GWT.create() is not called from within an initializer, constructor, or setUp()/tearDown().*

Seuraava esimerkki demonstroi sitä, kuinka testattava luokka on käärittty testiluokan sisään:

```
public class SupportTestLogGrid extends GWTTestCase {

  @Override
  public String getModuleName() {
        return "ekky.helpdesk.support.Support";
  }

  public void testLogGrid() {

        SimpleLogGrid simpleLogGrid = new SimpleLogGrid();
        ArrayList<HashMap<String, String>> logRows = new ArrayList<HashMap<String,
String>>();

        HashMap<String, String> logRow1 = new HashMap<String, String>();
        logRow1.put("id", "16");
        logRow1.put("logmoment", "2010-08-15 14:45:35.0");
        logRow1.put("logitem", "Pääkäsittelijäksi vaihtunut Matias Lindström (aiemmin Matias
Lindström)");

        HashMap<String, String> logRow2 = new HashMap<String, String>();
        logRow2.put("id", "25");
        logRow2.put("logmoment", "2010-08-15 10:29:12.0");
        logRow2.put("logitem", "Uusi käyttäjän lähettämä tukipyyntö koodilla TP5467");
        logRows.add(logRow1);
        logRows.add(logRow2);

        simpleLogGrid.fillListgrid(logRows);

        assertEquals(2, simpleLogGrid.getRecords().length);
        assertEquals("25", simpleLogGrid.getRecord(1).getAttribute("id"));
  }

}
```

Kun GWTTestCase-periytettyä luokkaa testataan, käynnistyy GWT-sovelluskehys näkymättömässä hosted-moodissa, jossa testitapaukset evaluoidaan. Näkymättömyys tarkoittaa tässä sitä, että sovellusta ajetaan muuten normaalisti, mutta käyttöliittymää ei tuoda esiin. Kaikki ei-näkyvän selaimen tarjoamat mahdollisuudet ovat testitapauksen käytettävissä: voidaan ajaa natiiveja JavaScript-metodeita, renderoida komponentteja tai suorittaa asynkroninen etä-

kutsu (Google 2010). On kuitenkin syytä huomioida, että hosted-moodin käynnistymisessä on jokaisella testauskerralla tietty viive (lähdekoodi täytyy kääntää ajettavaksi JavaScript-koodiksi, jollaista sovelluksen asiakaspuoli kokonaisuudessaan ajon aikana on).

Lisäksi jokaisen yksittäisen testitapauksen kohdalla näkymätön (headless) selain on suljettava ja käynnistettävä uudelleen, mihin kuluu muutamia sekunteja. Tämän välttämisessä auttaa testien ryhmittely testijoukkoihin seuraavanlaisesti:

```
public class SupportTestSuite extends GWTTestSuite {
  public static Test suite() {
        TestSuite suite = new TestSuite("Muutama käsittelijän moduulin testi");
        suite.addTestSuite(SupportTestSimple.class);
        suite.addTestSuite(SupportTestMockTestSimple.class);
        suite.addTestSuite(SupportTestLogGrid.class);
        return suite;
  }
}
```

Siinä vaiheessa kun varsinaisen ohjelmointikoodin voi katsoa olevan riittävän vakautunut, voidaan yksikkötestien joukoksi määrittää sellainen, joka testaa toiminnallisen määrittelyn mukaisia käyttötapauksia. Tämä ei välttämättä ole kaikissa tapauksissa mahdollista ja joissain tapauksissa rajanveto voi olla erittäin työlästä, mutta siitä olisi paljon hyötyä regressiotestauksessa eli sen jälkeen, kun jotain osaa ohjelmasta on muutettu ja halutaan varmistua, että tietty toiminnallisuus toimii edelleen kuten on tarkoitettu.

Jos testitapaus vaatii etäkutsun (RPC) suorittamista, on testimetodi määriteltävä hivenen toisin. Tämä vaatimus syntyy siitä, että etäkutsut ovat aina asynkronisia, eikä testi itsessään jää odottelemaan vastausta palvelimelta, vaan testi katsotaan päättyneeksi jo ennen kuin vastausviesti ehtii perille. Tämän vuoksi etäkutsuja käyttävien testien kohdalla on erikseen tehtävä tiettyä viivytystä ja kerrottava eksplisiittisesti (Google 2010d), milloin kyseinen testitapaus on päättynyt.

TestNG

TestNG:n kehittäjä on tietyssä JUnitin kehityksen vaiheessa (vuonna 2004) kokenut sen olevan monilta osin riittämätön (Beust 2004) vaateliaan testaajan tarpeisiin, minkä seurauksena alkoi testauskehys TestNG:n kehittäminen (kehittäjätahot eivät ole samat). Tämän opinnäytetyön esimerkeissä on viitattu lähinnä JUnitin versioon 3, eikä tässä yhteydessä ole tarkoituksena käydä vertailemaan näitä testauskehyksiä kovin tarkasti.

JUnitin versioon 4 on jo saatu mukaan annotaatiot, mutta TestNG vaikuttaa olevan niidenkin osalta joustavampi. Erityisen hyödyllisiltä vaikuttavat metodien testimetodeiksi merkkaamisen yhteydessä annettavissa olevat parametrit tai parametrien datalähteen asettaminen. Käytännössä tämä mahdollistaa yksittäisten testiparametrien ylöskirjaamisen metodin läheisyyteen, josta ne testin ajamisen yhteydessä poimitaan testattavaksi, minkä lisäksi mahdolliseksi tulee myös parametrigeneraattorin luominen.

```
@DataProvider(name = "someparameters")
public Object[][] makeSomeParameters() {
	HashMap<String, String> filtersHashMap = new HashMap<String, String>();
	filtersHashMap.put("state", "open");
	filtersHashMap.put("category", "36#7F34");
	return new Object[][] { { 0, 14, filtersHashMap, "active", false } };
}

@Test(dataProvider = "someparameters")
public List<SupportRequestFull> fetchRows(int startRow, int amount,
	  HashMap<String, String> filtersHashMap, String side,
	  boolean ownonesbuttonpressed) throws IllegalArgumentException {
```

Huomioi edellisessä koodikatkelmassa metodin palautustyyppi. Sen täytyisi olla void-tyyppinen, mutta ensi kertaa kokeiltaessa tuntui luontevalta käyttää olemassa olevia luokkia siten, että merkitsee vain Test-annotaatiolla metodin testimetodiksi ja käynnistää sitten testauksen. Tämä tuntui luontevalta siksi, että TestNG:tä käytettäessä ei ole tarpeen periyttää mitään luokkaa.

JUnitin ja TestNG:n vertailuja ja näkemyseroja löytyy muun muassa seuraavilta verkkosivuilta:

- http://www.mkyong.com/unittest/junit-4-vs-testng-comparison/
- http://www.ibm.com/developerworks/java/library/j-cq08296/index.html
- http://stackoverflow.com/search?q=junit%2Btestng

Mock-objektit

Mock-objektit ovat simuloituja objekteja, joilla voidaan korvata sellainen osa järjestelmästä, jota on joko hankalaa ottaa sellaisenaan mukaan yksikkötestaukseen tai jota ei ole vielä edes toteutettu, mutta sen käyttäytyminen voidaan mallintaa. Eräs käyttökohde voisi olla etäkutsuun vastaamisesta vastaavan luokan korvaaminen mock-objektilla. Tällöin jolloin etäkutsuun vastaisi mock-objekti siihen ohjelmoidun kontrolloidun käytöksen mukaan. Seuraava esimerkki havainnollistaa miten tämä tehtäisiin JMockilla:

```
public class RPCTest {

  private Mockery context = new Mockery();
  private UserRequestStubAsync service = context.mock(UserRequestStubAsync.class, "ser-
vice");
  final private SupportRequestFull srf = new SupportRequestFull();

  @Test
  public void fetchAsynchronously() throws Exception {

        final AsyncAction<SupportRequestFull> makeAsyncRequest = new AsyncAc-
tion<SupportRequestFull>();
        context.checking(new Expectations() {{
          oneOf(service).fetchSupportRequestDetails("TP3635", with(any(AsyncCallback.class)));
          will(makeAsyncRequest);
        }});

        makeAsyncRequest.succeedGiving(srf);
        context.assertIsSatisfied();
        // muuta testikoodia..
  }
}
```

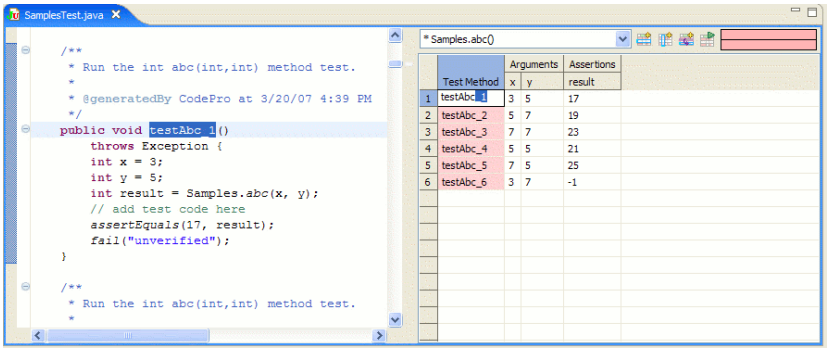
Mock-objektien käyttö voi aluksi tuntua erittäin hankalalta, minkä voi havaita sekä itse kokeilemalla, että päätellä epäsuorasti siitä, että on kehitetty useitakin mock-objektien käyttöä helpottavia vaihtoehtoisia mock-kirjastoja. Helppo, mutta silti toiminnoiltaan monipuolinen on Mockito. Eräs osa-alue, johon siinä on

panostettu, liittyy koodin luettavuuteen, mikä laskee sen käyttöönottokynnystä. Seuraava esimerkki, joka on suoraan Mockiton omista esimerkeistä, ilmentää hyvin tätä koodin luettavuus -aspektia:

```
//You can mock concrete classes, not only interfaces
 LinkedList mockedList = mock(LinkedList.class);

 //stubbing
 when(mockedList.get(0)).thenReturn("first");
 when(mockedList.get(1)).thenThrow(new RuntimeException());

 //following prints "first"
 System.out.println(mockedList.get(0));
  //following throws runtime exception
 System.out.println(mockedList.get(1));
  //following prints "null" because get(999) was not stubbed
 System.out.println(mockedList.get(999));
```

CodePro Analytix

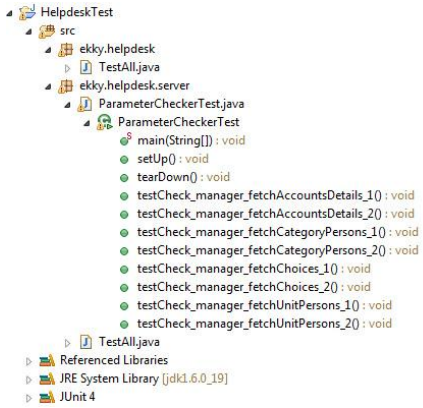
CodePro Analytix on erittäin hyödyllinen testityöväline (Eclipsen plugin), joka koostuu useasta testaustyötä helpottavasta osasta. Yksikkötestauksen kannalta sen oleellisimpia ominaisuuksia ovat testitapauksien automaattinen generointi ja JUnit-testieditori.



Kuva 41. CodePro Analytixin JUnit Test Editor

Yksikkötestieditorin avulla on helppo kirjoitella lukuisia parametrivariaatioita myös muillekin tietotyypeille kuin primitiivisille sellaisille. CodePro Analytixin testitapausten generoimiseen tarkoitettu toiminto on myös erittäin käyttökelpoinen. Alkuun pääsee, kun valitsee testattavan luokan ja käynnistää automaatti-

sen generoinnin. Tämän seurauksena luodaan uusi Eclipse-projekti (Kuva 41), jota uudet generoinnit täydentävät.



Kuva 42. CodePro Analytixin generoima testiprojekti, josta on selkeyden vuoksi poistettu osa testattavista metodeista

Generoitu projekti sisältää sekä yksittäistestit (luokka per testi), että testiryppäiden ajamiseen tarkoitetun TestAll-luokan, jota uudet testitapausgeneroinnit automaattisesti päivittävät. Alla olevasta koodikatkelmasta on nähtävissä, että testirypästä ajettaessa testattaisiin kahden luokan metodit.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
  ParameterCheckerTest.class,
  RandomGeneratorTest.class,
})
public class TestAll {
  /**
   * Launch the test.
   * @param args the command line arguments
   * @generatedBy CodePro at 10.10.2010 0:13
   */
  public static void main(String[] args) {
        JUnitCore.runClasses(new Class[] { TestAll.class });
  }
}
```

5.2.2 Toiminnallisuustestaus ja käyttöliittymätestaus

Toiminnallisuuden ja käyttöliittymien testaamiseen on saatavilla lukuisia testausohjelmia. Näiden ohjelmien ei aina voida sanoa kuuluvan kategorisesti jom-

paan kumpaan, sillä osassa testausohjelmista lähtökohdat ovat sellaisia, että toiminnallisuustestaus ja käyttöliittymän testaus ovat toisistaan erottamattomia. Selvyyden vuoksi poissuljettakoon käyttöliittymän testauksen määritelmästä (tässä yhteydessä) muun muassa eri selainten väliset erot käyttöliittymän esittämisessä.

Lisäksi on tarpeen tarkastella lyhyesti termin toiminnallisuustestaus kahta eri merkitystä, jotka ilmenevät selvemmin englanninkielisissä termeissä "functional testing" ja "functionality testing". Ohjelmistotestaajien sertifiointijärjestelmän (ISTQB) testaussanastossa (ISTQB 2007) on käytetty suomenkielistä termiä "toiminnallisuustestaus" niistä molemmista.

Lisäksi kyseisen sanaston suomenkielinen osuus sisällyttää termiin "functional testing" sekä yksittäisten komponenttien, että myös koko järjestelmän testaamisen siten kuin ne on toiminnallisessa määrittelyssä määritelty. Toisaalta sanaston englanninkielinen osuus ei viittaa toiminnalliseen määrittelyyn ollenkaan, vaan vain sanaan spesifikaatio. ISTQB:n sanastosta on poimittu seuraavaan taulukkoon (Taulukko 6) kyseiset päällekkäisiä merkityksiä sisältävät termit.

Taulukko 6. ISTQB:n testaussanastoa

Termi englanniksi	Termi suomeksi	Kuvausteksti englanniksi	Kuvausteksti suomeksi
Functional requirement	Toiminnallinen vaatimus	A requirement that specifies a function that a component or system must perform. [IEEE 610]	Vaatimus, joka määrittelee toiminnon, joka komponentin tai järjestelmän pitää suorittaa.
Functional testing	Toiminnallisuustestaus	Testing based on an analysis of the specification of the functionality of a component or system.	Komponentit tai järjestelmän toiminnallisuusmäärittelyihin pohjautuva testaus.
Functionality	Toiminnallisuus	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. [ISO 9126]	Kuinka hyvin ohjelmistotuote pystyy tuottamaan toiminnot, jotka täyttävät määrättyjen käyttöolosuhteiden edellyttämät tarpeet.
Functionality testing	Toiminnallisuustestaus	The process of testing to determine the functionality of a software product.	Ohjelmistotuotteen toiminnallisuutta mittaava testausprosessi.

Jos määriteltäisiin niin, että toiminnallisuustestaus viittaisi toiminnallisuusmäärit-

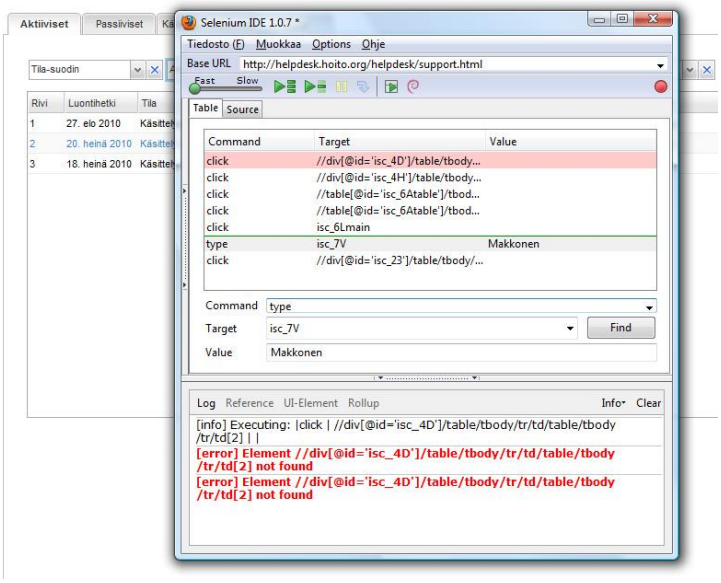
telydokumentissa määriteltyihin toiminnallisuuksiin, voitaisiin käyttäjän syöttölaitteilla (hiiri ja näppäimistö) suorittamien toimintojen nauhoittamisen katsoa olevan osa toiminnallisuustestausta, jos käyttäjä suorittaa tuolloin vaatimusmäärittelydokumentissa määritellyn toiminnallisuuden testaamisen määritellyn toiminnon suorittamalla. Muutoin kyse on enemmänkin käyttöliittymän itsensä testaamisesta tai toimivuuden testaamisesta.

Käyttöliittymän testaamiseenkin on useita lähestymistapoja. Osa testausohjelmista lähestyy asiaa käyttöliittymässä suoritettujen toimintojen nauhoittamisella aloittaen ja siitä nauhoitetta mahdollisesti eteenpäin jalostaen. Osassa testiohjelmia nauhoitteen voi viedä ohjelmasta ulos (eng. export) Java-ohjelmointikieliseksi, jolloin päästään ajonaikaisesti käsiksi käyttöliittymän komponentteihin. Nauhoitteen voi tuolloin käynnistää kyseisestä ohjelmointikoodista, mahdollisesti jopa yksikkötestausta mukaan lisäten.

Periaatteessa hyvänä vaihtoehtona toiminnallisuuden/toimivuuden testaamisessa olisi Selenium, joka on tarkoitettu erityisesti verkkosovellusten testaamiseen. Selenium koostuu itse asiassa useasta eri projektista, joista yhtenä soveltuvana osana on Selenium IDE, jolla voisi ensin nauhoittaa testisarjan (klikkaukset, ym.), minkä jälkeen käytettäisiin siihen sisältyvää "Export Test Case"-toimintoa luomaan testisarjasta Java-ohjelmointikielinen versio:

