VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Erno Isokangas

GRAAFISEN APUOHJELMAN LUONTI IT-HALLINTAJÄRJESTELMÄLLE

Tekniikka ja liikenne
2014

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Erno Isokangas
Opinnäytetyön nimi	Graafisen apuohjelman luonti IT-hallintajärjestelmälle
Vuosi	2014
Kieli	suomi
Sivumäärä	43
Ohjaaja	Pirjo Prosi

Tämä opinnäytetyö toteutettiin seinäjokelaiselle IT-yritykselle, joka on yksityisesti omistettu asiantuntijayritys. Sen tehtävänä on tuottaa tietohallinnon ja tietoturvallisuuden johtamis- ja asiantuntijapalveluita.

Työn tarkoituksena oli ensin tutustua yrityksen käyttämään IT-hallintajärjestelmään (myöhemmin tekstissä viitattu nimellä X), sen toimintaperiaatteisiin ja käyttötapoihin. Tämän jälkeen näitä tietoja hyväksikäyttäen tavoitteena oli rakentaa apuohjelma, joka parantaisi hallintajärjestelmän käytettävyyttä edelleen yrityksen tarpeiden mukaisesti. Aihe oli yritykselle tärkeä, koska he halusivat saada järjestelmän tiettyjen toiminnallisten osien käytöstä aikaisempaa nopeampaa, helppokäyttöisempää ja suoraviivaisempaa.

Tavoitteen toteuttamisessa käytettiin JavaFX-kehystä luomaan yksinkertainen graafinen käyttöliittymä. Apuohjelman toimintalogiikka (tietokantayhteys, tiedon välittäminen) rakennettiin käyttämällä X-järjestelmän omaa API-rajapintaa, jonka toiminta perustui SOAP-protokollaan ja JSON tiedonvaihtoformaattiin. SOAP-viestien luomiseen, lähettämiseen ja vastaanottamiseen käytettiin Javan SAAJ APIa.

Lopputulos oli kaikille osapuolille tyydyttävä. Sovellus valmistui aikataulun mukaisesti, toteuttaen asetetut minimivaatimukset ja enemmän.

Avainsanat IT-hallinnointi, JavaFX, SOAP, SAAJ, JSON

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikan koulutusohjelma

ABSTRACT

Author	Erno Isokangas
Title	Creating a GUI Utility Program for an IT Management System
Year	2014
Language	Finnish
Pages	43
Name of Supervisor	Pirjo Prosi

This thesis was done for an IT company located in Seinäjoki. The company is a privately owned specialist company. Its goal is to provide managerial and expert services for information management and information security purposes.

The purpose of this thesis was to first be familiarized with the IT management system used by the company (later referenced in the text as X), study how it works and how it is being used. After this the goal was to use this information to create a utility program, which would further enhance the usability of the management system according to the requirements of the company. The subject was important to the company, because they wanted to make the usage of certain functional elements in the system to be easier, faster and more straightforward.

In order to achieve this goal, the JavaFX framework was used to create a simple graphical user interface. The "business" logic of the application (database connection, data transfer) was built using the X provided API interface, which got its functionalities from the SOAP protocol and JSON data-interchange format. Java SAAJ API was used to create, send and receive SOAP messages.

The end result was satisfactory to all parties. The application was finished within the schedule, meeting all the set minimum requirements and more.

Keywords IT management, JavaFX, SOAP, SAAJ, JSON

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

LYHENTEET

API	Application Program(ming) Interface, ohjelmointirajapinta
CSS	Cascading Style Sheets, porrastetut tyyliarkit, “tyylisivut”
GUI	Graphical User Interface, graafinen käyttöliittymä
GUID	Globally Unique Identifier, tietokoneohjelmissa käyttäjälle annettava uniikki tunnistenumero
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
IDE	Integrated Development Environment, ohjelmointiympäristö
JDK	Java Development Kit, Java-ohjelmien kehitys- ja ohjelmointityökalut
JRE	Java Runtime Environment, Java-ohjelmien suoritusympäristö
JSON	JavaScript Object Notation, tekstipohjainen kevyt tiedonvaihtoformaatti
QFD	Quality Function Deployment, vaatimusmäärittely
RPC	Remote Procedure Call, etäproseduurikutsu
SAAJ	SOAP with Attachments API for Java, Javan ohjelmointirajapinta SOAP-protokollalle
SOAP	Simple Object Access Protocol, RPC:tä kehittyneempi tietoliikenneprotokolla etäproseduurikutsuille
UDDI	Universal Discovery, Description and Integration, palvelurekisteristandardi verkkopalveluille
UML	Unified Modeling Language, mallinnuskieli

URL	Uniform Resource Locator, tiedon paikkaosoite, “www-osoite”
URN	Uniform Resource Name, yksikäsitteinen nimi tiedolle
VPN	Virtual Private Network virtuaalinen erillisverkko
WSDL	Web Service Description Language, verkkopalvelujen kuvauskieli
XML	Extensible Markup Language, tietoa kuvaava merkintäkieli

ESIPUHE

”Kyllähän sut olis jo hyvä aika saada täältä pois.”

Nuo sanat ovat pyörineet päässäni siitä lähtien, kun kuulin ne ensimmäisen kerran keskustellessani mahdollisista opinnäytetyöaiheista ja valmistumisestani erään tekniikan alan opettajan kanssa alkuvuodesta 2014. Tiesin pääni sisällä tämän olevan totta.

Aloitin opiskeluni Vaasan ammattikorkeakoulussa 2008 ympäristöteknologian koulutusohjelmassa (kirjoilla jo vuodesta 2007). Kolme vuotta myöhemmin tein ehkä yhden elämäni suurimmista päätöksistä ja vaihdoin koulutusohjelmani tietotekniikan vastaavaan. Sen kai siitä saa, kun ei ole vielä lukiosta valmistuneena poikasena päättänyt mitä tulee elämällään tekemään. Vuodet vierivät. Ehkä sitä olisi vain pitänyt kuunnella omaa sisäistä ääntään silloin aikoinaan, mutta tehty mikä tehty ja eteenpäin täytyy mennä.

Nyt opinnäytetyöni on tehty. Se on valmis. Finito. Silloin tällöin en uskonut tätä päivää edes tulevaksi ja muistan vielä elävästi keväältä 2014 sen epämukavan kylmän hien, joka muistutti minua puuttuvasta opinnäytetyöaiheesta. Onneksi ahkeralla yrittämisellä on yleensä tapana palkita itsensä ja loppukeväästä 2014 löysin itselleni sopivan aiheen. Muistan tältä ajalta tunteen, joka vastasi ehkä hieman Jerikon muurien edessä seisomista. Nyt saan kuitenkin huokaista helpotuksesta, ainakin hetkeksi.

Tahdon kiittää ohjaajaani yliopettaja Pirjo Prosia hyvästä ohjauksesta, tekniikan yksikön lehtoria Kalevi Ylistä opastuksesta, opetussuunnitelmani räätälöinnistä ja muusta tuesta opintojeni loppupuolella sekä tietohallintopäällikkö Mikaa, joka auttoi minua ongelmatilanteissa ja ohjelman testauksessa tilaajayrityksen puolelta.

Seinäjoella, marraskuussa 2014

Erno Isokangas

1 JOHDANTO

Yritysten tarpeet oman liiketoimintansa harjoittamiseen ja toteuttamiseen ovat nykyään sidoksissa moneen eri suuntaan ja kiinni useista eri osatekijöistä. Yksi näistä tärkeistä tekijöistä on toimiva IT-ympäristö. Näiden ympäristöjen kasvaessa ja monimutkaistuessa myös huollon täytyy kehittyä vastaamaan lisääntynyttä valvonnan ja seurannan tarvetta. Markkinoilta löytyy nykypäivänä monenkirjavia hallinta- ja seurantaohjelmistoja, jotka ammattilaisten käsissä luovat luotettavan ylläpitoverkon. Jos talon sisältä ei löydy tarvittavaa IT-osaamista, yritykset voivat jopa ulkoistaa oman ylläpitonsa, mikä saattaa kuulostaa houkuttelevalta, koska se myös vapauttaa resursseja muihin työtehtäviin. Yritys, jolle tämäkin työ tehtiin, tarjoaa ylläpitopalveluita muille niitä tarvitseville yrityksille.