```
public class SeleniumTest extends SeleneseTestCase {
  public void testSeleniumtest() throws Exception {
        selenium.click("isc_16");
        selenium.type("isc_1T", "040573636");
```

Käytännössä tämä ei kuitenkaan onnistu, ainakaan ilman nauhoitetun suorituksen korjailua, sillä Selenium IDE ei onnistu poimimaan juuri mitään (esimerkkinä kuva 43), mitä tukipyyntöjenkäsittelyjärjestelmän käyttöliittymässä tehtiin nauhoituksen aikana.



Kuva 43. Selenium IDE:n yritys hiiritoimintojen sarjan tallentamiseksi SmartGWT-pohjaisessa sovelluksessa

Joidenkin nappien painallukset Selenium IDE havaitsee, sekä lomakkeiden tekstikenttiin kirjoitetut tekstit, mutta esimerkiksi välilehtien korvakkeiden klikkauksia ei ollenkaan. Lisäksi moni niistäkin, mitä se alun perin havaitsi, kirjautuu tallennuslokiin epätäsmällisesti. Tästä seuraa se, että tallennusta ei yleensä voi edes toistaa, koska viitattuja elementtejä ei löydy.

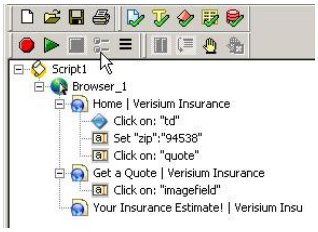
Selenium täyttää vain melkein tietyt kriteerit, joita tällaisella lähestymistavalla toiminnallisuutta testaavalta testiohjelmalta voisi vaatia:

- käyttäjän syöttölaitteilla (hiiri ja näppäimistö) tekemät toiminnot on pystyttävä nauhoittamaan
- jokaista nauhoitteen vaihetta on pystyttävä editoimaan
- ulosviemään nauhoitteen Java-ohjelmointikielisenä, sekä testiohjelman on oltava integroitavissa Eclipse-kehitysympäristöön
- nauhoite on pystyttävä toistamaan alusta loppuun ilman editointia

Viimeistä kohtaa Selenium ei pystynyt toteuttamaan. Edellisiä voi pitää vähimmäisvaatimuksina, mutta paremman hyödyn vastaavanlaisesta testiohjelmasta saisi, jos se lisäksi kykenisi:

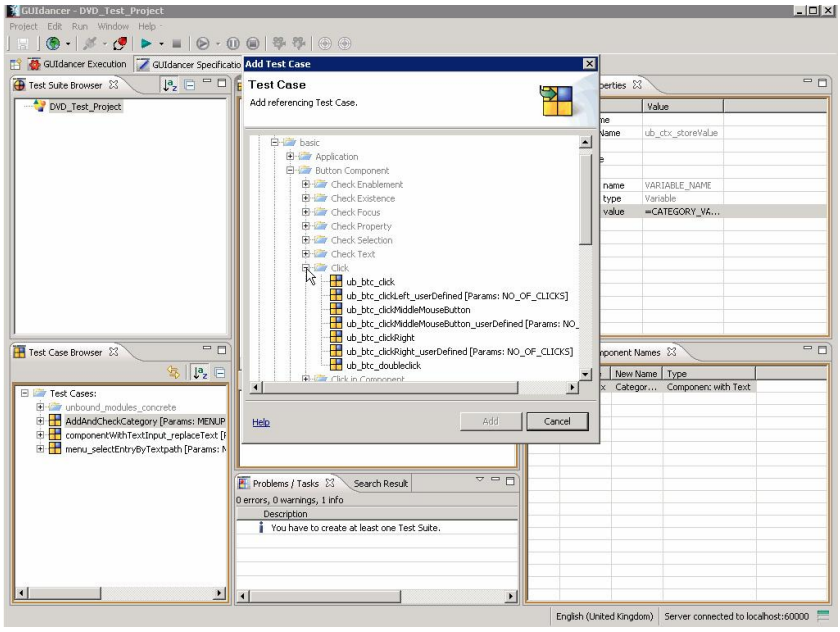
- adaptoitumaan käyttöliittymään tehtäviin muutoksiin (esimerkiksi elementeille annettua id:tä edelliseen nauhoitukseen vertaamalla ja jollakin hienostuneella algoritmilla)

Toinen käyttäjänä tekemiä toimintoja tallentava ohjelma, vTest (Kuva 44), väittää pystyvänsä adaptoitumaan käyttöliittymissä tehtäviin muutoksiin, olematta tukeutumatta objektien nimeen tai id:hen, käyttäen sen sijaan jonkinlaista objektientunnistusalgoritmia. Tämän väitteen todenpitävyys jäi todentamatta, sillä kyseisen ohjelman käyttäminen aiheutti niin suuren kuormituksen testikoneelle, että siitä oli seurauksena koko käyttöjärjestelmän kaatuminen, mihin tavataan viitata myös nimityksellä "Blue Screen of Death" (BSoD).



Kuva 44. vTestin käyttöliittymässä tehty tallennus

Joidenkin testiohjelmien lähtökohtana on testitapausten luominen Eclipse-kehitysympäristössä, ilman tarvetta osat ohjelmoida, mutta siltikin ainakin GUIdancerin (Kuva 45) käyttöönottoa vastustaa korkeahko aloituskynnys.



Kuva 45. GUIdancerin Eclipse-pohjainen kehitysympäristö

Osa testiohjelmista voi olla niinkin pitkälle kehittyneitä, että niissä on mahdollista johtaa testitapaukset ja -joukot järjestelmän käyttötapauksiin ja vaatimusmäärittelyihin asti ja täten asettaa tietyn testijoukon testaamaan täsmälleen jotain tiettyä määriteltyä toiminnallisuutta. Käytännössä käyttötapaus- ja vaatimusmäärittelyt olisi ainakin tässä tapauksessa jo luotu ja tallennettu erillisessä ohjelmassa (UML-kaavioiden suunnitteluohjelmat erityisesti) ja luotu ne testausta varten vielä uudemman kerran testausohjelmassa itsessään (paitsi jos ohjelmat käyttäisivät yhteistä tallennusmuotoa).

Jotkin toiminnallisuuden ja käyttöliittymän testausohjelmat mahdollistavat Eclipse-pluginiensa avulla mukautumisen ohjelmointikoodissa tehtäviin muutoksiin dynaamisesti, jolloin kertaalleen luodut testit ovat uudelleenkäytettävissä ilman niiden uudelleen kirjoittamista tai muokkaamista. Tämä helpottaa erityisesti regressiotestien tekemistä.

5.2.3 Suorituskykytestaus

Ohjelmiston suorituskykytestaus on termi, joka pitää sisällään muutamia erilaisia alalajeja. Kuormitustestausta voi ajatella testinä, josta muut suorituskykytestit (muun muassa stressitesti ja konfiguraatiotesti) on varioitu. Kuormitustestauksessa testattavaa verkkopalvelua kuormitetaan simuloitujen käyttäjien avulla sen normaaliksi määritellyn kuormituksen rajoissa, jolloin saadaan vasteajat ja lukuisia tilastotietoja erilaisille kuormitusasteille. Konfiguraatiotestissä varioitaisiin muuttujia, jotka vaikuttavat palvelun toimivuuteen, kuten sallittujen yhtäaikaisten käyttäjien määrään tai tietokyselyiden ja niiden vastausten välimuistittamiseen.

Stressitesti eroaisi kuormitustestistä siten, että siinä mentäisiin tarkoituksella yli normaalien rajojen (käyttäjämäärät, yhteydenottojen tiheys ja painotettaisiin enemmän prosessointiaikaa vaativia metodeita painottamista). Samalla voitaisiin niin haluttaessa painottaa sen havainnointia, minkälaisia virheilmoituksia palvelu antaa käyttäjälle ja mitä se kirjaa palvelinohjelmistojen lokeihin, sekä yleensäkin koetella sitä, miten palvelu selviytyy ylikuormituksesta.

Tietokanta on tukipyyntöjenkäsittelyjärjestelmän kuormittuvin osa tiedon prosessoinnin tarpeiden osalta. Tavallisten käyttäjien taholta järjestelmää oletetaan

käytettävän erityisesti lukuvuosien alkaessa syksyisin, mutta tuolloinkin yhtäaikaisten yhteyksien määrä asettuisi sadoillakin päivittäisillä käyttäjillä korkeintaan muutamaan lyhytkestoiseen hetkeen. Todennäköisyys sille, että samalle ajanhetkelle osuisi edes kaksi käyttäjää, on pieni. Tavalliset käyttäjät voivat tuottaa järjestelmään lisäkuormitusta silloin, kun he lähettävät tukipyyntöä tai hakevat verkko-osoitteen avulla jonkin tukipyynnön julkisia tietoja.

Monilukuinen käyttäjämäärä (kymmeniä yhtäaikaisia) voisi ehkä manifestoitua siinä tapauksessa, että kaikilla tukipyyntöjen lähettäjillä olisi jokin samantyyppinen tarve, joka saa kaikkien tukipyyntöjen lähettäjien osalta alkunsa muutaman minuutin mittaisen ajanjakson rajoissa. Tällöin tosin saattaisi olla järkevää käsitellä kyseisentyyppinen tilanne informoimalla koko käyttäjäkuntaa jollain yhtenäisellä tavalla.

Tukipyyntöjen käsittelijät, joita on < 10 kpl, eivät käytännössä ehdi pelkästään yksittäisten tukipyyntöjen tietoja käsitellessään käyttämään järjestelmää niin nopeasti, että käyttöintensiteetti siitä merkittävästi kasvaisi. Kuormittavimmat toiminnot ovat tiedonhaku tukipyynnöistä, tukipyyntölistojen muodostus ja raportointi.

SmartGWT-pohjaisen tukipyyntöjenkäsittelyjärjestelmän tapauksessa on käytännöllistä luoda Java Servlet, jota voidaan kutsua suoraan verkko-osoite antaen, parametrien kera. Seuraava servletin ohjelmointikoodi on osa pientä Eclipse-projektia (Dynamic Web Project), jonka kutsuihin vastaava osa suorittaa annetun parametrin perusteella yhden lähes identtisistä metodeista, joita varsinaisen tukipyyntöjenkäsittelyjärjestelmän etäkutsutkin käyttäisivät. Metodeista on karsittu pois autentikointi- ja metodeille itselleen syötettyjen parametrien tarkistukset.

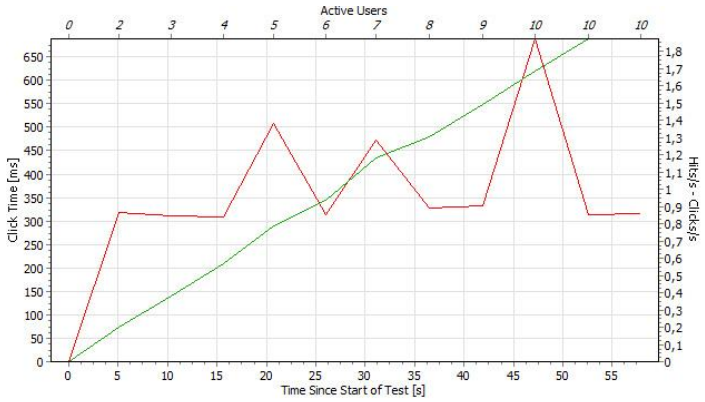
Kutsuttavan servletin nimi on CommandController. Ohjelmointikoodin tasolla se on tavallinen Java-luokka, joka on periyttänyt luokan HttpServlet. HTTP-protokolla GETin kutsumetodeista huolehtii servletin metodi *onGet*. Luokkaan Fetcher on sisällytetty muutamia varsinaisista metodeista testausta varten. Lisäksi on yksi metodi, joka luo uusia tukipyyntöjä. Seuraavat osoitteet ovat toimineet testiosoitteina:

- http://helpdesk.hoito.org/loadtest/CommandController?fetchtype=multisearch
- http://helpdesk.hoito.org/loadtest/CommandController?fetchtype=rowcount
- http://helpdesk.hoito.org/loadtest/CommandController?fetchtype=rows

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

    Fetcher scc = new Fetcher();
    String param = request.getParameter("fetchtype");
    PrintWriter out = response.getWriter();

    if (param != null) {

      if (param.equals("multisearch")) {
          HashMap<String, String> multisearchHashMap = new HashMap<String, String>();
          multisearchHashMap.put("description", "aristotele");
          List<SupportRequestFull> result = scc
                                  .fetchByMultisearch(multisearchHashMap);
          for (SupportRequestFull supportRequestFull : result) {
                  out.print(supportRequestFull.getCode() + ", ");
          }
      } else if (param.equals("rowcount")) {
          HashMap<String, String> filtersHashMap = new HashMap<String, String>();
          Integer result = scc.fetchRowCount(filtersHashMap, "active", false);
          out.print(result);

      } else if (param.equals("rows")) {
          HashMap<String, String> filtersHashMap = new HashMap<String, String>();
          List<SupportRequestFull> result = scc.fetchRows(0, 100, filtersHashMap, "active",
false);
          for (SupportRequestFull supportRequestFull : result) {
                  out.print(supportRequestFull.getCode() + ", ");
          }
      }
```

Autenttista kuormituksen mallinnusta ei vielä yhden metodin kuormittamisella muodostu, mutta tiettyä vaivaa näkemällä sitä voi päästä hyvin lähelle, sillä osa testiohjelmista mahdollistaa "persoonallisuuksien" luomisen simuloiduille käyttäjille. Käytännössä se tarkoittaa useiden eri metodien hyödyntämistä sarjoina, viiveiden kera ja tarkoin ajoituksin. Tässä yhteydessä kokeilluissa testeissä testiohjelma luo halutun määrän simuloituja käyttäjiä, jotka tietyin väliajoin lataavat yhtä testiosoitteista – joko yhtäaikaisesti tai pienin ajoitusvariaatioin.

Kuormitustestaus on testauksen laji, jossa erityisen oleellisen tilastotiedon saa vähäisemmilläkin ominaisuuksilla varustetulla ohjelmalla. Web Stress Tool on eräs tällainen. Kuvat 46 ja 47 on tuotettu käyttäen kyseisen ohjelman trial-versiota, jossa rajoitteena ovat maksimissaan 10 yhtäaikaista käyttäjää ja vähintään 5 sekunnin tauot sivujen lataamisten välillä.



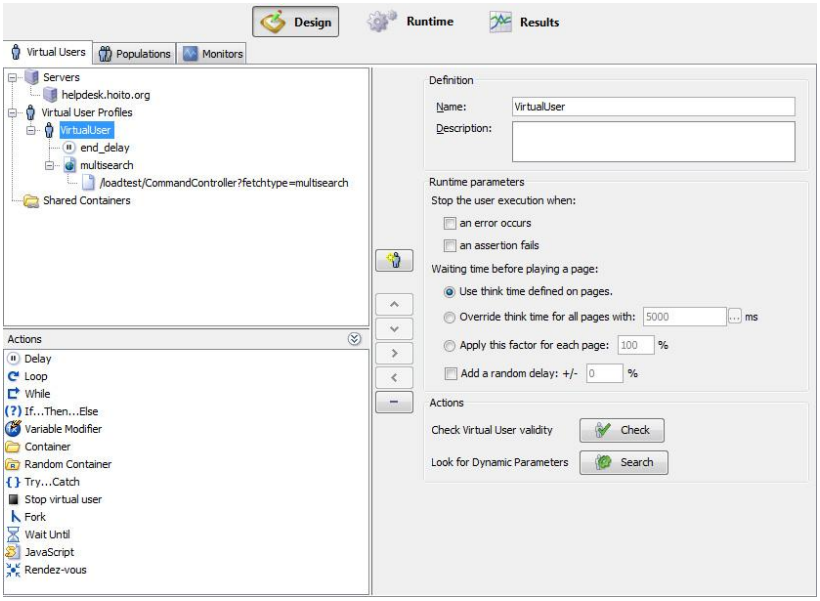
Kuva 46. Web Stress Tooling tuottamaa tilastoaineistoa: kuormituksen kasvu suhteessa vasteaikoihin

Jos taukoa sivulatausten välillä olisi ollut huomattavasti vähemmän, olisivat tulokset olleet toisenlaisia, mutta toisaalta siinä käyttökohteessa, jossa tukipyyntöjenkäsittelyjärjestelmää on aiottu käyttää, ei todennäköisesti koskaan ilmenisi tätä testiä suurempaa kuormitusta.

User No.	Clicks	Hits	Errors	Avg. Click Time [ms]	Bytes	kbit/s	Cookies
1	12	11	0	402	10 065	18,20	
2	11	11	0	388	10 065	18,84	
3	11	10	0	351	9 150	20,83	
4	10	9	0	345	8 235	21,25	
5	9	8	0	409	7 320	17,92	
6	8	7	0	389	6 405	18,81	
7	7	6	0	434	5 490	16,86	
8	6	5	0	402	4 575	18,22	
9	5	4	0	419	3 660	17,49	
10	4	3	0	311	2 745	23,52	

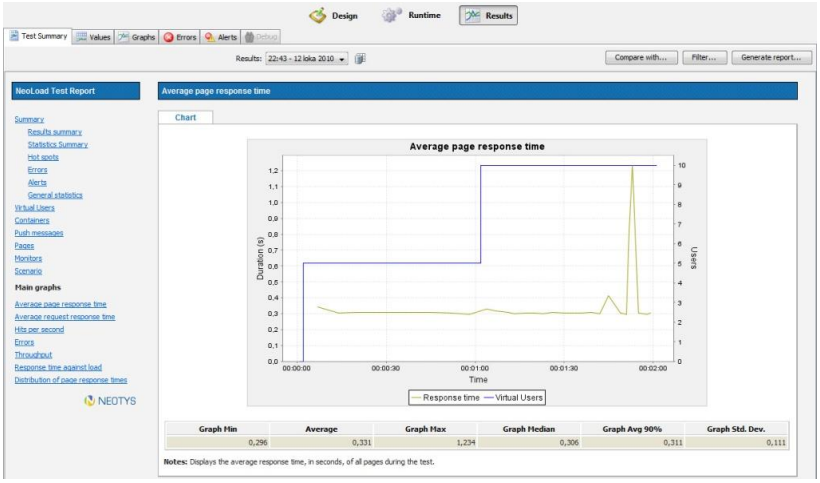
Kuva 47. Web Stress Tooling tuottamaa tilastoaineistoa: keskimääräinen vasteaika oli 300 – 450 ms

Neoload on erittäin monipuolinen kuormitustestausohjelma (Kuva 48), mikä näkyy myös sen hinnoittelussa, mutta sen lukuisista ominaisuuksista huolimatta siihen tutustumiskynnys on matala. Alkuun pääsee, kun a) luo yhden virtuaalisen käyttäjän, b) määrittää populaation, jossa sitä käytetään, c) määrittää kohdeosoitteet ja d) valitsee testiskenaarion. Tässäkin trial-versiossa on omat rajoituksensa, joten sitä kokeiltiin niissä rajoissa kuin oli mahdollista.



Kuva 48. Näyte NeoLoadin käyttöliittymästä

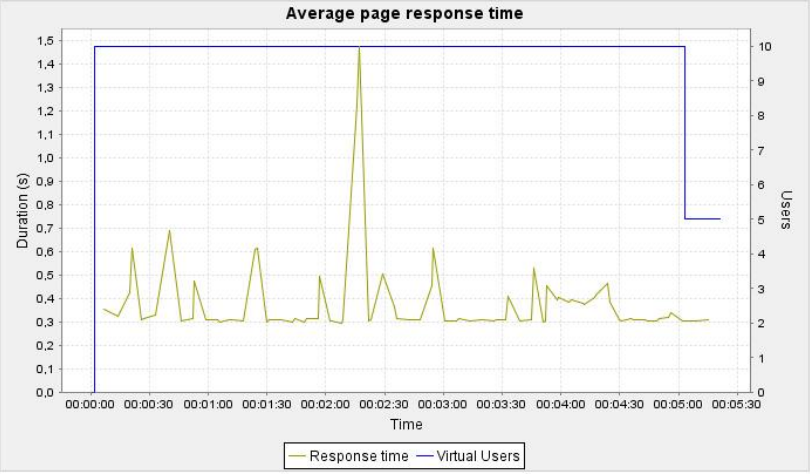
Ensinnä kokeiltiin miten vasteaikoihin vaikuttaa, jos kuormitus pysyy kahden minuutin ajan ensin viiden ja sitten kymmenen yhtäaikaisen käyttäjässä määrässä. Kuvasta 49 on havaittavissa, että viidellä yhtäaikaisella käyttäjällä vasteajat pysyivät varsin vakaana, mutta vasteajoissa ilmeni piikikkyyttä korkeamman kuormitusasteen loppupuolella.



Kuva 49. NeoLoadin tuottamaa tilastoaineistoa: kuormituksen aste suhteessa vasteaikoihin

Seuraavaksi kokeiltiin muuttaa testiä siten, että aloitettiin suoraan 10 yhtäaikaisella käyttäjällä ja kasvatettiin testin kestoa 5 minuuttiin. Tällöin saatettiin huomata kuvan 50 mukaisesti, että ajoittain ilmeni tiettyjä piikkejä vasteajoissa,

mutta edelleen pysyttiin alle sekunnin aikaluokassa, mikä on suorituksena täysin kelvollinen.



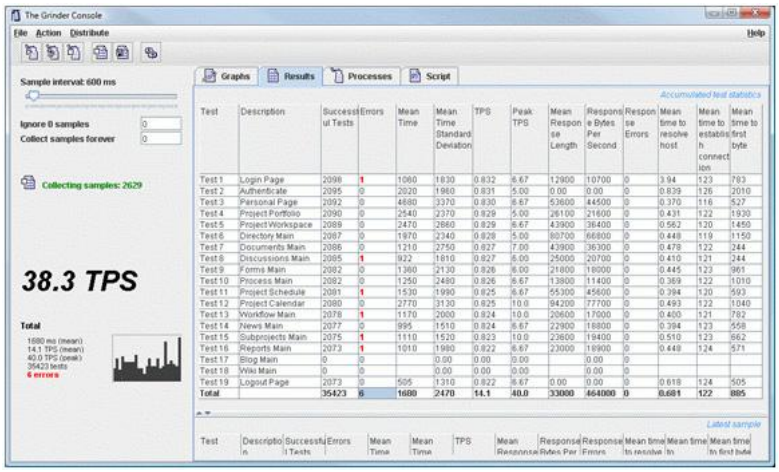
Kuva 50. NeoLoadin tuottamaa tilastoaineistoa: kuormituksen aste suhteessa vasteaikoihin

Työläämpinä, mutta mahdollisesti palkitsevimpina vaihtoehtoina ovat esimerkiksi Xceptance LoadTestin tai Grinderin käyttö, mutta silloin on varauduttava käyttämään ohjelmointikieliä Java tai Jython (Pythonin Java-implementaatio). Seuraavassa ote Python-ohjelmointikielisesti testiskriptistä.

```
from net.grinder.plugin.http import HTTPPluginControl
from HTTPClient import AuthorizationInfo

# Enable HTTPClient's authorisation module.
HTTPPluginControl.getConnectionDefaults().useAuthorizationModule = 1

test1 = Test(1, "Request resource")
request1 = test1.wrap(HTTPRequest())
class TestRunner:
    def __call__(self):
        threadContextObject = HTTPPluginControl.getThreadHTTPClientContext()
        # Set the authorisation details for this worker thread.
        AuthorizationInfo.addDigestAuthorization(
            "www.my.com", 80, "myrealm", "myuserid", "mypw", threadContextObject)
        result = request1.GET('http://www.my.com/resource')
```

Grinder tuottaa vain tekstipohjaista logitietoa, mutta sitä voi tarkastella Grinderin Consolen (Kuva 51) avulla, viedä lokitiedosto taulukkolaskentaohjelmaan tar-

kempaan analyysiin tai käyttää esimerkiksi Grinder Analyzeriä kerätyn datan esittämiseen visuaalisesti.