Parhaimmatkaan ohjelmistot eivät kuitenkaan välttämättä aina vastaa käyttäjänsä kaikkia mahdollisia tarpeita, jolloin saatetaan nähdä tarpeelliseksi laajentaa alkuperäistä ohjelmaa uusilla lisäominaisuuksilla. Uuden toiminnallisuuden lisääminen jo valmiiseen ohjelmaan saattaa kuitenkin olla työn ja tuskan takana, ellei peräti mahdotonta. Tällöin mahdollisuutena on esim. luoda erillinen apuohjelma, jonka kautta järjestelmää voidaan käyttää uusilla lisäominaisuuksilla höystettynä. Tähän tietenkin tarvitaan vain sopiva rajapinta, alkuperäisen ohjelman näin salliessa. Tämä opinnäytetyö liittyykin suoraan tällaisen ns. lisäpalikan luomiseen.

2 X IT-HALLINTAJÄRJESTELMÄ

Tässä kappaleessa tutustutaan lyhyesti X IT-hallintajärjestelmän toiminnallisuuteen, mitä tarkoitusta varten ohjelmisto on kehitetty ja mistä osatekijöistä se koostuu. Kappale käsittelee tämän hallintajärjestelmän eri osatekijöitä ja antaa kuvan näiden yhteistoiminnasta.

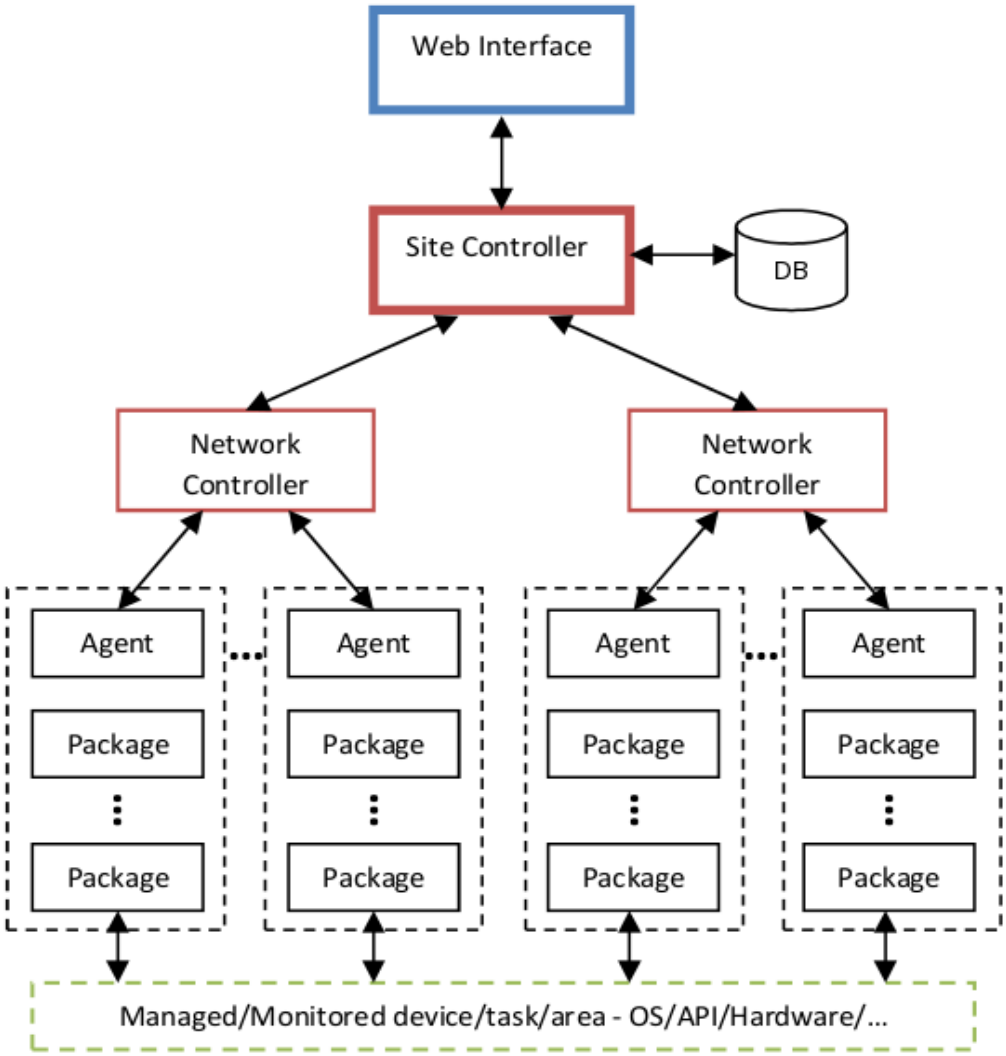
2.1 Mikä on X?

X on työkalu, jonka avulla voidaan seurata ja hallinnoida tietokoneita ja muita lisälaitteita. Tätä työkalua voidaan käyttää

- tiedottamaan käyttäjää tulevien tai jo olemassa olevien ongelmien vaikutuksesta liiketoimintaan
- toteuttamaan laitteistojen seurantaa ja raportoimaan niiden käytöstä ja senhetkisestä tilanteesta
- varoittamaan käyttäjää tulevista ongelmatilanteista ennen kuin todellista haittaa ehtii syntymään
- kartoittamaan IT-infrastruktuurin suorituskyvyn tasoa
- toteuttamaan työmäärän seuranta ja laskuttamaan asiakasta vastaavasti.

X-ohjelman tarkoituksena on tarjota tehokas kokonaisratkaisu IT-järjestelmien etätuelle ja -seurannalle ja edesauttaa liiketoimintaa. Käyttöliittymäksi sopii verkkoselain, mikä mahdollistaa työskentelyn mistä vain ja milloin vain. Verkkokäyttöliittymä tekee X-ohjelmasta nopean ja helpon käyttää ja käyttö on myös helposti opittavissa. Mahdolliset seurattavat tietokoneet ja järjestelmät voivat sijaita joko samassa rakennuksessa tai missä päin maailmaa tahansa, Internetin välityksellä ne ovat kaikki saavutettavissa. /3, 12/.

2.2 X-järjestelmän yleiskuvaus



Kuvio 1. X-järjestelmädiagrammi /3, 13/.

X-järjestelmä sisältää viisi tärkeää osatekijää, jotka käsitellään seuraavaksi. Kuvio 1 kuvastaa näiden tekijöiden suhdetta toisiinsa hieman selkeämmin. Tämä diagrammi on esitetty uudelleen myöhemmin hieman eri muodossa **(Kuvio 2.)**.

2.2.1 The Web Interface

Verkkoselain tarjoaa keinon käyttää itse ohjelmistoa ja tällöin on mahdollista sekä hallinnoida verkkoja että seurata virhetapahtumia.

2.2.2 The Site Controller

Jokaiseen X-asennukseen sisältyy yksi Site Controller. Sen tarkoituksena on koordinoida kyseisen X-järjestelmän toimintoja ja tarjota tietoa verkkoselaimessa esitettynä. Se on yhteydessä päätietokantaan ja näin ollen se vastaa myös kaikesta tiedon tallennuksesta ja varastoinnista.

2.2.3 The Network Controller

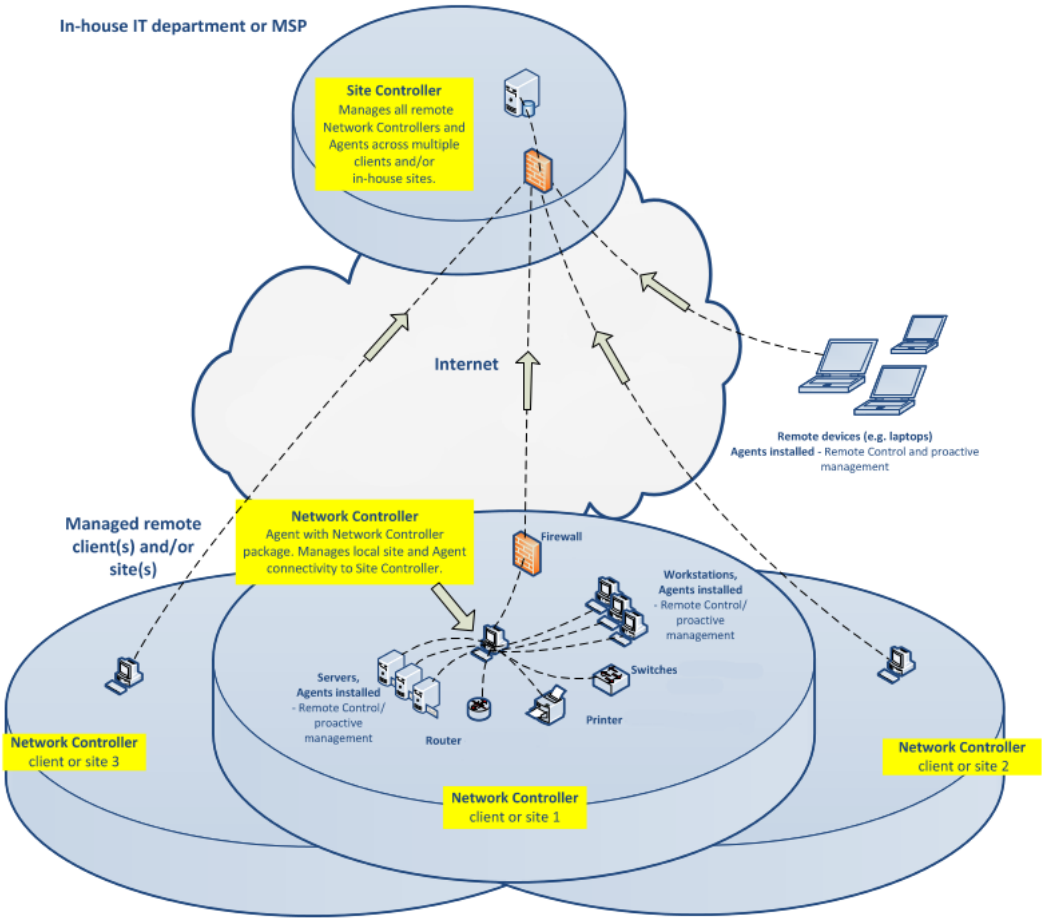
Järjestelmä voi sisältää yhden tai useamman Network Controllerin. Jokainen Network Controller saa hallinnoitavakseen jonkin tietyn osan verkosta ja tällöin ne toimivat tiedonvälittäjinä järjestelmän agenteille, joita löytyy jokaisesta verkossa olevasta seurattavasta laitteesta. Network Controllerit ovat yksinkertaisia olioita – niiden tarkoituksena on pääasiallisesti toimia yhdyskäytävänä agenttien ja Site Controllerin välillä. Niillä ei ole omaa tietokantaa. Tyypillisesti jokainen hallinnoitava verkko saa oman Network Controllerin, mutta joissakin tapauksissa tämä ei ole pakollista, jos hallinnoitava verkko on hyvin pieni (esim. yksittäiset kannettavat).