Kuva 51. Näyte Grinder Consolen käyttöliittymästä

Xceptance LoadTestia voidaan käyttää myös yksikkö- tai regressiotestien suorittamiseen, sillä se pyrkii simuloimaan selainta mahdollisimman tarkkaan. Testit luodaan käyttäen Java-ohjelmointikieltä, mutta se tukee myös mahdollisuutta JavaScriptin hyödyntämiseen. Alla olevaan esimerkkiin on poimittu kolme eri katkelmaa testistä, jossa tuotetaan tietyn id:n omaavan painonapin painallus ja tarkistetaan sen jälkeen, tapahtuiko oletettu muutos toisessa elementissä.

```
// we need the ajax button
this.ajaxButton = HtmlPageUtils.findSingleHtmlElementByXPath(page, "//button[@id='ajax']");

// now push the button
this.ajaxButton.click();

// now check if we change the content
Assert.assertTrue(this.page.getHtmlElementById("content").getTextContent().contains("foo bar
baz bum"));
```

Kuvassa 52 on näytteenä taulukkomuotoinen raportti, joka on peräisin ohjelman verkkosivuilta (ohjelma tuottaa myös graafisia raportteja).

Request Name	Count					Errors	Runtime [ms]				
	Total	1/s	1/min	1/h*	1/d*	Total	Med.	Mean	Min.	Max.	Dev.
AddComment [200]	+6.60%	+6.28%	+6.27%	+6.27%	+6.27%	0.00%	0.00%	-21.51%	0.00%	-19.66%	-31.57%
ConfirmComment [200]	+6.57%	+6.23%	+6.24%	+6.24%	+6.24%	0.00%	-42.86%	-21.69%	-26.67%	+861.48%	+144.78%
ConfirmComment [404]	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)
ConfirmPublishingDate [200]	+7.14%	+6.83%	+6.81%	+6.81%	+6.81%	0.00%	-50.00%	-14.33%	0.00%	+323.42%	+60.01%
ConfirmPublishingDate [302]	+7.22%	+6.83%	+6.89%	+6.89%	+6.89%	0.00%	-30.00%	-17.08%	-31.25%	+522.67%	+87.67%
GoAddArticle [200]	+7.22%	+6.83%	+6.89%	+6.89%	+6.89%	0.00%	0.00%	-21.45%	0.00%	+7.05%	-27.14%
GoAddComment [200]	+6.56%	+6.23%	+6.24%	+6.24%	+6.24%	0.00%	0.00%	-20.26%	0.00%	-14.20%	-31.08%
Homepage [200]	+11.60%	+11.25%	+11.26%	+11.26%	+11.26%	0.00%	0.00%	-15.46%	0.00%	+7.18%	-5.53%
Login [200]	+7.13%	+7.10%	+6.81%	+6.80%	+6.80%	0.00%	0.00%	-24.30%	0.00%	-24.09%	-16.14%
Login [302]	+7.13%	+7.10%	+6.81%	+6.80%	+6.80%	0.00%	0.00%	-27.90%	0.00%	-7.34%	-22.19%
Logout [200]	+7.13%	+7.10%	+6.81%	+6.80%	+6.80%	0.00%	0.00%	+51.00%	0.00%	+3,094.04%	+1,013.88%
Logout [302]	+7.13%	+7.10%	+6.81%	+6.80%	+6.80%	0.00%	0.00%	-2.87%	0.00%	+143.66%	+35.38%
Paging [200]	+6.61%	+6.28%	+6.29%	+6.29%	+6.29%	0.00%	0.00%	-18.28%	0.00%	-14.75%	-19.36%
PickADate [200]	+215.53%	+215.79%	+214.58%	+214.57%	+214.57%	0.00%	-73.00%	-70.55%	-98.03%	+17.43%	-22.01%
PickATag [200]	+10.89%	+10.61%	+10.56%	+10.55%	+10.55%	0.00%	0.00%	-14.18%	-11.11%	+475.34%	+152.22%
PickATag [302]	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)	(added)
PickAnArticle [200]	+7.30%	+6.97%	+6.98%	+6.98%	+6.98%	0.00%	-25.00%	-16.48%	-22.22%	+223.34%	+46.50%
PublishArticle [200]	+7.22%	+6.83%	+6.89%	+6.89%	+6.89%	0.00%	0.00%	-18.79%	0.00%	-6.73%	-33.63%
ReturnToHomepage [200]	+12.04%	+11.69%	+11.70%	+11.70%	+11.70%	0.00%	-33.33%	+8.19%	0.00%	+1,522.99%	+452.76%
SaveArticle [200]	+7.30%	+6.83%	+6.96%	+6.97%	+6.97%	0.00%	-33.33%	-18.81%	-12.50%	+376.04%	+64.32%
SaveArticle [302]	+7.30%	+6.83%	+6.96%	+6.97%	+6.97%	0.00%	-25.00%	-12.22%	-14.29%	+13.00%	+3.98%
Search [200]	+11.32%	+11.13%	+10.98%	+10.98%	+10.98%	0.00%	-53.33%	-38.55%	0.00%	+730.46%	+65.84%
Search [302]	+300.00%	∞	+294.12%	+298.70%	+298.78%	0.00%	∞	+345.00%	-20.00%	+1,020.00%	∞
ViewArticle [200]	+12.78%	+12.45%	+12.44%	+12.44%	+12.44%	0.00%	-33.33%	-6.65%	0.00%	+196.53%	+64.13%
WriteArticle [200]	+7.30%	+6.83%	+6.96%	+6.97%	+6.97%	0.00%	-33.33%	-25.56%	0.00%	-25.65%	-40.57%

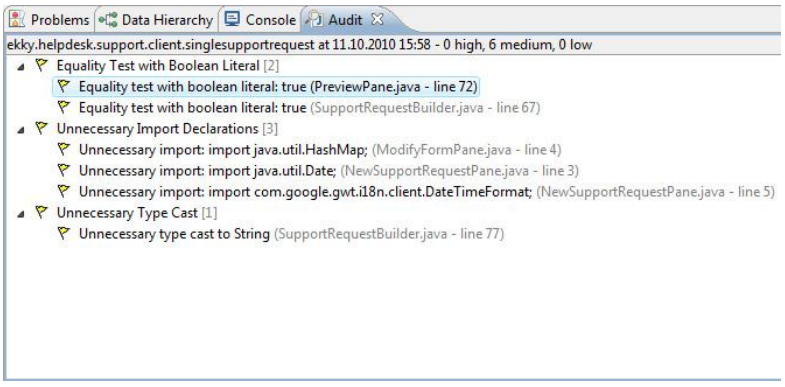
Kuva 52. Näyte Xceptancen tuottamasta raportista

5.2.4 Staattinen analyysi

Staattisia lähdekoodin analysointityökaluja, jotka soveltuvat bugien ja ohjelmointikonventioista poikkeavuuksien automatisoituun etsintään, ovat muun muassa PMD, FindBugs ja CodePro Analytixiin sisältyvä Audit. Nämä ovat kaikki saatavilla Eclipse-plugineina. Java-spesifisistä koodikonventioista kerrotaan yksityiskohtaisesti Oraclen verkkosivuilla. Sivuilla mainitaan perusteluina (Oracle 2009) näiden konventioiden olemassaololle muun muassa se, että niiden käyttö lisää koodin luettavuutta ja se, että ne tekevät koodiin tutustumisen helpommaksi ja nopeammaksi – harvoin mikään laaja ohjelmisto tulee ylläpidetyksi vain yhden ohjelmoijan toimesta.

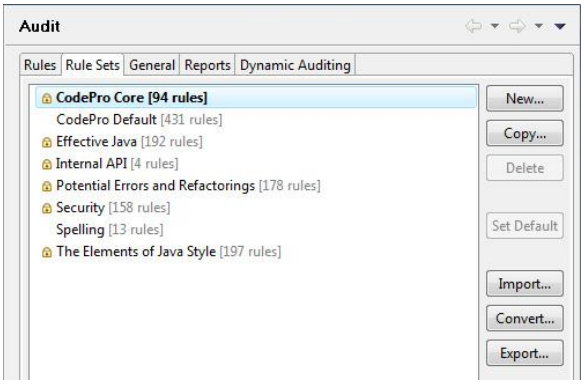
CodePro Analytix - Audit

Ensimmäinen kokeilu Auditilla antoi ymmärtää (Kuva 53), että testatussa paketissa oli vain muutama lähes merkityksetön pieni bugi, kuten tarpeeton tyyppimuunnos.



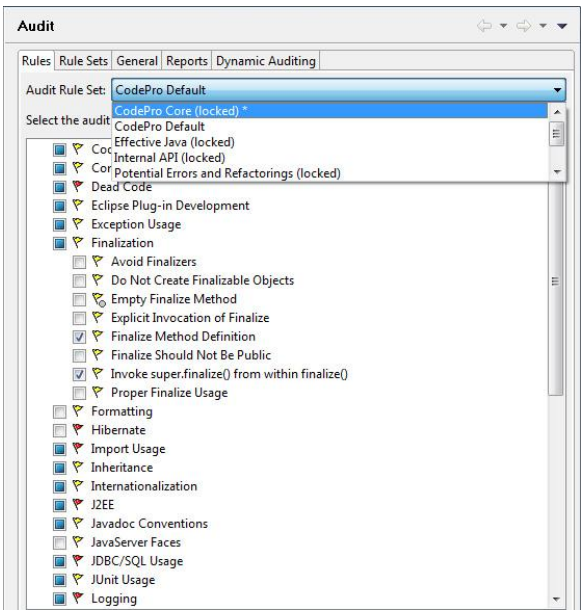
Kuva 53. CodePro Analytixin Audit-toiminnon käyttöä

Tämä johtui kuitenkin vain siitä, että käytössä oli melko minimaalinen tarkistussääntöjen joukko. Vaihtamalla toiseen tarkistussääntöjen joukkoon (Kuva 54) lueteltiin bugejakin paljon enemmän.



Kuva 54. Vaihtoehtoisia sääntöjoukkoja

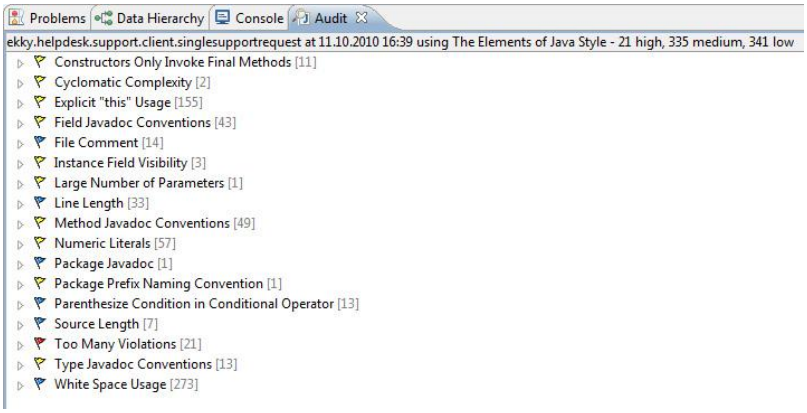
Sääntöjoukko voi vaihtaa, asettamalla Eclipsen asetuksista jonkin muun sääntöjoukon vakioksi tai käynnistämällä testi menuvalinnan "Audit Code Using" kautta. Valinta vaikuttaa siihen, mitä lukuisista erilaisista tarkistuksista (Kuva 55) suoritetaan.



Kuva 55. Eri sääntöjoukot sisältävät erilaisia tarkistuksia

Tällä kertaa, käyttäen The Elements of Java Style -sääntöjoukkoa (Kuva 56), ilmeni paljonkin sellaista tietoa, jonka ohjeistamana olisi hyödyllistä tehdä koo-

diin muutoksia. Moni kyseisellä sääntöjoukolla löydetyistä poikkeavuuksista todellakin on sellaisia, jotka kannattaa korjata.

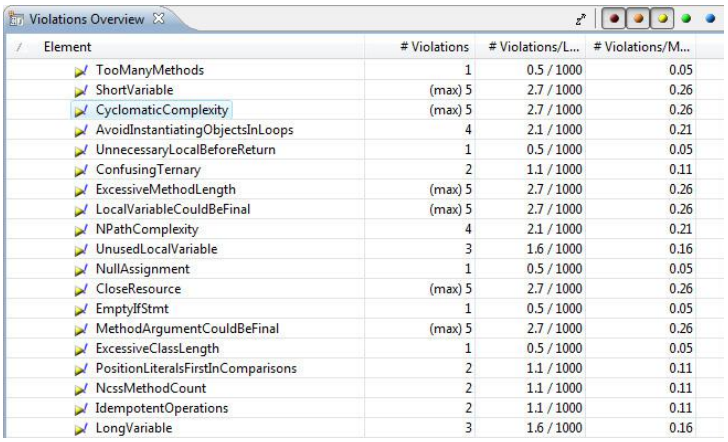


Kuva 56. The Elements of Java Style -sääntöjoukko

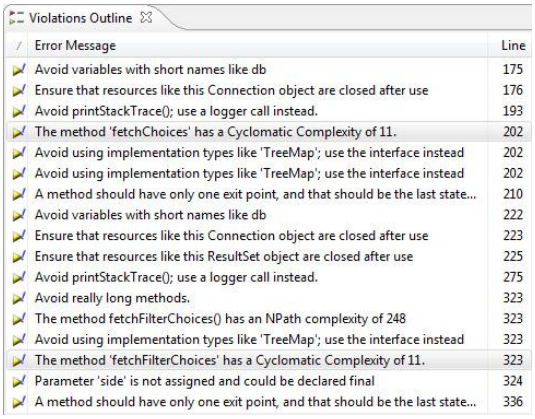
Esimerkiksi sellaista ei välttämättä tule koodatessa ajatelleeksikaan, että koodissa on käytetty sekä välilyöntiä, että tabulaattoria sisennyksien tuottamiseen ("Too Many Violations" ja "White Space Usage" sisältävät pelkästään näitä). Samoin rivien pituuksien ("Line Length") ei kannattaisi antaa kasvaa yli 80-merkkiä leveiksi, koska erityisesti tasaleveällä kirjasimella tulostettaessa siitä seuraa se, että osa koodiriveistä ei mahdu yhden sivunleveyden rajoihin. Sen sijaan avainsanan *this* käytön ("Explicit 'this' Usage") kohdalla moni koodaaja saattaa haluta toimia omien taipumuksien mukaisesti (moni jättää pois). Luonnollisesti Audit merkitsee myös koodiin ne kohdat, joissa havaittuja epäkohtia esiintyy.

PMD

PMD on erittäin helppokäyttöinen ja hyödyllinen työkalu bugien etsimiseen. Tätä voi suositella ensi kertaa staattiseen bugien etsimiseen ryhtyvälle. Tarkastelun laajuudeksi voi valita koko projektin kerralla tai paketti-/luokkakohtaisesti. Kuvat 57 ja 58 esittävät otteen eräistä palvelinpuolen luokasta löytyneistä bugeista. Melko yleisiä vikoja tässä tukipyyntöjenkäsittelyjärjestelmässä ovat final-avainsanojen puuttumiset, mutta ne eivät sinänsä vaaranna mitään. Kompleksisuuden aste sen sijaan on jotain sellaista, joka korkealla ollessaan voi indikoida koodin luettavuuden ja ymmärtämisen olevan hankalaa jopa sen kirjoittajalle itselleen.



Kuva 57. PMD:n havaitsemia konventionaalisia poikkeavuuksia yms. tukipyyntöjenkäsittelyjärjestelmässä



Kuva 58. Vaihtoehtoinen PMD:n esittämä näkemys yhden luokan sisältämistä konventionaalisista poikkeavuuksista yms.

5.2.5 Kattavuustesti

Koodin kattavuustesti on eräs "valkoinen laatikko" -testeistä (eng. white box -testing). Terminä se avautuu hiukan helpommin, jos siitä käytetään nimitystä lasilaatikon, läpinäkyvän laatikon tai rakenteen testaus. Vastakohtana olisi musta laatikko -testaus, jossa testattavan kohteen sisäisestä toiminnasta ei tiedettäisi mitään ennalta, eikä sitä voisi havainnoida suoraan.

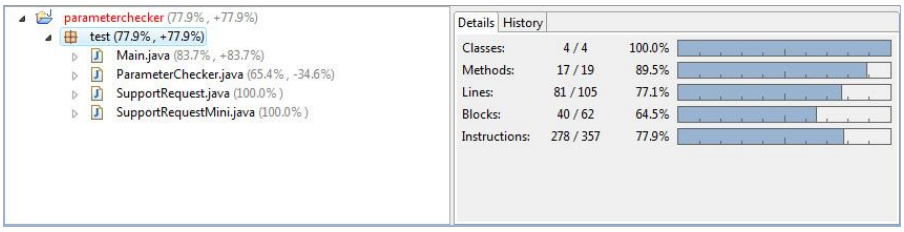
Kattavuustestejäkin on useampaa eri alalajia, joista mainittakoon funktiokattavuus, lausekattavuus ja ehtokattavuus. Kattavuustesti on dynaaminen testi eli testauksen kohteena oleva sovellus tai sen osa ajetaan (vertaa staattiseen testaukseen, jossa sovellusta ei varsinaisesti käytetä).

Moni kattavuustestiohjelmista vaatii testauskohteekseen tavallisen Java-sovelluksen, joka käynnistyy main-metodista, mutta osalla niistä pystyy testaamaan myös servlettejä ja ne osaavat hyödyntää erilaisia tagikirjastoja (Facelets, Java Server Faces, jne.). SmartGWT-pohjaisen tukipyyntöjen käsittelyjärjestelmän tapauksessa oli käytännössä pakko luoda uusi projekti Eclipseen, johon otti mukaan ne luokat, jotka sisältyivät halutunlaisen testin piiriin, minkä lisäksi täytyi luoda vielä yksi main-metodillinen (käynnistävä) luokka. Seuraavassa katkelma koodista, jota käytettiin kattavuustestien ajamisen kohteena:

```
public static void main(String[] args) {

        String code[] = { "TP75784", "TP56730" };
        for (int i = 0; i < code.length; i++) {
          if (ParameterChecker.check_fetchSupportRequestDetails(code[i])) {
              System.out.println("Koodi..ok");
          } else {
              System.out.println("Koodi numerolla " + i + " ..virhe!");
          }
        }
```

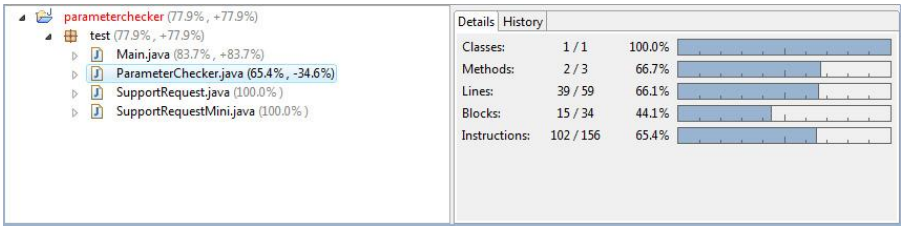
Kattavuustestausvälineitä on monia, kuten muitakin testausvälineitä, eivätkä niiden kehittäjät ole ottaneet tai pystyneet ottamaan huomioon muiden testiohjelmakehittäjien päätöksiä ja valintoja, mistä seuraa se, että tiettyjä Eclipse-plugineita ei pysty käyttämään samanaikaisesti samassa Eclipse-kehitysympäristökokoonpanossa. Tämä oli tyypillisenä ongelmana erityisesti Eclipsen vanhemmissa kehitysversioissa, mutta edelleen ainakaan versiossa 3.5 (koodinimi Galileo) esimerkiksi CodePro Analytixin osana olevaa Code Coveragea ei käytetyssä kokoonpanossa pystynyt hyödyntämään, jos samanaikaisesti oli asennettuna SpryTest.

Helpoimmillaan CodePro Analytixin Code Coveragea pääsee kokeilemaan valitsemalla main-metodin sisältävän luokan ja valitsemalla menusta Run Code Coverage.



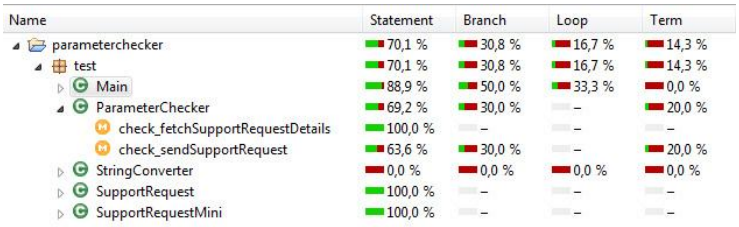
Kuva 59. Code Coveragen tuottamia tilastoja koko kohdepaketin osalta

Kohteena oleva ohjelma ajetaan kertaalleen, minkä aikana Code Coverage monitoroi erilaisia kattavuuksia, luoden lopuksi niiden pohjalta erilaisia tilastollisia esityksiä (Kuva 59 ja Kuva 60). Halutessaan saa myös raportin HTML-muodossa.



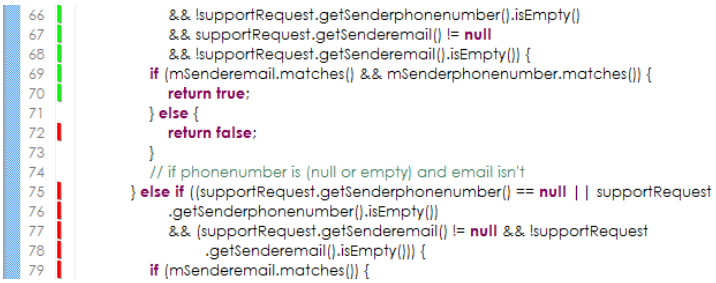
Kuva 60. Code Coveragen tuottamia tilastoja ParameterCheckerin osalta

Vaihtoehtoinen testiohjelma, CodeCover, ilmaisee tilastotiedot hieman eri tavalla, kuten kuvasta 61 voidaan havaita. Monet eroavaisuuksista ovat nyanssitason ja esitystavan eroja.



Kuva 61. CodeCoverin tuottamia tilastoja

Lisäksi kattavuustestausohjelmat merkitsevät varsinaiseen ohjelmakoodiin miltä osin sitä on ajon aikana käsitelty. Code Coveragen tapa ilmaista kattavuuksia on hillityimmillään kuvan 62 mukainen.



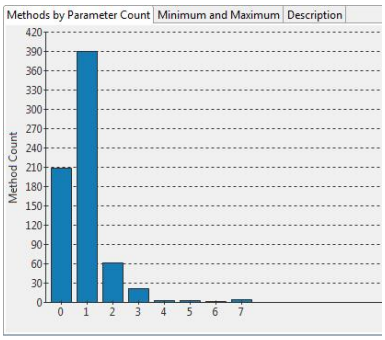
Kuva 62. Code Coveragen ohjelmointikoodiin lisäämiä koodikattavuuksia kuvaavia merkintöjä

5.2.6 Tilastoietoja toteutetusta sovelluksesta

Tilastotietojen vertailua johonkin määriteltyyn tai kuviteltuun ideaaliin voi myös ajatella eräänä soveltuna testinä. Seuraavassa listassa on lueteltu tukipyyntöjenkäsittelyjärjestelmän versio 1.0:n keskimääräiset rivimäärät per metodi, pakettikohtaisesti. Lista on tuotettu CodePro Analytixillä (saatavilla Eclipse-pluginina), joka sisältää useita testaustoimintoja, joista Metrics-toiminto osoittautui hyödylliseksi ohjelmointikoodin staattisessa analyysissä, mikä tässä tapauksessa tarkoittaa tilastotietojen luomista läpikäydyistä Java-luokista. Rivimääristä on havaittavissa, että metodit on pyritty pitämään lyhyinä, jotta niitä olisi helppo käyttää useissa eri yhteyksissä. Palvelinpuolen luokissa tämä ei aivan päde, koska etäkutsuja prosessoivat metodit sisältävät tyypillisesti paljon SQL- ja ehtolauseita.

- ekky.helpdesk.manager.client (10.27)
- ekky.helpdesk.manager.client.accounts (16.72)
- ekky.helpdesk.manager.client.categorypersons (16.33)
- ekky.helpdesk.manager.client.choosables (17.06)
- ekky.helpdesk.manager.client.unitpersons (16.19)
- ekky.helpdesk.server (52.36)
- ekky.helpdesk.shared (3.76)
- ekky.helpdesk.support.client (12.72)
- ekky.helpdesk.support.client.extratools (7.42)
- ekky.helpdesk.support.client.multiplesupportrequests (14.45)
- ekky.helpdesk.support.client.singlesupportrequest (14.14)
- ekky.helpdesk.user.client (21.08)

Metodien parametrimääristä on tarjolla myös pylväsdiagrammi, kuten kuvan 63 mukainen. Joitain asioita on helpompi arvioida, kun tieto esitetään visuaalisesti.



Kuva 63. CodePro Analytics Metrics-toiminnon tuottamaa tilastoaineistoa (metodien parametrien määrä)

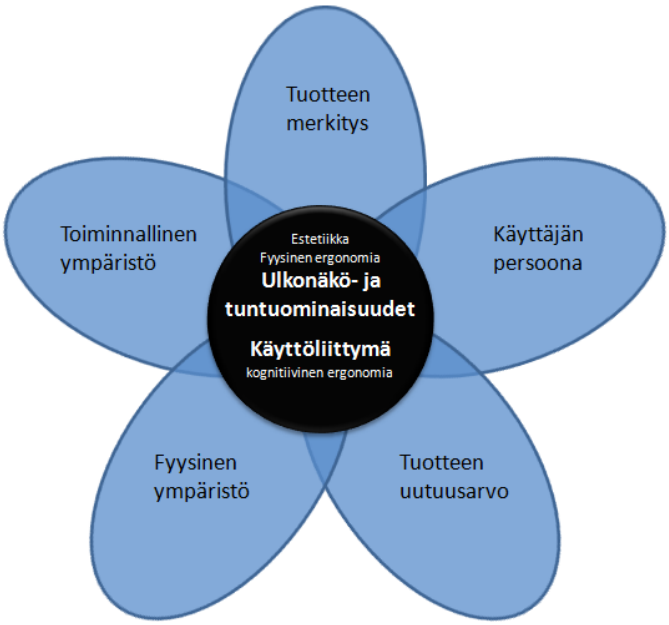
Metrics-toiminnon tuottamat tilastot olisivat saatavilla myös raporttimuodossa (HTML, XML ja tekstimuoto). Valitettavasti leikkaa ja liimaa -toimintoa ei ollut, jotta halutun osan ja vain sen osan saisi helposti liitettyä esimerkiksi tähän tekstiin eli tietyissä tarpeissa vaaditaan manuaalista jälkikäsittelyä.

Koko sovellus sisältää koodirivejä 12396 (kommenttien ja tyhjien rivien kanssa 16440 riviä), joista suurimman osan käyttävät palvelinpuolen luokat (noin kolmasosa). Nämä ovat siis itse tuotettua koodia, eikä niihin lasketa mukaan esimerkiksi käytetyn sovelluskehyksen koodia. Kirjainmerkkien määrä 540438 kuulostaa sellaisenaan melko paljoltakin, vaikkei kyseessä olekaan erityisen kookas sovellus.

5.2.7 Testaaminen käyttäjillä

Tietojärjestelmän, kuten fyysistenkin tuotteiden, toteutuksen onnistuneisuuden määrittelyyn vaikuttaa olennaisesti se, kuinka loppukäyttäjät ottavat tuotteen vastaan. Lähtökohtana voidaan pitää, että tietojärjestelmää ei voida toteuttaa onnistuneesti, jos sitä on testattu pelkästään niin sanotussa laboratorio-olosuhteissa. Tässä sillä viitattaisiin testaamiseen pelkästään yhdessä ympäristössä, joka pahimmillaan voisi olla sama ympäristö, jossa tuotetta kehitettiin.

Sampsa Hyysalo viittaa kirjassaan "Käyttäjätieto ja käyttäjätutkimuksen menetelmät" (2006, 24) käyttökokemuksen jäsentämiseen erilaisina "maailmoina". Kuva 64 havainnollistaa näiden eri maailmojen vaikuttavuutta ja toisiinsa kutoutuneisuutta.



Kuva 64. Käyttökokemuksen eri puolia (pelkistetty versio Hyysalon (2006, 25) kirjassa käytetystä kuvasta)

Tukipyyntöjenkäsittelyjärjestelmän loppukäyttäjinä on kaksi erityyppistä käyttäjäryhmää, joista käyttäjän käyttöliittymän monivaiheista tukipyyntölomaketta voivat käyttää opiskelijat, opettajat ja muu koulun henkilökunta. Tätä käyttöliittymää ei ole esitelty tarkemmin tässä opinnäytetyössä, sillä sen lopullinen toteutus ajoittuu projektin jatko-osaan, muiden lisätoiminnallisuuksien toteuttamisien kanssa. Toinen merkittävä käyttäjäryhmä on tukipyyntöjen käsittelijät, joilla on mahdollisuus käyttää järjestelmää monista muistakin paikoista kuin työhuoneestaan käsin: ulkona kannettavan tietokoneen ja langattoman verkon kautta tai mistä tahansa tietokoneelta, josta sisäverkkoon pääsee.

Käytettävyyteen ja samalla myös mahdollisiin virhetoimintoihin vaikuttavat tietojärjestelmän omien ominaisuuksien lisäksi myös ulkoiset olosuhteet, kuten varjoisuus ja melu. Myös tuotteen merkityksellä testikäyttäjälle on vaikutuksensa, mikä voi ilmetä esimerkiksi siinä, että kaikki testikäyttäjät eivät ole samalla tapaa motivoituneita raportoimaan sen käytön aikana tekemistään omista virheistään. Joillakin ihmisillä käytön yhteydessä tapahtuneista virheistä kertomatta jättäminen voi johtua häpeän tunteesta – ei kehtaa kertoa.

Jos järjestelmää testataan sellaisella, jolla on sopivasti aikaa sen testaamiseen, joka on motivoitunut ja tunnelma tilanteessa on rentoutunut, hän todennäköises-

ti vastaa kaikkiin asiaan liittyviin kysymyksiin mielellään. Varsinkin jos haastateltava kokee, että hänen mielipiteitään arvostetaan ja niillä on merkitystä tuotekehityksen kannalta. Ongelmien ja parannusehdotusten kartoitus on yleisin käyttäjäyhteistyön osa-alue (Hyysalo 2006, 92).

Yksittäinen haastattelukerta ei todennäköisesti riitä antamaan täyttä kuvaa siitä, millaiseksi käyttäjät tuotteen käytön kokevat, sillä käyttäjiltä vie oman aikansa kotouttaa ja "kesyttää" (eng. domesticate, Hyysalo 2006, 34) asioita niin, että ne istuvat heidän aiempiin tapoihinsa toimia. Kattava käyttäjätutkimus voikin viedä useita kuukausia.

Tukipyyntöjenkäsittelyjärjestelmää kehitettäessä onkin syytä ottaa huomioon Blaise Pascalin toteamus (1952, 50):

> *Mitä sitten halutaankin vakuuttaa, tulee ottaa huomioon se henkilö joka halutaan saada vakuuttumaan, tulee tuntea hänen mielensä ja sydämensä: mitkä periaatteet hän hyväksyy, mistä hän pitää, ja katsoa sitten miten kyseinen asia suhtautuu noihin periaatteisiin tai hänen mielinouteisiinsa, kun sen edullisesti esittelee*.

6 PROJEKTISTA YLEENSÄ

Jälkeenpäin on mahdollista spekuloida, että jos työtehtävien delegointi asiakkaan puolella olisi edennyt toisin, olisi projekti saavuttanut tiettyjä etappeja nopeammin. Itse asiassa opinnäytetyön tekijä ilmaisi jo 28.4.2010 kiinnostuksensa tällaista projektia kohtaan, jossa on tarkoituksena suunnitella ja toteuttaa tukipyyntöjenkäsittelyjärjestelmä, mutta kesti lähes kuukauden, ennen kuin asiakas, joka oli ehdotuksen haettavaksi jättänyt, palasi asiaan. Kesäkuussa käydyn toisen palaverin jälkeen opinnäytetyön tekijä itse ei ollut vielä täysin varma, haluaako ottaa tämän opinnäytetyöehdotuksen vastaan. Tähän vaikutti moni yksittäinen avoin kysymys, kuten "pitäisikö sittenkin miettiä vielä jotain muuta opinnäytetyön kohteeksi" ja "kuinka tosissaan asiakas itse on kerrotun tarpeen suhteen". Alaluvussa 2.2 kerrotaan tarkemmin asiakkaan tarpeiden kartoittamisesta ja siinä onnistumisesta.

Voidaan kuitenkin sanoa, että projekti on ollut heinäkuun alusta lähtien käynnissä, vaikka ensimmäisen kerran sopimuksia allekirjoitettiin vasta ensimmäisessä ohjauspalaverissa elokuussa. Kyseinen sopimus on opinnäytetöiden vakiosopimus, jossa kuvaillaan muutamalla sanalla opinnäytetyön aihe ja määritetään päivämäärien avulla tiettyjä etappeja.

6.1 Yhteistoiminnan toimivuudesta ja viestinnästä

Projektin kuluessa opinnäytetyön tekijä koki erityisesti mielipiteiden ja kollektiivisen palautteen saamisen käyttöliittymien käytettävyydestä olevan liian vähäistä. Ajoittain opinnäytetyön tekijälle jäi tuntuma, että tämä on merkityksellisempi hänelle itselleen kuin asiakkaalle, mikä perustui muun muassa siihen havaintoon, että opinnäytetyön tekijä itse oli aina yhteydenottojen aloitteentekijänä. Suora yhteydenotto asiakkaaseen tuotti usein muihin työkiireisiin viitanneen selityksen, mikä sinällään indikoi tarpeesta tehostaa organisaation toimintaa joko tietojärjestelmäpohjaisten ratkaisujen kautta tai henkilöstöresurssien uudelleen arvioimisen kautta (kompetenssit ja kapabiliteetit huomioiden). Tämä puoltaisi alalu-

vussa 2.2 ("Asiakkaan tarpeiden kartoittamisesta ja siinä onnistumisesta") viitattujen Business Intelligence -ratkaisujen kehittämistä.

Projektin etenemisnopeutumisen kannalta olisi voinut olla parempi, jos asiakas olisi pitänyt tiukasti kiinni tietyistä määräajoista, mutta käytännössä ne olivat aina melko suurpiirteisiä kuten "olisi hyvä, jos järjestelmän saisi käyttöön ennen koulujen alkua" tai "mutta ei se haittaa, jos menisi syysloman jälkeenkin, että pääsee koekäyttämään".

Loppukäyttöympäristöä vastaavan palvelimen rakentaminen, jossa järjestelmää voisi testata kesti asiakkaalta siinä määrin kauan, että opinnäytetyön tekijä koki sopivammaksi hankkia omalla kustannuksellaan Java-hosting -palvelun jostain toisaalta, jossa järjestelmää voi demota. Myöhemmin opinnäytetyön teki päätöksen asentaa järjestelmä asiakkaan palvelimelle siinä vaiheessa, kun tekijänoikeussopimuksesta päästäisiin yhteisymmärrykseen. Kulukorvauksien ja opinnäytetyötä varten hankittujen ohjelmien ja muiden tarvikkeiden yhteenlaskettu summa ei ollut taloudellisesti merkittävä, mutta asiakkaat täyttymättömät lupaukset niistä vastaamiseksi vaurioittivat osaltaan asiakas—ratkaisun tarjoaja -suhdetta.

Ohjaajana koulun puolelta on toiminut Martti Ylä-Jussila, joka asetettiin ohjaajaksi vasta 13.8.2010. Käytännössä opinnäytetyötä onkin tehty ilman ohjaajaa (koulun puolelta) koko kesän ajan.

Ohjauspalavereita, joihin osallistui myös ohjaaja koulun puolelta, alettiin järjestää elokuusta alkaen. Niissä käsiteltiin muun muassa projektin kulkua, aikataulutusta, kustannuksia, riskitekijöitä ja käytettyjen työtuntien määriä, sekä tarvittaessa mahdollisia tulevia tai varmuudella toteutuvia suunnanvaihdoksia, joita projektin edetessä saattoi ilmetä. Jokaisen näistä palavereista saattoi katsoa hyödylliseksi. Ne tutustuttivat asiakasta ja järjestelmän toimittajaa toisiinsa, sekä ne olivat välttämättömiä projektin käyntiin saamiselle.

Projektiorganisaation koostumus muuntui projektin aikana siltä osin, että kesälomaltaan palannut, paremmin opinnäytetyön tekijän oman koulutustaustan kanssa yhteensopinut, järjestelmätuessa työskentelevä asiakkaan edustaja,

korvasi toisen samalla osastolle työskennelleen henkilön. Tuossa vaiheessa järjestelmän toiminnallinen määrittely oli kehittynyt jo pitkälle.

6.2 Raportointi ja dokumenttien hallinta

Projektin yhteydessä ei tuotettu säännöllisiä viikkoraportteja, eikä pidetty yllä esimerkiksi verkkopohjaista projektinhallintaa, josta olisi voinut havainnoida tarkasti missä vaiheessa mikäkin projektin osa-alue kulloinkin on meneillään. Opinnäytetyön tekijä pyrki aluksi pitämään ohjaajana toimineen EKKYn IT-koordinaattorin tietoisena työn etenemisestä ja projektin edetessä ilmenevistä huomiota tarvitsevista seikoista, mutta myöhemmin kommunikointi asettui enemmän opinnäytetyön tekijän ja järjestelmätuessa työskennelleen henkilön väliseksi, jolle toiminnallisuuksista keskusteleminen ja yksityiskohdista päättäminen oli delegoitu.

Projektikansio on ollut pelkästään opinnäytetyön tekijän käytettävissä, josta hän tarpeen vaatiessa välitti asiakkaalle dokumentteja tai niiden osia sekä kaavioita – siinä määrin kuin kulloinkin oli tarpeen, toteuttamisen tai asiakohdista keskustelemisen kannalta. Erityyppisiä dokumentteja varten on luotu oma yksilöivä tiedostonsa, jonka eri versiot eroteltiin myös tiedostonnimessä (esimerkiksi *opinnäytetyö - projektisuunnitelma - 0.3.docx*).

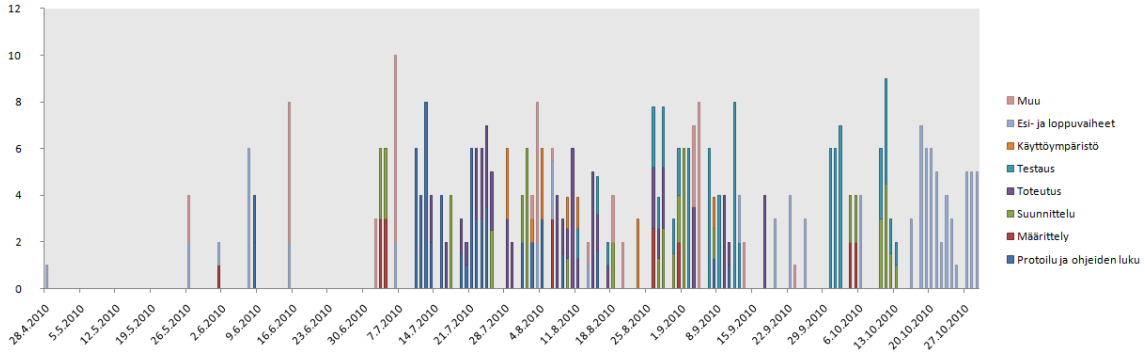
Paljon liitekuvia, taulukoita ja kaavioita sisältävien dokumenttien tapauksessa, luotiin projektikansioon alikansio (esim. *Tekninen määrittely* tai *Testaussuunnitelma*), johon liitetiedostot ja dokumentti eri versioineen sijoitettiin. Tarvittaessa myös alikansiot olisi eroteltu versionumeroilla toisistaan, mutta käytännössä näin ei ollut tarpeen tehdä.

Projektikansion päähakemistosta löytyivät muun muassa esitutkimus-, projektisuunnitelma- ja muut kooltaan pienemmät dokumentit, kuten muistiot ja kokouskutsut. Näihin dokumentteihin liittyvät liitetiedostot kytkettiin isäntätiedostoon nimeämällä ne muuten samankaltaisesti, mutta lisäämällä tiedostoon sana liite ja liitteen yksilöivä nimi (esimerkiksi *opinnäytetyö - projektisuunnitelma - liite - suunnanmuutos.docx*).

Projektikansiosta otettiin päiväyksellä varustettu varmuuskopio ja talletettiin se opinnäytetyötä varten dedikoidulle USB-muistitikulle, sekä eri kiintolevylle kuin millä projektikansio sijaitsi. Samoin tehtiin myös lähdekoodille ja muulle projektin aikana tuotetulle materiaalille. Tämä varmuuskopiointi suoritettiin jokaisen yhtäjaksoisen muutostyön päätteeksi, sekä tarvittaessa useamminkin. Lähdekoodin versionhallinnasta on kerrottu alaluvussa 4.5.1.

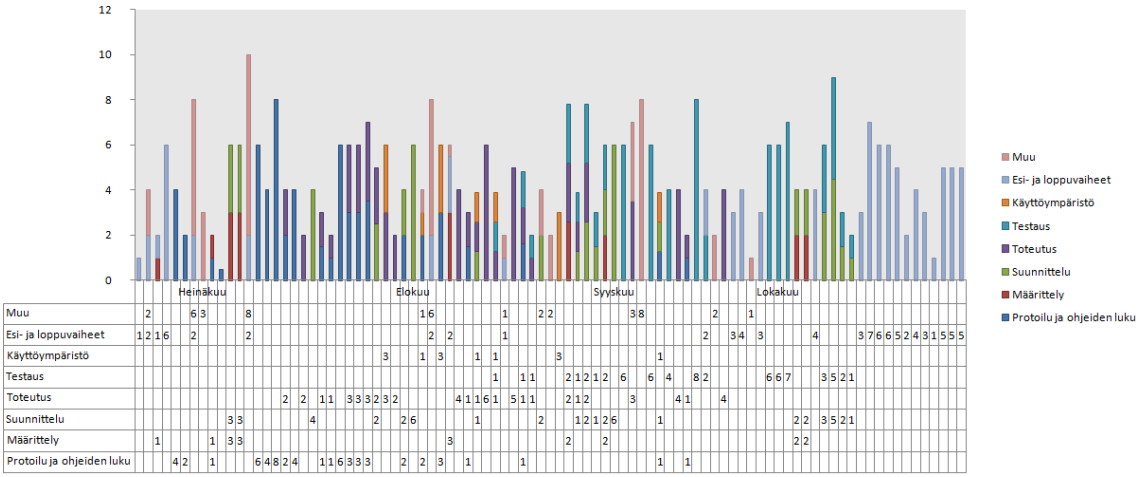
6.3 Tehdyt työtunnit ja projektin vaiheiden sisällöt

Voidaan sanoa, että projekti on ollut käynnissä vasta heinäkuun alusta lähtien, vaikka opinnäytetyön tekijä ilmaisikin jo huhtikuun lopulla kiinnostuksensa tällaista projektia kohtaan, jossa on tarkoituksena suunnitella ja toteuttaa tukipyyntöjenkäsittelyjärjestelmä. Tätä havainnollistaa kuva 66. Pitkät vaaleanpunaiset pylväät viittaavat muutamiin palaverimatkoihin Lappeenrannasta Imatralle, joiden yhteydessä oli mahdollista tutustua myös asiakkaan toimintaympäristöön – ainakin osaan siitä.