2.2.4 The Agent

Agentti asennetaan seurattavaan laitteeseen. Agentti kykenee keräämään tietoja laitteesta, tekemään muutoksia laitteen asetuksiin jne. Agentti toimii isäntänä järjestelmän paketeille ja se suorittaa myös muutamia järjestelmän sisäisiä toimintoja.

2.2.5 Packages

Agentit isännöivät paketteja. Jokainen paketti on vastuullinen hallinnoimaan jotain tiettyä osaa seurattavasta isäntäkoneesta. Esimerkiksi käyttöjärjestelmäpaketti kerää tietoa isäntäkoneen käyttöjärjestelmän asennuksesta, ympäristöstä, suorituskyvystä ja vastaa asiakkaan pyytämiin kutsuihin. /3, 13/.



Kuvio 2. X-verkkodiagrammi /3, 15/.

Yksittäinen Site Controller kykenee hallinnoimaan useita verkkoja samaan aikaan. Verkot voivat kuulua yhdelle ja samalle tai useammalle organisaatiolle, joita X-ohjelmassa kutsutaan asiakkaiksi (client). Seurattava verkko voi sijaita samassa paikallisverkossa kuin Site Controller tai se voi olla etäverkko, johon yhteys luodaan tällöin joko Internetin tai VPN:n yli. /3, 15/

3 TYÖSSÄ HYÖDYNNETYT TEKNOLOGIAT

Edellisessä kappaleessa saatiin tietää, mitä ominaisuuksia X tarjoaa ja mitä tarkoitusta varten ohjelma on kehitetty. Tässä kappaleessa kurkistetaan hieman pintaa syvemmälle ja tutustutaan työssä hyödynnettyihin teknologioihin, joiden pohjalta oli mahdollista rakentaa visioitu apuohjelma. Kerrottavaa löytyisi sivukaupalla, joten keskitytään vain tämän opinnäytetyön tehtävänannon kannalta tärkeisiin osa-alueisiin, jotta pysytään asian ytimessä. Ensin käsitellään graafisen käyttöliittymän (GUI, Graphical User Interface) luonnissa käytetty JavaFX-kehys, sitten SAAJ API, XML-merkintäkieli, X-ohjelman SOAP-protokolla lyhyellä historiakatsauksella ja viimeisenä JSON tiedonvaihtoformaatti.

Se miten nämä teknologiat toimivat keskenään lopullisessa sovelluksessa, on luettavissa kappaleesta 5, joka käsittelee lähemmin apuohjelman toteutusvaihetta.

3.1 JavaFX

JavaFX on Oraclen kehittämä kokoelma grafiikka- ja mediapaketteja, jotka antavat ohjelmistokehittäjille mahdollisuuden suunnitella, luoda, testata ja ottaa käyttöön rikkaita asiakasohjelmia. JavaFX kehitettiin korvaamaan vanhempi Java Swing -kehys, mutta tämän työn kirjoitushetkellä Swing-kirjastot ovat kuitenkin vielä käytettävissä näin siirtymävaiheessa /7/. Koska JavaFX-kirjasto on kirjoitettu API-muodossa, voidaan lähdekoodissa tällöin viitata muihin olemassa oleviin Java-kirjastoihin ja hyödyntää niitä parhaaksi katsotulla tavalla.

JavaFX tarjoaa monipuoliset työkalut graafisten ohjelmien luomiseen. Käyttöliittymän ja ns. ”business” logiikan voi surutta toteuttaa samassa koodissa, mutta mahdollisuutena on myös jakaa nämä osa-alueet erilleen: käyttöliittymä voidaan rakentaa FXML-skriptikielellä ja tyylisivuilla (CSS, Cascading Style Sheets) ja logiikka lisätä myöhemmin erikseen jonkin muun tahon toimesta. JavaFX API on käytettävissä JDK (Java Development Kit) ja JRE (Java Runtime Environment) versiosta 7 eteenpäin. /5/.

3.2 SAAJ

SAAJ (SOAP with Attachments API for Java) on JavaFX:n tavoin Oraclen kehittämä API SOAP-viestien lähettämiseen ja vastaanottamiseen. SAAJ noudattaa SOAP 1.1, 1.2 ja 1.3 spesifikaatioita ja on näin ollen muutenkin otollinen työkalu tämän työn kannalta. /8/

3.3 XML

XML (Extensible Markup Language) on W3C:n (World Wide Web Consortium) standardoima yksinkertainen ja joustava tekstiformaattiin perustuva merkintäkieli, jota käytetään tässä työssä tiedon välityksessä, kuten nykyään laajalti on tapana siirrettäessä dataa Internetin välityksellä /9/. XML on tärkeä kieli kaikenlaisissa sovelluksissa, koska sen tarjoamat edut ovat houkuttelevia. XML:n etuja ovat mm. siirrettävyys, yhteiskäytettävyys ja lähestulkoon rajaton määrä soveltamismahdollisuuksia. /1, 19-22/

3.4 XML-RPC

Ennen SOAPin esiintuloa, käytössä oli RPC (Remote Procedure Call) eli etäproseduurikutsu, joka kehitettiin hajautettuja järjestelmiä varten, joissa asiakassovellus voi RPC:tä käyttäen pyytää jotain verkon palvelinta suorittamaan sille jonkin tehtävän. RPC:n tarkoituksena oli tarjota kevyt ja yksinkertainen rajapinta, joka takaisi yhteistoiminnan epäyhtenäisten järjestelmien ja sovellusten välille riippumatta esim. ohjelmointikielestä. RPC:n yhtenä valttikorttina on se, että tiedon kuljettamiseen ei vaadita mitään erikoista ja uniikkia protokollaa, vaan tiedon kuljetus voidaan toteuttaa puhtaasti HTTP-protokollan yli. /1, 322-325/

RPC:n lukuisista hyvistä puolista huolimatta protokollalla oli yksi suuri heikkous, enkoodaus, eli minkälaisessa muodossa informaatio kuljetetaan lähettäjältä vastaanottajalle ja takaisin. Ongelma ratkaistiin käyttämällä enkoodauksessa XML:ää, joka tarjosi yksinkertaisen ja W3C:n standardoiman esitystavan etäproseduurikutsuille. /1, 326/

3.5 SOAP

SOAP (Simple Object Access Protocol) on nykyään yksi tärkeimmistä ja tunnetuimmista tekniikoista web-ohjelmoinnin saralla ja sillä on ollut tärkeä osa esim. Microsoftin .NETin ja ns. ”peer-to-peer-vallankumouksen” kehityksessä. SOAP on kevyt ja yksinkertainen protokolla ja se pohjautuu jo edellä mainittuun XML-RPC:hen, tuoden mukanaan muutamia uusia ominaisuuksia, kuten tuen omille tietotyypeille, epästandardien tietotyyppien sarjallistamisen, tuen erilaisille enkoodauksille ja paremman virheenkäsittelyn. Monet yritykset ovat ottaneet SOAPin omakseen ja esim. sekä Microsoft että Apache ovat olleet tärkeässä asemassa SOAPin jatkokehityksessä useiden erilaisten projektien muodossa. /1, 359-361/

SOAP-palveluun kuuluu kolme erilaista pääkomponenttia: SOAP-kirjekuori, enkoodaussäännöt sekä palvelimen ja asiakkaan vuorovaikutusten määrittely.