Kuva 65. Projektin vaiheiden ajoittumista kuvaava pylväskaavio

Koska projektissa käytettiin sellaista sovelluskehystä, joka ei ollut opinnäytetyön tekijälle aikaisemmin tuttu, sillä tehtiin runsaasti kokeiluja ja prototyypiteltiin sen komponentteja tulevaa järjestelmää varten. Järjestelmän määrittely oli lähes valmiina ennen toteutusvaihetta (Kuva 66), mutta varsinainen tekninen määrittely (vihreä pylväs, suunnittelu) alkoi vakautua vasta syyskuun puolella. Tämä johtui siitä, että teknisistä yksityiskohdista ja pienehköistä käyttöliittymään tehtävistä muutostöistä neuvoteltiin asiakkaan edustajan kanssa melko usein. Nämä muutokset heijastuivat viiveellä myös toiminnalliseen määrittelyyn, jota tarkennettiin syyskuun alussa.



Kuva 66. Projektin työvaiheista tuntien tarkkuudella kertova pylväskaavio arvokenttineen

Osa varsinaisesta testaussuunnitelman mukaisesta testauksesta suoritettiin syyskuun alkupuolella, mutta koska projektiin oli ohjauspalaverissa tehty muutos (ks. alaluku 1.3, "Opinnäytetyön alkuperäinen näkökulma"), joka mahdollisti testaukseen kokonaisuudessaan käytettyjen työtuntien siirtämisen ainakin osittain Ohjelmistotestaus-kurssin osaksi, katsottiin sopivaksi antaa testaamiselle lisäaikaa. Käytännössä tämä tarkoitti testiohjeiden parantelua, testien raportointimallin uudelleenmiettimistä, testityökalujen kokeilua ja testisuunnitelman tarkentamista.

Ne testit, jotka alun perin haluttiin mukaan opinnäytetyön sisällöksi, suoritettiin lopulta kahtena päivänä aivan lokakuun alussa. Niistä testeistä, joista haluttiin kertoa teoriapohjaisesti, mutta kuitenkin toteutettuun järjestelmään kohdistettuina, muodostui ajanjakson 10.10. – 13.10.2010 välisenä aikana useita kymmeniä sivuja pitkä selvitys, joka kattoi lukuisia testityyppejä, esimerkkikoodeineen ja työväline-esittelyineen.

6.4 Työskentely-ympäristössä esiintyneet häiriöt

Harvempi projekti on täysin reaalitodellisuudesta erillään oleva saareke, johon mikään ulkopuolinen tapahtuva ei voisi vaikuttaa. Tämänkin opinnäytetyön kulkuun ovat vaikuttaneet sellaiset tapahtumat, jotka eivät suoranaisesti liittyneet meneillä olleeseen projektiin, vaan ne liittyivät käytetyn käyttöjärjestelmän muo-

dostamaan ohjelmistoympäristöön, joka internettiin kytkeytyneenä oli altis tietoturvaongelmille.

Epävarmuus tietoturva-asioiden todellisesta tilasta uutti tiettyä epämukavuutta työsuoritteiden aikana, sillä on erittäin vaikea todistaa edes itselleen sitä, että koko tietokonelaitteisto voi olla jonkun tuntemattoman etäkäyttäjän ohjailtavissa. Tietoturvapäivitysten säännöllinen asentaminen käyttöjärjestelmään ja erillisiin ohjelmiin ei varsinaisesti takaa sitä, että tietoturvauhkat eivät voisi kohdistua omaan laitteistoon. Tästä on eräänlaisena osoituksena se, että tietoturvapäivityksiä julkaistaan melko usein myös laajalti tunnettuihin ohjelmiin kuten Windows-käyttöjärjestelmiin ja Firefox-selaimeen.

Helpoimmin osoitettavissa olevia tapahtumia ovat olleet projektin aikana lisääntyneessä määrin esiintyneet reitittimen kaatumiset ja käyttöjärjestelmän asennuksen vahingoittumiset sekä oheislaitteiden (tulostin, hiiri ja näppäimistö) poikkeuksellinen toiminta. Poikkeavaa toimintaa on tässä esimerkiksi se, että ruudulla auki olevaan tekstinkäsittelyohjelmaan ilmestyy kirjaimia silloin, kun ei itse koske näppäimistöön; mallinnusohjelmalla tehtyjen kaavioiden sekaan on tallennuksen ja seuraavan avaamiskerran välillä ilmestynyt runsaasti ylimääräisiä kuvioita; tiedostoja on levitelty pitkin levyasemien sisältämiä kansioita. Myös itse käyttöjärjestelmä on aloittanut usein suorittamaan jotain sellaista kovalevyyn kohdistuvaa luku- tai kirjoitustoimenpidettä, jolle ei ole löydettävissä minkäänlaista ennalta määritettyä ajoitusta, eikä sitä pysty yhdistämään mihinkään taustaprosessiin.

Melko helposti osoitettavissa olevia poikkeavia tapahtumia ovat esimerkiksi jonkun tuntemattoman osapuolen suorittama käyttäjätunnuksen luonti johonkin verkkopalveluun – siten, että luotu käyttäjätunnus muistuttaa sellaista, jota opinnäytetyön tekijä yleensä tapaa käyttää ja jonka luomisessa on käytetty tiettyä opinnäytetyön tekijän sähköpostiosoitetta. Opinnäytetyön tekijän itsensä rekisteröitymiin verkkopalveluihin on kohdistunut lukuisia "palauta salasana"-toimintoja, joka samalla paljastaa toiminnon käyttäjälle mihin sähköpostiosoitteeseen jokin tietty tunnus on liitetty.

Käytettyjen verkkopalveluiden toteuttajien tekemistä virheistä johtuvia poikkeavuuksista palveluiden toiminnassa on vaikeampi erottaa niin sanotun sisäpii-

riläisen tekemästä tahallisesta haitanteosta. Esimerkiksi Facebook-profiiliin ilmestyneet verkkoystävät, joita opinnäytetyön tekijä ei itse ole lisännyt, voivat olla päätyneet listoille johtuen yksittäisen Facebook-kehittäjän ohjelmointivirheestä. Tiedostontallennuspalveluja tarjoavan yrityksen verkkopalveluun tallennettujen tiedostojen korruptoituminen voi sekin johtua palveluntarjoajan suorittaman testauksen tai järjestelmän monitoroinnin puutteesta.

Erikoisempia ja samalla vaikeimmin tulkittavia ovat sellaiset tapahtuvat kuin sähköpostin roskapostin määrän aaltoilu paljosta lähes olemattomaan; sähköpostien täydellinen perillepääsemättömyys; lähetettyjen sähköpostien löytyminen entistä useammin vastaanottajan roskakorikansiosta; omien verkkosivujen Google-haulla löydettävyyden kärsiminen (hakutuloksissa ensimmäisenä muille sivuille linkittämättömiä irtosivuja). Selittämättömäksi jäi sekin, miksi ulkomaisessa, Java-hosting -palveluja tarjonneessa yrityksessä alkoi alkuvaiheiden jälkeen esiintyä halukkuutta esittää huonosti englantia puhuvaa tahoa, vaikka oli toiminut sillä kielellä jo useita vuosia – luottokorttiakin veloitettiin tarpeettomasti.

Ohjelmistotestaus-kurssin aineistoon, joka toimi apuna tämän opinnäytetyön testauksenhallinnan ja testaamisen ymmärtämisessä yleensä, ei voinut ulkomaailmassa (esimerkiksi rantakivien äärellä) tutustua rauhassa. Käytännössä tämä johtui siitä, että sellaisia asioita, joista opinnäytetyön tekijä oli implisiittisesti kertonut (verkossa), ettei niistä pidä, ilmestyi tiettyjen ihmisen stereotyyppien muodossa lukupaikan välittömään läheisyyteen. Tyypillistä oli, että juuri ennen ohjauspalavereita ja kesän aikana suoritettuja Ohjelmistotestaus-kurssin kokeita tapahtui aina jotain sellaista, jolla oli potentiaalia häiritä orientoituneisuutta jonkin asian suorittamiseksi. Tyypillistä oli myös se, että näillä poikkeaville ja häiritsevästi ajoittuville tapahtumille oli aina luonnollinen selitys kuten sille, että testikäytössä olleen ulkomaisen testausohjelmiston valmistaja soittaa ja tiedustelee ohjelman käyttökokemuksista täydellistä englantia puhuen.

Tavanomaisempina, projektin aikatauluttamista ja sen vaiheiden suorittamista hankaloittavina asioina olivat opinnäytetyön tekijän työskentelytilojen ympäristössä esiintyneet meluhaitat. Nämä esiintyivät remonttitöiden muodossa. Aikataulutusta hankaloitti se, että taloon, jossa työskentelytilat sijaitsivat, oli alun perin kaavailtu tehtävän kattoremonttia kesä- heinäkuun aikana, mutta se siirtyi

myöhemmin syksyllä tehtäväksi (alkoi 6.9.2010 ja jatkui lokakuun alkuun). Kyseistä remonttia ennakoidessa ja riskiä määriteltäessä oli ennakko-olettamuksena, että melusta voi aiheutua stressiä ja keskittymisvaikeuksia, jotka voivat hidastaa jonkin verran dokumentaation tuottamista, raporttien kirjoittamista, testitapausten suunnittelua, testausta ja toteutusta. Häiriötekijä-aihetta sivuavat eri näkökulmista myös alaluvut 3.1 ("Käyttöliittymien suunnittelun teoriaa") ja 5.2.7 ("Testaaminen käyttäjillä"). Projektin aikataulutus kärsi siitä, ettei ollut varmuutta siitä, milloin remontti alkaa.

6.5 Tekijänoikeussopimuksen allekirjoittamista

Koulun puolelta ei kukaan opettajista, jolta opinnäytetyön tekijä tiedusteli opinnäytetöihin liittyvän tekijänoikeussopimuksen muodosta ja sisällöstä, kokenut itseään sopivaksi henkilöksi määrittelemään sellaista tai osoittamaan sopivia mallipohjia, joten vaihtoehdoksi muodostui sellaisen tuottaminen itse. Kyseisen tekijänoikeussopimuksen luonnos saatiin valmiiksi päivää ennen ohjauspalaverin kokousta (13.9.2010) ja laitettiin sähköpostitse opinnäytetyön ohjaajille tutustuttavaksi. Sitä ei ollut tarkoitus käsitellä kyseisessä kokouksessa yksityiskohtaisesti. Ohjaaja asiakkaan puolelta kommentoi siinä olevan paljon "hyviäkin kohtia", mutta myös paljon sellaista, johon asiakas ei kokenut voivan suostua. Muutosehdotuksia kyseisen sopimuksen muuttamiseksi ei ollut tuotu opinnäytetyön tekijän tietoon marraskuun alkuun mennessäkään, vaikka hän sellaisia kolmeen otteeseen olikin tiedustellut.

Tekijänoikeussopimuksen luonnos sisälsi yksittäisten pykälien lisäksi muun muassa seuraavan alaotsikon alaisen tekstin:

> *Riippumatta siitä, tullaanko opinnäytetyö eli helpdesk-järjestelmä luovuttamaan asiakkaalle kokonaisuudessaan (sisältäen dokumentaation, kaikki toiminnallisen määrittelyn mukaiset vaatimukset toteutettuna, lähdekoodin, sekä muun oleelliseksi katsotun kuten ohjeet lähdekoodin kääntämiseen, ohjeet kehitysympäristön rakentamiseen ja järjestelmän asennusohjeet), vai mahdollisesti joiltain osin karsittuna, on tämän opinnäytetyön tekijän ensisijaisena intressinä tämän sopimuksen hyväksymiselle* maineenhallinta *ja toissijaisena pidättää itsellään oikeudet jatkokehittää järjestelmää kuten parhaaksi katsoo.*

6.6 Ohjelmistojen lisenssit ja niiden yhteensovittaminen

Projektiin, jonka puitteissa tukipyyntöjenkäsittelyjärjestelmä toteutettiin, sisältyi eräs karkea virhe, joka olisi voinut olla kustannuksien ja järjestelmän käyttöönottamisen kannalta erittäin haitallinen, jos projektiin olisi käytetty rahaa ja jos asiakkaalla olisi jonkinlainen ehdoton aikataulu - tai jos asiakas olisi sijoittanut paljon omaa aikaansa. Näin ei kuitenkaan ollut. Kustannuspuoleen asiakas ei osallistunut, aikataulu oli erittäin väljä ja opinnäytetyön tekijän omat intressit painottuivat tietojärjestelmän toteuttamiseen yleensä (pyrkimys oppia uusia tekniikoita), joten lisenssiasioiden tarkempaan miettimisen tarpeeseen havahduttiin vasta hyvin projektin loppupuolella.

Osittain lisenssiasioiden tarkemman miettimisen jääminen myöhäiseen vaiheeseen saattoi vaikuttaa tottuneisuuskin, sillä esimerkiksi MySQL oli aina tuntunut ilmaiselta vaihtoehdolta, josta maksaisivat vain maksullista tuotetukea haluavat, eikä edes JDBC-ajuria itselle ladattaessa tullut ajatelleeksi, että se on GPL v2 -lisenssin alainen. GPL-lisenssin yleisimmin käytetyt versiot v2 ja v3 vaativat molemmat, että niiden alaisia tuotteita käyttävät sovellukset on nekin julkaistava GPL-lisenssillä, mikä tarkoittaa muun muassa sitä, että koko tietojärjestelmän lähdekoodi olisi annettava kokonaisuudessaan sillä, joka on ollut oikeutetta pääsemään siihen käsiksi. Tämän jälkeen hän voisi muokata siitä omansa ja julkaista sen vapaasti, ilman että tekijä voi esittää mitään rajoituksia, sillä varauksella, että myöhempi julkaisija noudattaa GPL-lisenssin ehtoja muun muassa alkuperäisen tekijän mainitsemisesta.

Opinnäytetyön tekijä itse ei hyväksy GPL-lisenssin käyttöä. Hän on tekijänoikeussopimuksen luonnoksessakin pyrkinyt ilmaisemaan, että ei halua tuotteen joutuvan luvattomien tahojen käsiin, mutta GPL mahdollistaa, tietyllä myötävaikutuksella, tälläisen tapahtumisen luvallisesti. Jos tukipyyntöjenkäsittelyjärjestelmä itse olisi GPL-lisenssillinen, niin pitäisi olla erittäin vankka luottamuussuhde asiakkaan ja ratkaisujan toimittajan välillä, jotta ratkaisun toimittaja voisi olla varma, että tuote ei leviä eteenpäin.

Avainsanana GPL-lisensseissä on sana 'levittäminen'. Verkon kautta saavutettavissa olevia palveluita käytettäessä ei tapahdu lisenssin tarkoittamassa mie-

lessä lähdekoodin levittämistä, mutta jos EKKY:n IT-osaston välinen suhde on muotoa asiakas--ratkaisun toimittaja, niin tällöin opinnäytetyön tekijän järjestelmän toteuttajana tulee levittäneeksi tuotetta, mikä aiheuttaa sen, että GPL-lisenssin ehtojen mukaan tuotteen käsiinsä saaneellakin on oikeus tehdä tuotteella melkein mitä vain. Tilanne olisi aivan erilainen, jos kyse olisi palkkatyönä tehdystä tietojärjestelmästä. GPL v3 -lisenssin julkistamisen jälkeen on julkaistu myös AGPL-lisenssi, joka sisältää pykälän, jonka mukaan AGPL-lisenssillä julkaistusta sovelluksesta on pyydettäessä tarjottava lähdekoodin kaikkien sellaistenkin käyttöön, jotka eivät pääse suoraan käsiksi itse lähdekoodiin, mutta pystyvät käyttämään kyseiseen lähdekoodiin pohjautuvaa sovellusta.

Lisenssien yhteensovittaminen on suhteellisen hankalaa, eikä esimerkiksi edes sellainen ole mahdollista, että tukipyyntöjärjestelmän julkaisisi GPL v3 -lisenssin alla, jos yksikin sovelluksen tarvitsemista artifakteista, lisäosista, ohjelmistoalustoista tai tietokannoista on GPL v2 -lisenssin alainen, eikä kyseisen tuotteen lataamisen yhteydessä sanota "GPL v2 tai uudempi". Tämä on erikseen mainittu lisenssiteksteissä.

Vaikka lisenssiehtoja tulkiten saisi MySQL:n käyttöönsä ilmaiseksi ja ilman GPL-lisenssin velvoittamaa lähdekoodin jakamisen velvollisuutta sitä pyydettäessä, ei käytettyä InnoDB-tietokantamoottoria saisi mitenkään ilmaiseksi käyttöönsä ilman, että koko tietojärjestelmä lähdekoodeineen olisi asettava GPL-lisenssin alaiseksi.

Opinnäytetyön tekijä on tehnyt asiakkaalle ehdotuksen PostgreSQL-tietokannan käyttämisestä, joka on BSD-lisenssin (Open Source Initiativen hyväksymä versio) alainen, eikä täten aiheuta vaatimuksia asettaa koko järjestelmää tietyn lisenssin alle, eikä vaadi julkistamaan lähdekoodia. Lähdekoodi sisänsä ei ole niin eksentrinen tai originellisti toteutettu, että sitä sinänsä olisi hyödyllistä suojata, mutta opinnäytetyön tekijä haluaa pitäytyä tälläisessä valinnassa. Päätös tietokantaratkaisun vaihtamisesta ei aiheuta merkittäviä muutoksia ohjelmointikoodiin. Lisäksi asiakas on jo alun perin esittänyt omana vaihtoehtonaan Microsoft SQL Server Expressin käyttöä, jonka Redistribution Rights -lisenssi on myös sopivan salliva.

Käytetty sovelluskehys, SmartGWT, on julkaistu LGPL-lisenssillä, mikä käytännössä tarkoittaa sitä, että sitä voi vapaasti hyödyntää omissa sovelluksissaan ilman velvoitteita asettaa omakin sovellus jonkun tietyn lisenssin alaiseksi. Tämä oli eräs valintaperuste kyseiselle sovelluskehykselle, vaikka mietintä muiden käytettyjen ohjelmistojen lisenssin osalta jäikin myöhempään vaiheeseen. GPL-päätteisten lisenssin yhteensopivuudesta on matriisi osoitteessa: http://www.gnu.org/licenses/gpl-faq.html#AllCompatibility

MySQL:n ja sen tietokanta-ajurin (Connector/J:n) voisi lisensoida myös FOSS-poikkeuksen kautta eli tuolloin voisi hyödyntää ilmaiseksi kyseisiä tuotteita, jos niitä käyttävä tuote itse on lisensoitu jollakin annetun listan mukaisilla avoimen lähdekoodin lisensseillä. Käytännössä tämä tarkoittaisi sitä, että myös FOSS-poikkeus antaisi asiakkaalle täyden vapauden siirtää tuote kenen tahansa muun käyttöön. Lisäksi tietokantamoottori InnoDB:n kanssa ei ole mahdollista käyttää mitään muuta kuin GPL v2 -lisenssiä – tai ostaa kaupallinen lisenssi. InnoDB:tä tarvitaan tietokannan taulujen välisten viiteavainten (foreign keys) käyttöön.

PostgreSQL olisi eräs täysin kelpo vaihtoehto, mahdollistaen tukipyyntöjenkäsittelyjärjestelmän pitämisen suljettuna lähdekoodina, jolle voisi asettaa täsmälleen halutunlaiset rajoitteet – olettaen, että niistä päästäisiin yhteisymmärrykseen asiakkaan kanssa.

Muiden käytettyjen ohjelmistojen ja komponenttien osalta ei ole lisensseihin liittyvää ongelmaa. Näitä muita ovat siis: JavaMail API (vapaasti käytettävissä), Google Web Toolkit (Apache License v2), Java JDK & JRE (voidaan käyttää ilmaiseksi, jos tarkoituksena on "Ohjelman ajaminen"). Javallekin olisi olemassa vaihtoehtoisia virtuaalikoneita ja koko Java-alustan avoimeksi lähdekoodiksi muuntaneita Java-alustan implementaatioita (esimerkiksi OpenJDK), mutta niistä ei ole kokeiltu mitään.