SOAP-kirjekuori (SOAP envelope) sisältää

- lähettäjän tiedot
- viestin (datan)
- vastaanottajan tiedot
- tiedot enkoodauksesta.

SOAP on XML-RPC:tä kehittyneempi protokolla, koska SOAPin kirjekuoren ylätunnisteen (header) avulla voidaan määrittää viestin prosessointitapa. Tällöin sovellus voi päätellä, osaako se käsitellä saamansa tiedon ja mahdollisesti hylätä viestin kokonaan, jos sovellus ei kykene prosessoimaan saamaansa informaatiota halutulla tavalla. /1, 362/.

Paremman **enkoodauksen** johdosta SOAP mahdollistaa uusien tietotyyppien (complexType) määrittelemisen XML-skeemoja hyväksi käyttäen, kun taas XML-RPC:ssä tulee pidättäytyä ennalta määritellyissä tietotyypeissä. Paremmat enkoodausmahdollisuudet tekevät SOAPista edeltäjäänsä joustavamman. /1, 362-363/

Vuorovaikutuksesta puhuttaessa viitataan yleensä SOAP-kutsuihin ja -vastauksiin. Ohjelmointivaiheessa kyetään hyvin eksplisiittisesti (”suorasti ilmaistu”) määrittelemään kutsun muoto (vastaanottaja, kutsuttava metodi, enkoodaus jne.), kun taas edeltäjänsä XML-RPC toteuttaa monet määrittelyt implisiittisesti (”epäsuorasti ilmaistu”) ohjelmoijalta piilossa. Toisin sanoen SOAP antaa paremmat mahdollisuudet kontrolloida ohjelman lopullista toiminnallisuutta ja tällöin ohjelma saadaan tekemään juuri sitä, mitä halutaan. /1, 364/

SOAPissa kutsut ja vastaukset voidaan toteuttaa kahdella tavalla, joko perinteisellä XML-RPC -tyylillä tai SOAP-*viestinä*. SOAP-RPC -kutsu on toteutettu perinteisen XML-RPC -etäproseduurikutsumallin mukaisesti, käyttäen hyväksi SOAPin mukanaan tuomia parannuksia, joista edellä onkin jo puhuttu. Mahdollisuutena on kuitenkin myös käyttää ns. SOAP-viestejä. Tällöin informaatio välitetään viesteinä, eikä asiakasohjelman tarvitse tällöin tuntea palvelimella olevien palveluiden metodeja. Viestien käyttö on ohjelmallisesti hieman vaikeampi toteuttaa kuin perinteinen XML-RPC -tyylinen lähestymistapa. /1, 369/