OpenJDK olisi mahdollista lisensoida lisenssillä "GPL v2 + Classpath-poikkeus", jossa Classpath-poikkeus tarkoittaa sitä, että vaikka itse OpenJDK:ta käyttäisi GPL v2 -lisenssiä, ei sitä käyttävää tuotetta tarvitsisi asettaa saman lisenssin alaisuuteen, jos se pelkästään hyödyntää Java SE:n peruskomponenttien tarjoamia palveluita ja metodeita eli ei siis muokkaa itse OpenJDK:tä.

Aiemmin vaihtoehtojen joukossa mukana ollut Glassfish Open Source Edition osoittautuikin sekin tarkemmassa tarkastelussa ongelmalliseksi, sillä sen monilisenssimalli mahdollistaa sen lisensoimisen joko GPL v2 -lisenssillä tai CDDL-lisenssillä – joka ei ole yhteensopiva GPL v2:n kanssa. Sen sijaan JBoss AS Community Edition olisi ollut käypä vaihtoehto, sillä se on LGPL-lisenssinen.

Jos pyrkimyksenä on pitää lähdekoodi niin sanottuna suljettuna koodina, pääsee näissä lisenssiasioissa yleensäkin helpoimmalla, jos valitsee sellaisia ohjelmistoja, jotka on julkaistu joko LGPL-, BSD-, MIT- tai Apache License v2 -lisenssillä.

6.7 Kustannukset

Uuden järjestelmän toteuttamisesta ei muodostunut asiakkaalle muita kustannuksia kuin järjestelmätuen varsinaisesta omasta työstään poissaolevuus opinnäytetyön tekijän kanssa palaveroidessaan tai neuvotellessaan. SmartGWT itsessään on ilmainen asiakkaalle, koska käytössä on sen LGPL-lisenssillinen editio. Sovelluspalvelinvaihtoehdoista ilmaisia olisivat kaikki (Tomcat ja JBoss AS:n Community edition). Käytetyn MySQL tietokannanhallintajärjestelmän voi myös vaihtaa esimerkiksi Microsoft SQL Server Expressiin tai PostgreSQL:lään, jotka ovat kaikki saatavilla ja käytettävissä ilmaiseksi.

Opinnäytetyön tekijän ohjelmistokehitysympäristö oli jo lähes valmiina projektin alkaessa, joten sen rakentamisesta ei aiheutunut kustannuksia muilta osin kuin projektin kuluessa hankitun Visual Paradigm for UML:n päivityksen ja MyEclipsen ostamisen osalta. Jälkimmäinen ostettiin vasta tämän opinnäytetyön kirjoittamisen yhteydessä (aiemmin käytetty trial-versiota tiettyjen kaavioiden tuottamiseen). Opinnäytetyön tekijän omat kustannukset ovat muodostuneet seuraavista:

- Java-hosting (25 eur / kk): 25 eur * 3 = 75 eur
- Visual Paradigm for UML -päivitys: 36 eur
- MyEclipse: 47 eur
- Matkakulut (välillä Lappeenranta–Imatra, 8 tai 11 eur/suunta + ruokailut) = 80 eur

- Safari Books Onlinen 8000 e-kirjan valikoima (20 eur / kk) = 20 eur * 4 = 80 eur
- Tulostimen mustekasetit (n. 15 eur / kpl) = 15 * 3 = 45 eur

Kustannukset yhteensä: 363 euroa.

6.8 Työvälineet

Opinnäytetyö itsessään on ollut kehittämishanke (eräs projektityyppi), jonka hallinnan tukena ovat olleet dokumenttirungot, joita projekteissa tyypillisesti käytetään. Käytännössä projektin on opinnäytetyön rajoissa määritellyt, suunnitellut, testannut, dokumentoinut ja toteuttanut opinnäytetyön tekijä itse. Projektin jatko-osassa projektiin saattaa tulla mukaan muitakin henkilöitä (asiakkaan edustajien lisäksi), todennäköisesti testaukseen liittyen. Toteutuksessa on hyödynnetty erityisesti Eclipse-ohjelmistokehitysympäristön oiminnallisuutta, UML-kaavioiden tuottamisessa pääasiallisena työvälineenä ollen Visual Paradigm for UML. Tietokannan relaatiomalli on toteutettu MySQL WorkBenchillä.

Ohjelmistotuotantoon liittyvillä oppitunneilla on tyypillisesti ollut käytössä Rational Rosen varhaisempi versio, jonka käytänteitä myötäillen on suunnitteluvaiheen malleja pyritty Visual Paradigm for UML:lällä jäljittelemään. Aivan kaikilta tämä ei ollut mahdollista, sillä ohjelmat muodostavat malleista ja niiden sisältämistä artefakteista hiukan erilaisia hierarkiapuita ja Logical Viewin käsite oli aivan erilainen.

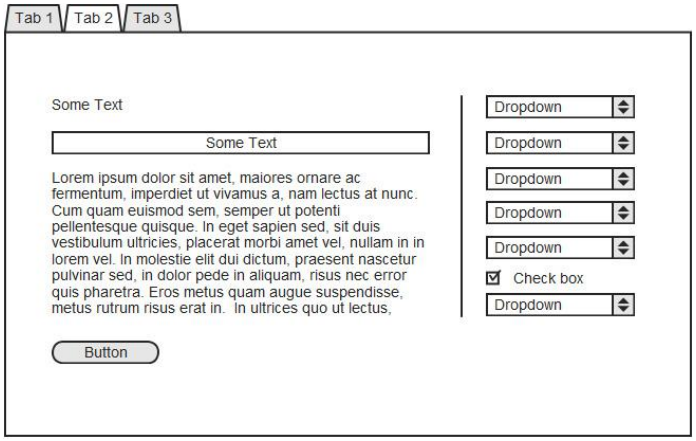
Vapaamuotoisemman esisuunnitteludokumentin lisäksi projektin aikana on tuotettu seuraavat dokumentit: projektisuunnitelma, toiminnallinen määrittely, tekninen määrittely, testaussuunnitelma ja loppuraportti. Dokumenttirunkoina on käytetty pääasiassa Tampereen teknillisen yliopiston Ohjelmistotekniikan laitoksen dokumenttirunkopankkia, sekä saman oppilaitoksen Ohjelmistotuotannon projektityö -kurssin materiaalista löytyviä dokumenttirunkoja. Testaussuunnitelma ja loppuraportti olivat sellaisia, joita opinnäytetyön tekijä ei ollut aiemmin tehnyt.

Monet työvälineistä (erityisesti ohjelmointiin ja kaaviointiin tarkoitetut), joilla tukipyyntöjenkäsittelyjärjestelmä oli aiottu toteuttaa, olivat jo ennestään laajassa

määrin tuttuja, joten työvälineiden käytön erilliseen opetteluun ei ollut tarvetta käyttää aikaa. Järjestelmää suunniteltiin ja kehitettiin opinnäytetyön tekijän omissa työtiloissa. Projektinhallinnassa yleensä tukeuduttiin muun muassa Kai Ruuskan (2007) "Pidä projekti hallinnassa" -kirjaan (silmäilevästi).

SmartGWT:hen tutustuminen vei oman aikansa, eikä sen komponenttien ominaisuuksien monipuolisuudesta johtuen ollut aina helppo kuvitella ennalta mielessään, millainen olisi sellainen käyttöliittymä, johon olisi niin tyytyväinen, ettei sitä katsoisi tarpeelliseksi muuttaa toisenlaiseksi. Käytettävissä ollut SmartGWT-pohjaisten käyttöliittymien visuaaliseen suunnitteluun tarkoitettu Eclipse-plugini (SmartGWT GUI Builder) oli käytössä liian hidas, jotta sitä olisi ollut mielekäs käyttää.

Käytännössä käyttöliittymien näyttöjä suunniteltiin aluksi paperille piirtäen ja prototyypitellen, sekä käyttäen kaaviointityökaluja apuna elementtien sijaintien asettelussa. Tästä kerrotaan tarkemmin alaluvussa 4.4.1 ("Näyttöjen mallintamisesta ja tuottamisesta"). Jälkeenpäin on jäänyt mietityttämään olisiko ollut käytännöllistä luoda "rautalankamallit" käyttöliittymistä esimerkiksi Mockingbirdiä käyttäen. Kuvan 67 esimerkinomaisen käyttöliittymän osan tekemiseen kului aikaa noin 2 minuuttia.



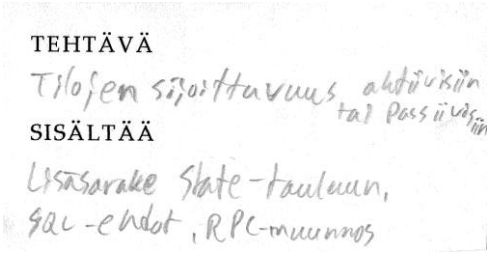
Kuva 67. Esimerkki Mockingbird-palvelun helppokäyttöisyydestä

Erityisesti projektin alkuvaiheilla intuitiolla oli merkittävä rooli tukipyyntöjärjestelmän kehityksen ohjaajana, sillä määrittelyn ja toteutuksen suhteen ei ollut tarpeen pitäytyä erityisen tiukoissa raameissa, vaan oli mahdollista tehdä sellaista mikä sopivalta tuntui. Intuitio-kirjassaan Tony Dunderfelt (2010, 19) hah-

mottelee kahta intuition lajia, joista toinen on "nopea ja kirkas intuitio", joka edustaisi tilanteen näkemistä ja havainnointia intuitiivisesti, toisen ollessa "hitaampi ja tunteen kaltainen intuitio", joka perustuisi alitajuisen tiedon prosessointiin. Voisikin sanoa, että mitä pidemmällä projekti eteni ja mitä selkeämmäksi tehdyt määrittelyt kävivät, sitä selvemmin tuli esille "nopea ja kirkas intuitio".

Hidasta intuitiota vastustavina voimina voi nähdä mielleyhtymäketjujen purkaantumisen, varsinkin niinä hetkinä, joina erilaista mietittävää oli keskeneräisenä erityisen paljon. Tämä ajatusten harhailun uhka on varmasti eräs peruste sille, miksi kannattaa käyttää valmiita malleja esimerkiksi käsitekaavion tekemiseen, joka tehdään järjestelmän tietotarpeiden hahmottamista varten ja josta johdetaan eksakti tietokannan relaatiomalli. Kun kehittyvää järjestelmää oli toteutuksen aikana tarkastellut lukuisista eri näkökulmista, se alkoi muodostua niin tutuksi, että kävi useimmissa tapauksissa erittäin helpoksi hahmottaa miten asiakkaan ehdottama muutos vaikuttaisi ja mitä sen toteuttaminen vaatii.

Intuition, tunteen ja emootion erot ovat hienosyisiä, eikä niille ole helppoa löytää tarkkoja sanallisia ilmaisuja (Dunderfelt 2010, 58), mutta ideoita ja ehdotuksia oli käytännöllistä kirjata muutamalla sanalla ylös TODO-korteille, joista esimerkkinä kuva 68.



Kuva 68. Eräs TODO-kortti (näitä kertyi lähes 100 kpl)

Tällaisten TODO-korttien idea tuli mieleen siinä yhteydessä, kun oli tarpeen kirjata ylös sekä omia, että asiakkaan sähköpostilla ja puhelimitse laittamia ehdotuksia. Ne olisi voinut kirjoittaa johonkin tekstitiedostoonkin, mutta paperilappujen selaaminen käsissä viehätti ajatuksena, sillä niitä oli helppo lajitella ja tyhjiä kortteja oli aina nopeasti käsillä. Aiemmin ajatellun asian muistiinpalauttamiseksi ei tarvittu muuta kuin yksi tai kaksi sanaa otsikon "tehtävä" alle, minkä lisäksi

jokin lyhyt täydentävä selitys otsikon "sisältää" alle. Näitä ei ollut tarkoitettu muiden kuin opinnäytetyön tekijän itsensä käytettäväksi.

Mainittakoon, että ketterän ohjelmointikehityksen projekteille on tyypillistä visualisoida ja jakaa tietoja projektin tilasta pitämällä seinillä "suuria ja näkyviä kaavioita" (Hiranabe 2007). Tätä opinnäytetyöraporttia kirjoitettaessa selvisi sekin, että TODO-korteissa on yksinkertaisuudessaan paljon samaa kuin Kanban-korteissa. Niitä voidaan käyttää sekä henkilökohtaisten asioiden tyypittelemiseen tehtäviksi asioiksi, työn alla oleviksi ja valmiiksi saaduiksi (Personal Kanban 2009), kuten Personal Kanbanissa. Autovalmistaja Toyota on sisällyttänyt omaan tuotantosysteemiinsä pitkälle kehitetyn Kanban System -järjestelmän, jota myös supermarket-järjestelmäksi (Toyota 2010) kutsutaan, sillä kyseisessä järjestelmässä käytetyillä erityisillä Kanban-korteilla osoitetaan mitä osia tarvitaan, mistä ne löydetään, mitä on käytetty ja mitkä ovat lopussa.

Projektin ja testauksen hallintaan ei käytetty esimerkiksi SaaS-pohjaisia (verkossa käytettäviä palveluita, joilla on tyypillisesti pienehkö palvelutason mukaan porrastettu maksu) projektinhallintaohjelmia, mutta demotarkoituksiin käytetyn Daily Razorin verkkopalvelimen kautta asiakkaan oli milloin tahansa mahdollista tarkastella ja käyttää viimeisintä tai lähes viimeisintä versiota. Tämän lisäksi asiakkaan kanssa viestiteltiin paljon sähköpostitse projektin yksityiskohtiin ja toiminnallisuuksiin liittyen. Huolimatta siitä, että ei ollut mitään keskitettyä viestintäjärjestelmää, johon olisi kirjattu ylös esimerkiksi tukipyyntöjenkäsittelyjärjestelmän toiminnallisuuden kehittämiseen liittyvät ehdotukset, ei sanaa "muistaakseni" esiintynyt sähköpostiviesteissä kuin sellaisten asioiden osalta, jotka oli tietoisesti jätetty avoimeksi ja hieman myöhemmin mietittäväksi.

Liitteissä on mukana lista kaikista ohjelmista, verkkopalveluista, lisäosista ym., joita projektin aikana on hyödynnetty. Testausvälineistä on kaikista kerrottu tarkemmin alaluvussa 5.2 ("Soveltuvia testejä ja testityyppejä").

7 LOPPUPÄÄTELMÄT

Tässä opinnäytetyössä on ollut lähtökohtana selvittää, kuinka hyvin SmartGWT-sovelluskehys soveltuu asiakkaan toiminnallisten vaatimusten mukaisen tukipyyntöjärjestelmän toteuttamiseen, ohjelmistotuotannon käytänteitä noudattaen. Jotta tällainen selvitystyö olisi loogisesti mahdollinen, täytyy olla jokin vakaana pysyvä määritelmä, johon sovelluksen kehitysprosessia vertaa. Ilmaisu "ohjelmistotuotannon käytänteet" antaa implisiittisesti ymmärtää, että on olemassa jonkinlaiset perustavanlaatuiset säännökset, standardit, määritykset ja ohjeet, joita ohjelmistotuotannon parissa työskentelevät noudattavat. Käytännössä asia ei ole näin yksiselitteinen.

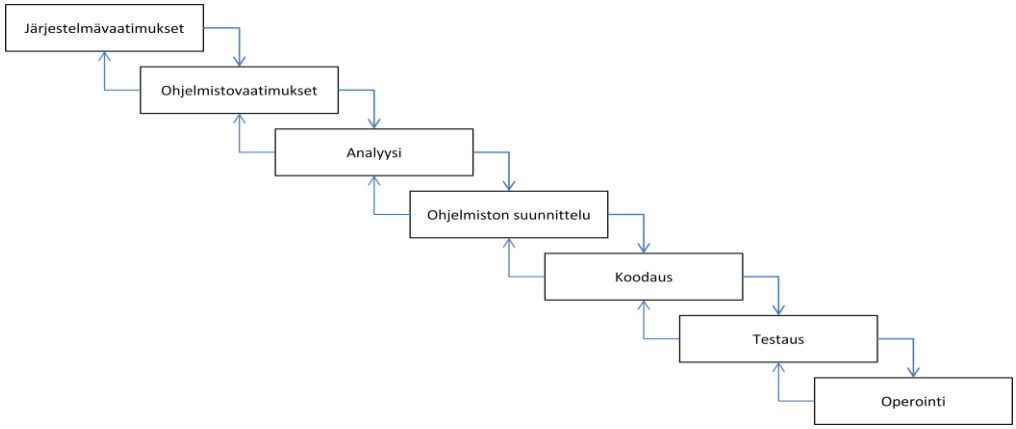
7.1 Toteutus vesiputousmallin muunnoksena

Käytännössä tukipyyntöjenkäsittelyjärjestelmä on toteutettu niin sanotun vesiputousmallin muunnoksena, mutta johtuen kyseiseen ohjelmistotuotannon prosessimalliin liittyvistä vääristä käsityksistä, on tarpeen tarkastella hiukan vesiputousmalli-termin kehityksen vaiheita ja sitä miksi vesiputousmalliin suhtaudutaan pääasiassa negatiivisesti.

Jokainen ohjelmistoalalle kouluttautunut tulee varmuudella tutustuneeksi vesiputousmalliksi nimettyyn vaiheelliseen ohjelmistotuotantoprosessiin. Sitä käytetään usein esimerkkinä huonosta prosessimallista. Väittämää perustellaan usein sillä, että siinä ei voida palata takaisin johonkin prosessin aikaisempaan vaiheeseen, mistä seuraisi se, että se mikä on kertaalleen suunniteltu, olisi pakko viedä toteutukseen sellaisenaan. Kirja "Essential GWT: Building for the Web with Google Web Toolkit 2" (Kereki 2010) on esimerkki lähteestä, jossa tyrmätään "klassinen vesiputousmalli" sillä perusteella, että "prosessi virtaa vesiputousmaisesti tasolta tasolle" (SmartGWT perustuu Google Web Toolkittiin).

Kielikuvana vesiputousmalli on helposti ymmärrettävissä, mutta yksinkertaisimmassa muodossaan se vastaa lähinnä Herbert Beningtonin vuonna 1956 kuvailemaa (Ruparelia 2010) ryöppymallia (eng. cascase model), sekä Winston Roycen (1987) alustavaa mallia, josta hän toteaa, että se on riskialtis ja suorastaan

houkuttelee epäonnistumisia (eng. invites failure). Kumpikaan heistä ei varsinaisesti nimittänyt malliaan vesiputousmalliksi, eikä ole täysin selvää, mistä alkaen termin käyttö alkoi vakiintua.



Kuva 69. Winstonin "vesiputousmallin" eräs kehitysversio, jossa esiintyi jo iterointia edellisen vaiheen kanssa

Kyseinen Winstonin malli oli osa kirjoitusta, jossa hän kehitteli toimivalta tuntuvaa ohjelmistotuotannon mallia vaiheittain. Vaikka mahdollisuus muuttaa ohjelmiston määrittelyä epäonnistuneiden testien perusteella oli Winstonin kaaviossa (Kuva 69) mukana, sisälsi se edelleen riskin, että ohjelmiston kehityskustannukset nousisivat suuriksi, jos huono suunnittelu paljastuisi vasta myöhäisessä vaiheessa. Kuvan mukaisessa muodossaan malli sisälsi idean, että jokaisen vaiheen jälkeen oli iterointia edellisen vaiheen kanssa, mutta ei juurikaan sitä edemmäs.

Seuraavaksi Winston toi esiin tarpeen täydentää kaaviota lisäämällä siihen analyysi- ja ohjelmiston suunnittelu -vaiheita edeltäväksi vaiheen "alustava suunnittelu", jonka suorittaja huolehtii siitä, että analyysivaiheen tekijä "aistii tallennuslaitteitteisiin, ajoituksiin ja toiminnallisuuteen liittyvien rajoituksien" (eng. "the storage, timing and operational constraints") seuraukset. Lisäksi oli tarpeen luoda yleiskuvan muodostava dokumentti, jonka jokaisen toteutukseen osallistuvan osanottajan on ymmärrettävä, sekä vähintään yhdellä henkilöllä täytyi olla erityisen syvä ymmärrys siitä, mitä ollaan tekemässä. Winston korostaa hyvän ja runsaan dokumentaation merkitystä paljon:

- ilman hyvää dokumentaatiota, vain se joka on hoitanut jotain järjestelmän kehityksen (tai testauksen) osa-aluetta, pystyy kyseistä osa-aluetta analysoimaan
- jotta järjestelmän vikoja olisi helppo paikallistaa ja korjata, on dokumentaation oltava selkeää ja informatiivista
- jos dokumentaatiota ei ole, järjestelmän muuntelu pieniltäkin osin, voi olla mahdotonta

On havaittavissa, että kehitellessään malliaan Winsto erkaantuu hyvinkin nopeasti siitä, mihin viitataan puhuttaessa "klassisesta vesiputousmallista". Jo huolellisesti tuotettu dokumentaatio takaa sen, että yksittäinen prosessin vaihe ei ole riippuvainen vain edellisestä vaiheesta.

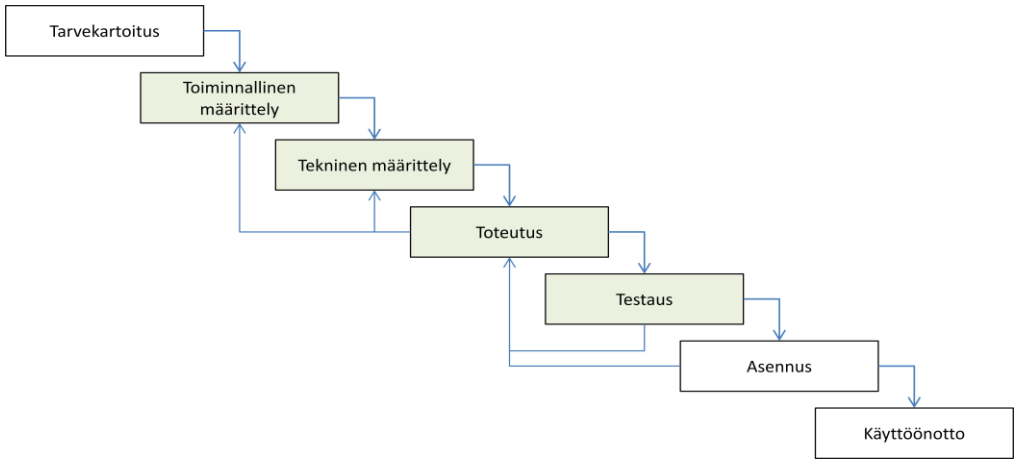
Dokumentteja hänen mallissaan kertyy kuusi kappaletta (määrittelydokumentti, käyttöliittymän kuvausdokumentti, lopullinen spesifikaatio, testisuunnitelma ja käyttöohje). Lisäksi hän kehotti pohtimaan sitä, onko kehitettävä tuote niin ainutlaatuinen, että vaiheet olisi syytä käydä läpi kahteen kertaa: ensimmäisellä kerralla pilottina, käyttäen aikaa vain kolmasosan mitä siihen muutoin arvioitaisiin kuluvan. Tällöin projektin henkilöstöltä vaaditaan kuitenkin erityisen laaja-alaista kompetenssia ja heillä täytyisi olla intuitiivinen ote analyysiin, suunnitteluun ja koodaukseen, sekä heidän olisi kyettävä havainnoimaan potentiaaliset ongelmakohdat nopeasti.

Pilotoinnin tarkoitus oli, yhdessä alustavan suunnittelun ja kattavan dokumentaation avulla, paljastaa ja selvittää ongelmat ennen kuin pidempään kestävän toisen kierroksen testausvaiheeseen päästäisiin. Oli kuitenkin paljolti kehitettävästä järjestelmästä riippuvaista, kuinka laajalti sitä on suunniteltava, dokumentoitava ja analysoitava, ennen kuin voidaan varmuudella todeta, että järjestelmän toteuttaminen asiakkaan antamien rajoituksien puitteissa on mahdotonta.

Per Kroll kuvailee artikkelissaan "Transitioning from waterfall to iterative development" (Kroll 2004) sellaisen prosessimallin, jota tukipyyntöjenkäsittelyjärjestelmäkin on käytännössä myötäillyt: järjestelmä on jo määrittelyvaiheessa ositettu pienempiin melko itsenäisesti kehitettäviin osiin, mutta silti kyse on ollut vain muunnellusta vesiputousmallista.

Kroll suosittelee käytettävän iteratiivisia ohjelmistotuotantoprosesseja, joista valitsee esimerkiksi Rational Unified Processin (RUP), mistä olisikin ollut hyötyä esimerkiksi tilanteiden ennakoitavuuden kannalta, mutta sitä olisi ollut myös haasteellisempi noudattaa. RUP:ssa on tiettyjä ominaispiirteitä, joista osa on yhteisiä muiden iteratiivisten ohjelmistotuotantoprosessien kanssa, kuten aiemmista iteraatiovaiheista oppiminen seuraavien iteraatioiden suunnitelmia luodessa, mutta siinä on myös uniikkeja piirteitä, kuten jatkuva laadunvarmistus, testaaminen ja vaatimusmäärittelyjen täsmentäminen (Kruchten 2004).

Käytännössä tukipyyntöjenkäsittelyjärjestelmän kehitys on siinä mielessä sujunut odotetusti, että vaiheisiin Toiminnallinen määrittely, Tekninen määrittely, Toteutus ja Testaus liittyviä dokumentteja tarkennettiin lukuisia kertoja (pelkistetty prosessimalli kuvassa 70), mikä on lopulta johtanut, aikataulussa lähes pysyen, toivotunlaisen järjestelmän kehittymiseen. Aikataulussa pysymiseen vaikuttivat pitkälti uuteen sovelluskehykseen tutustumisen tarpeesta, sopivien toimintamallien tunnustelusta, testausta aiemmin tekemättömyydestä, mutta osittain myös siitä, että asiakkaan näkemykset muuttuivat toteutusvaiheen edetessä.



Kuva 70. Tukipyyntöjenkäsittelyjärjestelmän on toteutettu vesiputousmallin muunnelmana

Järjestelmän jakaminen kolmeen eri moduuliin ei tehnyt niistä täysin itsenäisiä, toisista moduuleista riippumattomia kehittämisen kohteita, vaan niissä tehdyt muutokset heijastuivat muihinkin moduuleihin, tietokantaan ja yhteisiin dokumentteihin.

7.2 SmartGWT:n ja ohjelmistotuotannon käytänteiden yhteentoimivuus

Ohjelmistotuotannon käytäntöjä voi lähtökohtaisesti ajatella olevan kirjava joukko, joista kaikki eivät ole keskenään yhteentoimivia. Osa käytänteistä saattaa olla lainannut fundamentaalisen ideansa jostain aivan toiselta teollisuuden alalta tai se perustuu johonkin tiettyyn rajattuun tarpeeseen, josta on myöhemmin johdettu pelkistetty versio, jotta kehiteltyä mallia, kaavaa tai prosessia voitaisiin soveltaa laajemminkin.

Winstonin (1987) "vesiputousmalli" on eräs pelkistämisen lähteistä, ITIL:län (Information Technology Infrastrucre Library) ollessa laajasti hyväksytty lähestymistapa IT-palveluiden hallintaan ja toteuttamiseen – se tarjoaa yhtenäisen parhaiden käytäntöjen joukon, jotka on johdettu sekä julkiselta että yksityiseltä sektorilta, kansainvälisesti. Vuonna 2007 julkaistu ITIL:n versio 3 koostuu viidestä peruskirjasta, jotka sisältävät ohjeistuksia ja malleja prosessien määrittelyyn, organisointiin ja käyttöön ihmisten, prosessien ja teknologioiden kannalta (itSMF).

Näiden ITIL-kirjojen tarpeeseen ovat osaltaan vaikuttaneet yhtenäisten käytänteiden määrittämisen tarve, sillä erilaisia käytänteitä on niin paljon, että jo pelkässä niihin tutustumisessa ja niiden keskinäisessä vertailussa kestäisi suhteettoman kauan, mikä korostuisi pienemmissä projekteissa. Siltikään ei ole perusteltua ajatella, että juuri ITIL olisi se ideaali, johon noudatettuja käytänteitä pitäisi verrata, sillä vastaavanlaisia on muitakin (esimerkiksi Application Services Library, ASL; Microsoft Operations Framework, MOF).

Täten voidaankin pitää perusteltuna sitä lähtökohtaa, ettei SmartGWT:n yhteentoimivuutta arvioida vertaamalla suoraan johonkin tiettyyn laajalti käytettiin malliin, kaavaan tai prosessiin, vaan arvioidaan sitä, onko käytettävissä olevista erilaisista käytänteistä saatu koottua koherentti ratkaisu, jota on lisäksi helppo muunnella. Tämä määritelmä pitää sisällään tarkastelun ohjelmointimukavuudelle, testattavuudella, mallinnettavuudella, muiden teknologioiden kanssa yhteentoimivuudelle, tuotetun dokumentaation hyödynnettävyydelle, osioinnin ja aikataulutuksen hallinnalle, kehityksen kulun sulavuudelle ja reaalimaailman vaatimusten huomioivuudelle.

SmartGWT-pohjaisen järjestelmän kehittämisestä on kerrottu runsaasti yksityiskohtia tämän opinnäytetyön muissa luvuissa ja seuraavissa alaluvuissa nostetaankin esille vain kertauksenomaisesti muutamia yksityiskohtia jo kerrotusta ja mainitaan olennaisia asioita muunneltavuuteen liittyen.

7.3 Näkemys SmartGWT:n soveltuvuudesta

Loppupäätelmänä voi todeta SmartGWT:ssä olevan tiettyjä yhdentyyppisiä ongelmia, josta lähes kaikki muut ongelmat ovat seurausta. Kyse on kehitettävän järjestelmän semiautomatisoidusta takaisinmallinnuksesta UML-kaavioiksi. Jos järjestelmästä voisi tarpeen vaatiessa takaisinmallintaa tietyn halutun osan, olisi dokumentaatio helpompi pitää ajan tasalla ja täten tehdä mahdollisesta monijäsenisestä tiimityöskentelystä paljon helpompaa.

Tarkkaan ottaen tämä ei ole varsinaisesti SmartGWT:n itsensä suunnittelullinen vika, vaan kyse on takaisinmallinnusominaisuuden tarjoamien ohjelmien ja Eclipse-pluginien rajoittuneisuudesta. Mainittakoon, että järjestelmää kehitettiin tietyiltä osin prototyypittelyn avulla, jolloin suunnitteluvaiheen mallit muodostuivat vasta sen jälkeen, kun niitä vastaava ohjelmointikoodi oli jo kirjoitettu.

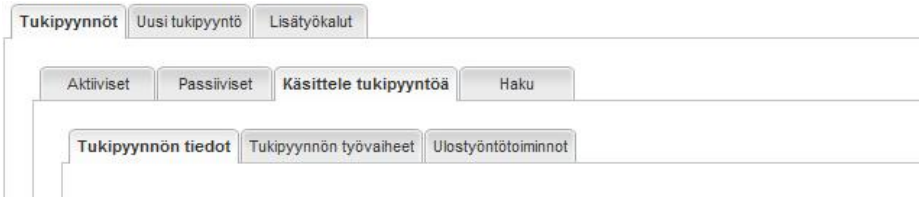
SmartGWT:n versio, jota toteutuksessa käytettiin, oli 2.2 (LGPL-editio). Google Web Toolkitistä, johon se pohjautuu, oli käytössä versio 2.0.3.

Runsas dokumentointi ja kaaviointi ovat olleet merkittävässä roolissa järjestelmän kehityksessä. Ne ovat toimineet tukena kaikissa järjestelmän kehityksen vaiheissa. Dokumenteista ei muodostunut rakenteellisesti SmartGWT-spesifisiä, mutta ilman hyvin jäsenneltyjä dokumentteja SmartGWT-pohjaista tuotekehitystä olisi erittäin vaikeaa toteuttaa tiimissä.

Yksin tehdessä dokumentaation tarve ei ole niin korostuvaa, mutta jos tarkoitus on keskustella toiminnallisesta ja teknisestä määrittelystä asiakkaan kanssa, on hyödyllistä pyrkiä hyvälaatuiseen dokumentaatioon. Johtuen vaaditusta manuaalisesti työstä, joka riittävän hyvien takaisinmallinnusohjelmien puutteessa oli tehtävä, ei dokumenttien päivitystahti voi olla aivan niin nopeaa kuin se parhaimmillaan voisi olla.

7.3.1 Refrakmentointiongelma

SmartGWT ei pakota sellaiseen järjestelmäkehitykseen, jossa pienikin muutos heijastuu heti kaikille muille osa-alueille, mutta tiettyjen helpoiltakin kuulostavien muutoksien tekijän täytyy olla perehtynyt järjestelmään ja sen dokumentaatioon erityisen hyvin, jotta voisi ensinnäkin toteuttaa muutoksen ja sen lisäksi dokumentoida muutoksen eri dokumentteihin. Tällaisesta muuntelun vaikeudesta toimisi esimerkkinä yksinkertainen välilehden siirtäminen tasolta toiselle.



Kuva 71. Välilehtiä käsittelijän käyttöliittymässä

Samalla tasolla olevien välilehtien paikan muuttaminen (kuvan 71 mukainen tilanne) vaatisi vain yhden koodirivin muuttamisen, mutta esimerkiksi "Tukipyynnön työvaiheet"-välilehden siirtäminen ylätasolle, Lisätyökalut-välilehden viereen, nostattaisi tarpeen käsitellä ja muokata useita dokumentteja, kaavioita ja luokkia. Ongelma saa alkunsa siitä, että GWT:hen saatavilla olevaa visuaalista editoria (WindowBuilder Prota) käyttäen ei voi toimia siten, että siirtää välilehteä tasolta toiselle, koska välilehtiryhmät sijaitsevat eri luokkatiedostoissa, eikä editori osaa käsitellä sellaista tilannetta. Olisi erittäin kätevää ja ohjelmistotuotannon käytänteiden kannalta tavoiteltavaa, jos käyttöliittymän muokkaajan ei tarvitsisi kuin siirtää välilehti toiseen paikkaan ja jättää sovelluksen ohjelmointikoodin rakenteen refrakmentointi SmartGWT:n, Google Web Toolkitin ja Eclipse-kehitysympäristön huolehdittavaksi.

Käyttöliittymien prototyypittelyyn SmartGWT-yhteensopiva WindowBuilder Pro sopii hyvin, mutta varsinaisia muutostöitä tehtäessä on välilehden paikan vaihtajan osattava ottaa huomioon se, että välilehdestä tai sen sisällöstä riippuvaiset esityskerroksen komponentit eivät kadota yhteyttä siihen. Tämä riippuvaisuus johtuu osittain siitä miten sovellus on laadittu (katso 4.4.4, "Sovelluksen automatisoidusta takaisinmallintamisesta kaavioiksi"), mutta myös siitä, että kooditasolle luokan instanssia refrakmentoimalla siirrellen, ei Eclipse sellaise-

naan ymmärrä SmartGWT-spesifisiä riippuvaisuuksia (sama ongelma kuin takaisinmallinnuksessa).

7.3.2 Modulaarisuuden ja koherenttiuden saavuttamisen vaikeudesta

Tarkasteltaessa työnkulkua. jossa asiakkaan toiminnallisista vaatimuksista on saatu johdettua käyttökelpoinen ja dokumentoitu sovellus, on siinä ollut nähtävissä kaksi ongelmaa ongelmaa:

- arkkitehtuurin käyttäjäryhmäkohtaisia moduuleja pienempiin osiin jakamisen vaikeus
- analyysivaiheen mallin saaminen vastaamaan suunnitteluvaiheen mallia

Molemmat näistä ongelmista juontuivat osittain siitä, että SmartGWT:stä valittiin käytettäväksi sen LGPL-lisenssinen editio. Tämä ratkaisu teki mahdottomaksi käsitellä analyysivaiheen tietomalleja samassa merkityksessä myös toteutusvaiheessa, koska Hibernatea, joka olisi mahdollistanut olio-relaatio -mallinnuksen (eng. Object-relational mapping, ORM) ei ollut saatavilla LGPL-editioniin (asiakkaan asettamasta kustannusrajoitteesta johtuva valinta). Pro-editiosta alkaen SmartGWT olisi mahdollistanut visuaalisten komponenttien kytkeytymisen suoraan tietokantaan datalähdeominaisuuden avulla eli täten, arvioitaessa SmartGWT:n soveltuvuutta tukipyyntöjenkäsittelyjärjestelmän kehitykseen, on huomioitava SmartGWT:n olevan enemmän kuin sen LGPL-lisenssillinen editio.

Tietty väljyys (eng. gap) ja paikoitellen huono kohdistuvuus tiettyjen analyysi- ja suunnitteluvaiheen mallien välillä (erityisesti luokka- ja sekvenssikaaviot) oli hyväksyttävissä, koska käytännössä analyysivaiheen malleihin ei ollut erityistä tarvetta palata usein ja niitä käytettiin lähinnä keskusteluissa, joiden pohjalta projekti saatiin liikkeelle. Suunnitteluvaiheen UML-kaavioiden osalta voidaan todeta, että lisärahoituksella (Visual Paradigm for UML:n -ohjelman päivitys) olisi ollut mahdollista pitää tehdyt UML-kaaviot synkronissa toteutusvaiheen ohjelmakoodin kanssa automaattisesti – ainakin niiltä osin kuin ne eivät liittyisi SmartGWT-pohjaisen sovelluksen esityskerroksen ja kontrollikerroksen väliseen asynkroniseen kommunikointiin, koska sitä ei käytetty mallinnusohjelma hallitsisi.

7.3.3 Sovelluskehyksen vaikutus aikataulussa pysyvyyteen

Toteutetun tukipyyntöjenkäsittelyjärjestelmän muodostumiseen ei ole vaikuttanut millään erityisellä tavalla se, että kyse on julkisyhteisön tilaamasta järjestelmästä. Noudatetut projekti-, dokumentointi- ja tarvekartoituskäytännöt ovat tuottaneet syötetietoa SmartGWT-pohjaisen sovelluksen toteutukselle, eikä sovelluskehyksen valinnan vuoksi ole tarvinnut tehdä muutoksia näiden projektin alkuvaiheiden käytäntöihin.

On normaalia, että uuteen sovelluskehykseen tutustumiseen kuluu aina jonkin verran aikaa suhteessa koko työmäärään ja SmartGWT:n eduksi onkin todettava, että se on erittäin helposti omaksuttavissa. Ainoat tilanteet toteutusvaiheissa, joissa aikaa kului tarpeettoman kauan, olivat niitä, joissa pyrkimyksenä oli kokeilla jotain erikoisempaa. Tällainen kokeilu saattoi perustua SmartGWT:n komponenttien harvemmin käytettyihin vakio-ominaisuuksiin, mistä johtuen kaikkia niissä esiintybiä bugeja ei välttämättä oltu löydetty. Jos myöhemmin aloittaisi uuden SmartGWT-pohjaisen järjestelmän kehittämisen, olisi aikataulusta helpompi tehdä tarkempi jo aikaisemmassa vaiheessa.

7.3.4 Monipuolista visuaalisesta editoria ei ollut käytettävissä

Jälkeenpäin voidaan arvioida, että visuaalisen editorin käyttö suoran ohjelmointikoodin kirjoittamisen lisänä olisi saattaisi ilmetä siinä, että se vaikuttaa orientoituneisuuteen toteutusvaiheessa: on erilaista kirjoittaa ohjelmointikoodia ja samalla kuvitella mielessään, miltä koodissa luotu näkymä tulee näyttämään, kuin asetella ensin elementit paikoillaan ja sen jälkeen luoda niitä kontrolloiva koodi. Visuaalista editoria käyttäessä voi myös olla helpompaa asettua loppukäyttäjän asemaan. Toisaalta jatkuva ohjelmointikoodin tarkastelu ja visuaalisen editorin käyttämättömyys (tästä lisää alaluvussa 4.4.1, "Näyttöjen mallintamisesta ja tuottamisesta") voi johtaa siihen, että ohjelmoijan ymmärrys järjestelmän toiminnasta muodostuu syvemmäksi.

HTML-koodia ja CSS-tyylejä käsittelevien komponenttien ohjelmointi kooditasolla tuntui usein erittäin kömpelöltä, sillä muutoksia ei voinut havaita ilman sovelluksen uudelleenlataamista selaimessa ja hakeutumalla käyttöliittymässä sinne, missä muutokset olivat havaittavissa. Nämä ovat aina kvalitatiivisia arvioita, ei-

kä niitä olisi helppo mitata ja johtaa niistä yleispäteviä tuloksia. SmartGWT:n maksullisissa editioissa olisi ollut saatavilla SmartGWT:n tuottajatahon itsensä valmistama visuaalinen editori, mutta sitä ei tämän opinnäytetyön puitteissa kokeiltu.

7.3.5 Tuotetukea saatavilla riittävästi

SmartGWT:n käyttäjäyhteisö on varsin aktiivinen sen keskustelufoorumin perusteella, käyttäjien lähettämät julkiset bugiraportit ovat tuoreita, sovelluskehyksen kehittäjät vastaavat SmartGWT:n (maksuttomilla) tukifoorumeilla aktiivisesti käyttäjien kysymyksiin ja foorumien käyttäjien keskinäinen aktiivisuus on runsasta suhteessa tuotteen tunnettuuteen. Sovelluskehyksen valmistajataho on luonut valmiiksi runsaasti esimerkkejä visuaalisten komponenttien käytöstä, mikä madalloitti sovelluskehykseen tutustumista. Komponenttien dokumentointi on ensiluokkaista. Maksullista tuotetukea ja koulutusta olisi myös saatavilla.

7.3.6 Soveltuva pienimuotoisen tietojärjestelmän toteuttamiseen

Tämä sovelluskehys toimii hyvin yhteen erilaisten testausvälineiden kanssa, olettaen että kirjavasta valikoimasta ensin löytää sopivat, sen etuihin kuuluu moniselaintuki (vähäisin bugein) ja sillä tehdyt sovellukset voivat kommunikoida muiden järjestelmien ja tietokantojen kanssa. Valmiin sovelluksen alustavan verkosta lataamisen jälkeen on SmartGWT-pohjaisen sovelluksen käyttäminen nopeaa.

SmartGWT integroituu hyvin Eclipse-kehitysympäristöön, paikallinen debuggaus on ongelmatonta ja yleensäkin SmartGWT-pohjaisen sovelluksen kehittäminen yhdessä apuohjelmien ja liitännäisteknologioiden kanssa on ollut mielekästä. Yksittäisenä osa-alueena, joka toistuvasti vaati järkeilyn avulla selvittämään, mikä on mennyt vikaan, ilmeni tilanteissa, joissa järjestelmää etäpalvelimella kokeiltaessa havaittiin jonkin ongelma, jota ei ilmennyt paikallisesti. Järkeilyssä on se ongelma, että se on paljon työläämpää kuin debuggaus, joka tarjoaa pienellä vaivalla eksaktia ja ehdottaman varmaa tietoa.

Etädebuggausta varten sovelluskehittäjä joutuu kehittämään omat ratkaisunsa tai turvautumaan johonkin etäpalvelimelle asennettavaan lisäkirjastoon, joka

tarjoaa esimerkiksi logipalveluita. Tämä voi muodostua vakavaksi ongelmaksi, jos lopullista käyttöympäristöä ei päästäisi testaamaan myös paikallisesti, sillä se häiritsisi muuten hyvin kulkevaa työnkulun flowta. Tästäkin huolimatta SmartGWT:tä voi suositella käytettävän ainakin pienimuotoisten tietojärjestelmien toteuttamiseen, varsinkin jos on mahdollista käyttää SmartGWT:n kaupallisia editioita ja tietyt refaktorointi- mallintamisongelmat ovat hyväksyttävissä.

LÄHTEET

Kirjat

Dunderfelt, T. 2010. Intuitio – sisäinen viisaus. Jyväskylä: Kirjapaja.

Hovi, A., Hervonen H. & Koistinen H. 2009. Tietovarastot ja business intelligence. Porvoo: WSOY.

Hovi A., Huotari J. & Lahdenmäki T. 2005. Tietokantojen suunnitelu & indeksointi. 2. p. Porvoo: Docendo.

Hyysalo, S. 2006. Käyttäjätieto ja käyttätutkimuksen menetelmä. Helsinki: Edita Publishing Oy.

Jarrett, C. & Gaffney, G. 2009. Forms that Work: Designing Web Forms for Usability. Morgan Kaufmann

Kereki, F. 2010. Essential GWT: Building for the Web with Google Web Toolkit 2. Crawfordsville: Pearson Education Inc.

Mahemoff, M. 2006. Ajax Design Patterns. Sebastopol: O'Reilly.

Pascal, B. 1952. Geometrisestä mielestä. Suomennos Martti Anhava. Juva: WSOY.

Reason, J. 1990. Human Error. New York: Cambridge University Press.

Ruuska, K. 2008. Pidä projekti hallinnassa. Talentum. Helsinki: Gummerus.

Saariluoma, P. 2004. Käyttäjäpsykologia – ihmisen ja koneen vuorovaikutuksen uusi ajattelutapa. Vantaa: WSOY.

Tähtinen, S. 2005. Järjestelmäintegraatio. Jyväskylä: Talentum Media Oy.

Artikkeleita (saatavilla pdf-muodossa)

Bertolino, A. 2007. Software Testing Research: Achievements, Challenges, Dreams. Future of Software Engineering, 2007 FOSE '07 85 – 103. http://dx.doi.org/10.1109/FOSE.2007.25 (Luettu 19.10.2010)

Hiranabe, K. 2007. Visualizing Agile Projects using Kanban Boards. InfoQ. http://www.infoq.com/articles/agile-kanban-boards (Luettu 29.10.2010)

Janzen, D. 2005. Software architecture improvement through test-driven development. Konferenssiteksti. OOPSLA '05. http://doi.acm.org/10.1145/1094855.1094954 (Luettu 19.10.2010)

Kushwaha, D ja Misra, A. 2008. Software test effort estimation. SIGSOFT Software Engineering Notes 33 1 – 5. http://doi.acm.org/10.1145/1360602.1361211 (Luettu 19.10.2010)

Royce, W. 1987. Managing the development of large software systems: concepts and techniques. Proceedings of the 9th international conference on Software Engineering. IEEE Computer Society Press. http://portal.acm.org/citation.cfm?id=41801. (Luettu 29.10.2010)

Ruparelia, N. 2010. Software development lifecycle models. SIGSOFT Software Engineering Notes 35 8 – 13. http://www.deepdyve.com/lp/association-for-computing-machinery/software-development-lifecycle-models-hyg22vLFMC (Luettu 1.11.2010)

Internet-lähteet

Addicott Web. 2009. Clear Instructions Will Improve Your Form's Usability. http://www.addicottweb.com/2009/06/clear-instructions-will-improve-your-forms-usability/ (Luettu 4.9.2010)

Appleseed, J. 2010. Form Field Usability: Matching User Expectations. Baymard Institute. http://baymard.com/blog/form-field-usability-matching-user-expectations (Luettu 4.9.2010)

Beust C. 2004. JUnit pain. http://beust.com/weblog/2004/02/08/junit-pain/ (Luettu 4.9.2010)

Chui, T. 2009. Web form design guidelines: an eyetracking study. cxpartners. http://www.cxpartners.co.uk/cxinsights/web_forms_design_guidelines_an_eyetracking_study.htm (Luettu 4.9.2010)

Crescimanno, B. 2005. Sensible Forms: A Form Usability Checklist. A List Apart Magazine. http://www.alistapart.com/articles/sensibleforms (Luettu 4.9.2010)

Dawson, A. 2010. How Choice Impairs Your Visitors. UX Booth. http://www.uxbooth.com/blog/how-choice-impairs-your-visitors/ (Luettu 4.9.2010)

Forman, I. 2001. Usable Forms (for an international audience). evolt.org. http://evolt.org/node/15118/ (Luettu 4.9.2010)

Google. 2010. Testing Methodologies Using Google Web Toolkit. http://code.google.com/intl/fi-FI/webtoolkit/articles/testing_methodologies_using_gwt.html (Luettu 19.10.2010)

Google. 2010b. Communicate with a Server. http://code.google.com/intl/fi-FI/webtoolkit/doc/latest/DevGuideServerCommunication.html (Luettu 19.10.2010)

Google. 2010c. Google Web Toolkit: JRE Emulation Reference. http://code.google.com/intl/fi-FI/webtoolkit/doc/latest/RefJreEmulation.html (Luettu 19.10.2010)

Google. 2010d. Google Web Toolkit: Developer's Guide - Testing: Asynchronous Testing. http://code.google.com/intl/fi-

FI/webtoolkit/doc/latest/DevGuideTesting.html#DevGuideAsynchronousTesting (Luettu 19.10.2010)

Google Web Toolkit Blog. 2010. Google Relaunches Instantiations Developer Tools - Now Available for Free. http://googlewebtoolkit.blogspot.com/2010/09/google-relaunches-instantiations.html (Luettu 19.10.2010)

Hamill, D. 2009. Validating web forms. Good usability. http://www.goodusability.co.uk/2009/08/form-validation/ (Luettu 4.9.2010)

Hamill, D. 2010. How does your web form flow? Good usability. http://www.goodusability.co.uk/2010/02/how-does-your-web-form-flow/ (Luettu 4.9.2010)

Hudson, R & Weakley, R & Firminger, P. 2005. An Accessibility Frontier: Cognitive disabilities and learning difficulties. webusability. http://www.usability.com.au/resources/cognitive.cfm (Luettu 4.9.2010)

Isomorphic Software. 2010. Smart GWT Quick Start Guide. http://www.smartclient.com/releases/SmartGWT_Quick_Start_Guide.pdf (Luettu 19.10.2010)

Isomorphic Software. 2010b. SmartGWT FAQ: Common Issues: I'm seeing misalignments or other glitches only in IE8. http://forums.smartclient.com/showthread.php?t=8159#aIE8 (Luettu 4.9.2010)

ISTQB. 2007. ISTQB:n testaussanasto. http://ttlry-fi-bin.directo.fi/@Bin/ff11c2a2e24dbb2c12c3efd1e24c70e2/1289859297/application/pdf/14155799/istqb_sanasto.pdf (Luettu 4.9.2010)

itSMF. ITIL. http://www.itsmf.fi/itil (Luettu 1.11.2010)

Jakob Nielsen's Alertbox. 2006. F-Shaped Pattern For Reading Web Content. http://www.useit.com/alertbox/reading_pattern.html (Luettu 4.9.2010)

Jivan S. 2008. SmartGWT 1.0 Released!. Sanjiv Jivan's J2EE Blog. http://www.jroller.com/sjivan/entry/smartgwt_1_0_released (Luettu 19.10.2010)

Kroll, Per 2004. Transitioning from waterfall to iterative development. IBM. http://www.ibm.com/developerworks/rational/library/4243.html (Luettu 1.11.2010)

Kruchten, Philippe. 2004. Going Over the Waterfall with the RUP. IBM. http://www.ibm.com/developerworks/rational/library/4626.html (Luettu 1.11.2010)

Nielsen, J. 2005. Ten Usability Heuristics. useit.com. http://www.useit.com/papers/heuristic/heuristic_list.html (Luettu 4.9.2010)

Oracle. 2009. Code Conventions for the Java TM Programming Language. http://www.oracle.com/technetwork/java/codeconvtoc-136057.html (Luettu 19.10.2010)

Oracle. 2010. MYSQL 5.1 Reference Manual 13.6.11 InnoDB Disk I/O. http://dev.mysql.com/doc/refman/5.1/en/innodb-disk-io.html (Luettu 19.10.2010)

Penzo, M. 2006b. Evaluating the Usability of Search Forms Using Eyetracking: A Practical Approach. Uxmatters. http://www.uxmatters.com/mt/archives/2006/01/evaluating-the-usability-of-search-forms-using-eyetracking-a-practical-approach.php (Luettu 4.9.2010)

Penzo, M. 2006a. Label Placement in Forms. Uxmatters. http://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php (Luettu 4.9.2010)

Personal Kanban. 2009. What is Kanban?. http://www.personalkanban.com/pk/primers/what-is-a-kanban/ (Luettu 19.10.2010)

Phillips, C. & Chaparro, B. 2009. Visual Appeal vs. Usability: Which One Influences User Perceptions of a Website More? Software Usability Research Laboratory. http://www.surl.org/usabilitynews/112/aesthetic.asp (Luettu 4.9.2010)

Sininen Meteoriitti. Yksikkötestaus. http://www.ketteratkaytannot.fi/fi-FI/Kaytannot/Yksikkotestaus/ (Luettu 19.10.2010)

Sonar. Documentation. http://docs.codehaus.org/display/SONAR/Documentation (Luettu 19.10.2010)

The University of Maryland. 2009. FindBugs Bug Descriptions. http://findbugs.sourceforge.net/bugDescriptions.html (Luettu 19.10.2010)

Toyota 2010. Toyota Production System. http://www2.toyota.co.jp/en/vision/production_system/just.html (Luettu 19.10.2010)

World Wide Web Consortium. 2010. W3C W3C QA: Recommended list of Doctype declarations. http://www.w3.org/QA/2002/04/valid-dtd-list.html (Luettu 4.9.2010)

Wroblewski, L. 2005. Web Application Form Design. LukeW Ideation + Design. http://www.lukew.com/resources/articles/web_forms.html (Luettu 4.9.2010)

Wroblewski, L. & Etre. 2007. Primary & Secondary Actions in Web Forms. LukeW Ideation + Design. http://www.lukew.com/resources/articles/psactions.asp (Luettu 4.9.2010)

KUVAT

LIITTEET

Liite 1 Käytetyt välineet

Liite 1 Käytetyt välineet

Sovelluskehykset ja Java

- SmartGWT(http://code.google.com/p/smartgwt/, http://www.smartclient.com/smartgwt/)
- Google Web Toolkit (http://code.google.com/intl/fi-FI/webtoolkit/)
- Java-teknologiat (http://www.oracle.com/us/technologies/java/index.html)
- Java JDK 6 (Standard Edition) Update 21 (http://www.oracle.com/technetwork/java/javase/downloads/index.html)

Asiaan liittyviä teknologioita ja standardeja

- JavaScript (http://en.wikipedia.org/wiki/JavaScript)
- HTML (http://www.w3.org/TR/1999/REC-html401-19991224/, http://en.wikipedia.org/wiki/HTML)
- CSS (http://www.w3.org/Style/CSS/, http://en.wikipedia.org/wiki/Cascading_Style_Sheets)
- XML (http://www.w3.org/TR/2006/REC-xml11-20060816/, http://en.wikipedia.org/wiki/XML)
- SHA1 (http://en.wikipedia.org/wiki/SHA-1)
- RUP (http://www-01.ibm.com/software/awdtools/rup/)
- UML (http://www.uml.org, http://en.wikipedia.org/wiki/Unified_Modeling_Language)

Ohjelmistokehitysympäristöt

- Aptana Studio 2.0 (http://aptana.com)
- Eclipse 3.6 (http://eclipse.org)
- MyEclipse 8.0 (http://www.myeclipseide.com)

Tietokantaan liittyvät

- InnoDB (http://dev.mysql.com/doc/refman/5.0/en/innodb.html)
- JDBC (http://dev.mysql.com/downloads/connector/j/)
- MySQL 5.1 (http://www.mysql.com)
- phpMyAdmin 3.3.7 (http://www.phpmyadmin.net)
- SQL Manager 2010 Lite for MySQL 4.6.0.1 (http://www.sqlmanager.net)

Kaaviointi, mallinnus ja kuvankäsittely

- ConceptDraw Project (http://www.conceptdraw.com/en/products/project/main.php)
- Excel 2007 (http://office.microsoft.com/en-us/excel)
- GWT Designer (http://code.google.com/intl/fi-FI/webtoolkit/tools/gwtdesigner/index.html)
- MySQL Workbench 5.2 (http://wb.mysql.com)
- Photoshop CS4 (http://www.adobe.com/fi/products/photoshop/photoshop/)
- PowerPoint 2007 (http://office.microsoft.com/fi-fi/powerpoint/)
- SmartDraw VP (http://www.smartdraw.com)
- Structure101 for Java (http://www.headwaysoftware.com/products/structure101/java/index.php)
- Visual Paradigm for UML 8.0 (Modeler edition) (http://www.visual-paradigm.com/product/vpuml/)

Palvelinohjelmistot ja versionhallinta

- Apache Tomcat 6.0 (http://tomcat.apache.org)
- Apache Subversion (http://subversion.apache.org)
- Subversive 0.79 (http://www.eclipse.org/subversive/)
- VisualSVN Server Standard Edition (http://www.visualsvn.com/server/)
- XAMPP Lite 1.7.3 (http://www.apachefriends.org/en/xampp-windows.html)

Selaimia

- Chrome (http://www.google.com/chrome)

- Firefox 3.6 (http://www.mozilla-europe.org/fi/firefox/)
- Internet Explorer 7, 8, 9 (beta) (http://www.microsoft.com/windows/internet-explorer/default.aspx)
- Opera 10 (http://www.opera.com)
- Safari 5 (http://www.apple.com/fi/safari/)

Käyttöjärjestelmät

- Linux (http://linux.fi)
- Windows Vista Home Premium SP2 (http://www.microsoft.com/finland/windows/windows-vista/)
- Windows XP Home Edition SP3 (http://www.microsoft.com/windowsxp/home/default.mspx)

Apuohjelmia ja pieniä verkkopalveluja

- ABBYY Finereader 6.0 (http://finereader.abbyy.com)
- BB Flashback Express 2.7.6.1586 (http://www.bbsoftware.co.uk/BBFlashBack_FreePlayer.aspx)
- CDBurnerXp 4.3.7 (http://cdburnerxp.se/)
- Clippy 1.20 (http://www.snapfiles.com/get/clippy.html)
- Delicious (http://www.delicious.com)
- Deskspace 1.5.8.2 (http://www.otakusoftware.com/deskspace/)
- Dropbox (https://www.dropbox.com)
- Epson Scan (http://www.epson.com)
- Faves (http://faves.com)
- Lovely Charts (http://www.lovelycharts.com)
- Mindomo (http://www.mindomo.com)
- <Oxygen/> XML Editor 10.3 (http://www.oxygenxml.com)
- PowerArchiver 2010 (http://www.powerarchiver.com)
- uberOptions 4.80.5 (http://uberoptions.net)
- WebResearch 3 (http://www.macropool.de/en/)
- WinSCP 4.2.9 (http://winscp.net/eng/index.php)
- Yahoo Bookmarks (http://bookmarks.yahoo.com)

Selaimen (Firefox) lisäosia (addons)

- BetterSearch (http://mybettersearch.com)
- Firebug 1.5.4 (http://getfirebug.com)
- Firecookie (http://www.softwareishard.com/blog/firecookie/)
- FireGestures (http://www.xuldev.org/firegestures/)
- Yahoo! YSlow for Firebug (http://developer.yahoo.com/yslow/)
- Xmarks (http://www.xmarks.com)

Testausohjelmia ja -plugineita Eclipseen

- CodeCover (http://codecover.org)
- CodePro Analytix Audit (http://code.google.com/intl/fi-FI/webtoolkit/tools/codepro/doc/features/audit/audit.html)
- CodePro Analytix: Code Coverage (http://code.google.com/intl/fi-FI/webtoolkit/tools/codepro/doc/features/codecoverage/code_coverage.html)
- CodePro Analytix: Metrics (http://code.google.com/intl/fi-FI/webtoolkit/tools/codepro/doc/features/metrics/metrics.html)
- FindBugs (http://findbugs.sourceforge.net)
- JMock (http://www.jmock.org)
- JUnit (http://www.junit.org)
- Grinder Analyzer (http://track.sourceforge.net)
- Grinder (http://grinder.sourceforge.net)
- GUIdancer (http://www.bredex.de/en/guidancer/first.html)
- Mockito (http://code.google.com/p/mockito/)
- NeoLoad (http://www.neotys.com/product/overview-neoload.html)
- PMD (http://pmd.sourceforge.net)
- Selenium (http://seleniumhq.org)
- SpryTest (http://www.sprystone.com/spryweb/home.do)
- TestNG (http://testng.org)
- vTest (http://www.verisium.com/products/vTest/index.html)

- Webserver Stress Tool (http://www.paessler.com/webstress)
- Xceptance LoadTest (http://www.xceptance-loadtest.com)

Muut

- JavaMail API 1.4.3 (http://www.oracle.com/technetwork/java/index-jsp-139225.html)
- Smart GWT Technical Q&A (http://forums.smartclient.com)

Ei-käytetty, mutta mainittu tekstissä

- Apache Struts 2 (http://struts.apache.org/2.x/)
- Backbase (http://www.backbase.com)
- Glassfish Server Open Source Edition (https://glassfish.dev.java.net)
- ICEFaces (http://www.icefaces.org)
- JBoss AS Community Edition (http://www.jboss.org/jbossas)
- Joomla (http://www.joomla.org)
- LDAP (http://en.wikipedia.org/wiki/LDAP)
- Microsoft Server 2008 r2 (http://www.microsoft.com/windowsserver2008/en/us/default.aspx)
- Microsoft SQL Server (http://www.microsoft.com/sqlserver/2008/en/us/)
- Mockingbird (https://gomockingbird.com)
- OpenJDK (http://openjdk.java.net)
- PostgreSQL (http://www.postgresql.org)
- Rational Rose (http://www-01.ibm.com/software/awdtools/developer/rose/)
- Sencha (ent. Ext GWT) (http://www.sencha.com)
- Spring (http://www.springsource.org)
- Vaadin (http://vaadin.com)
- VPN (http://en.wikipedia.org/wiki/Virtual_private_network)

Ohjelmistolisenssit

- AGPL (Affero General Public License) – (http://www.gnu.org/licenses/agpl-3.0.html)
- Apache Licenses (http://www.apache.org/licenses/)
- BSD (Berkeley Source Distribution) – (http://www.opensource.org/licenses/bsd-license.php)
- CDDL (Common Development And Distribution License) (http://www.sun.com/cddl/cddl.html)
- GPL v2 (GNU General Public License) – (http://www.gnu.org/licenses/gpl-2.0.html)
- GPL v3 (GNU General Public License) – (http://www.gnu.org/licenses/gpl-3.0.html)
- GPL v2 (suomeksi) (http://www.turre.com/licenses/gpl-2.0_fi.html)
- GPL v3 (suomeksi) (http://www.turre.com/licenses/gpl_fi.html)
- LGPL v2.1 (GNU Lesser General Public License) (http://www.gnu.org/licenses/lgpl-2.1.html)
- LGPL v3 (GNU Lesser General Public License) (http://www.gnu.org/licenses/lgpl-3.0.html)
- MIT (http://www.opensource.org/licenses/mit-license.php)

Dokumenttirungot

- Ohjelmistotekniikan laitoksen dokumenttirunkopankki (http://www.cs.tut.fi/ohj/dokumenttipohjat/)
- Ohjelmistotuotannon projektityö - dokumenttirungot ja muut ohjeet (http://www.cs.tut.fi/kurssit/OHJ-3500/materiaali.html)