Esimerkki SOAPilla lähetettävästä kutsusta kirjauduttaessa johonkin palveluun voisi näyttää seuraavalta.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">                    (1)
 <s:Body>
  <Login xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://raven.net">
   <username>testuser</username>
   <password>fakepassword</password>
  </Login>
 </s:Body>
</s:Envelope>
```

3.6 UDDI ja WSDL

SOAPin tapaisissa verkkopalveluympäristöissä vaaditaan, että sovellus kykenee vaihtamaan tietoa sekä palveluiden että muiden sovellusten kanssa. Pelkkä SOAPin käyttö yksinään ei takaa kaikenkattavaa toiminnallisuutta vaan tiedon vaihto vaatii muita standardeja avuksi. Kaksi tärkeintä tällaista standardia ovat UDDI (Universal Discovery, Description and Integration) ja WSDL (Web Service

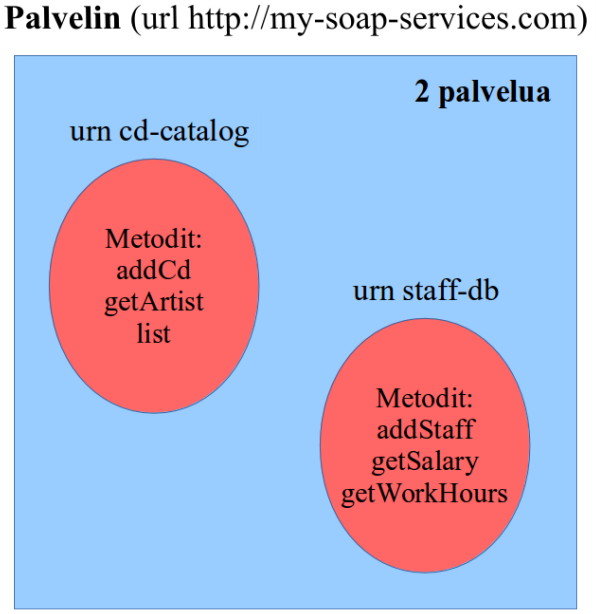
Description Language). /1, 388/

UDDI on eräänlainen rekisteri, jonne palveluiden tuottajat voivat rekisteröidä palvelunsa. UDDIn ideana on tarjota palvelu, jonne kuka tahansa voi rekisteröidä haluamiaan palveluita ja myös etsiä niitä tarpeiden mukaan. /1, 392/

WSDL:n tarkoituksena on antaa kuvaus kyseessä olevasta palvelusta. Tämä on tärkeää, jotta asiakasohjelma pystyisi kommunikoimaan palvelun kanssa onnistuneesti. WSDL sisältää tiedot mm.

- palvelun nimestä (URN-osoite)
- palvelun paikasta (URL-osoite)
- palvelun tarjoamista metodeista
- syöte- ja tulosteparametrien tyypeistä metodeille.

Yhdessä nämä tiedot antavat palvelusta kattavan kokonaiskuvan **(Kuvio 3.)**. /1, 392-393/.



Kuvio 3. Havainnekuva SOAP-palvelun rakenteesta.

3.7 JSON

JSON (JavaScript Object Notation) on kevyt tiedonvaihtoformaatti. Sitä on helppo ymmärtää ja kirjoittaa ja se on myös tietokoneella helposti jäsennettävissä ja luotavissa. Formaatti on alkujaan lähtöisin JavaScript ohjelmointikielestä, mutta se ei kuitenkaan ole millään tavalla sidonnainen JavaScriptiin, vaan se on käytettävissä useilla eri kielillä aina C-kielestä Javaan ja Pythoniin asti. JSON on yksinkertainen tekstiformaatti, joka on täysin kieliriippumaton, mutta käyttää hyväkseen muista kielistä löytyviä tuttuja käytänteitä. JSON rakentuu kahden rakenteen varaan:

- Kokoelma nimi/arvo pareja. Monissa kielissä tämä realisoituu esim. objektina, tietueena, sanakirjana, hash-taulukkona, avainlistana tai assosiatiivisena taulukkona.

- Järjestetty arvolista. Monissa kielissä tämä realisoituu taulukkona (array), vektorina, listana tai sekvenssinä.

Edellä mainitut ovat universaaleja tietorakenteita, joita lähestulkoon jokainen moderni ohjelmointikieli tukee ja ymmärtää muodossa tai toisessa. /4/.

JSON-tyyppinen vastaus esim. kirjauduttaessa sisään johonkin palveluun voisi näyttää seuraavalta.

```
{                                                                    (2)
        ”LoginType”:1,
        ”UserID”:”123”,
        ”Success”:true,
        ”SessionGUID”:”8dnf9g-0030rt-fijg04-490fhh”,
        ”Error”:””
}
```

4 APUOHJELMAN MÄÄRITTELY JA SUUNNITTELU

Tässä kappaleessa käydään läpi vaiheet, joiden avulla apuohjelma suunniteltiin ennen varsinaisen toteutusvaiheen aloittamista. Näitä vaiheita olivat mm. apuohjelmalta vaadittujen ominaisuuksien määrittely sekä erinäisten UML-kaavioiden (Unified Modeling Language) luonti kuvastamaan apuohjelman toimintaa ja käyttömahdollisuuksia.

4.1 Vaatimukset ja määrittelyt

Apuohjelman luontiprosessi aloitettiin kuten minkä tahansa uuden sovelluksen luonti: palaverilla tilaajan ja tekijän välillä. Tässä palaverissa käytiin läpi vaatimukset, joita apuohjelmalle haluttiin asettaa toiminnallisuuden ja käyttökokemuksen näkökulmasta. Mitä ohjelman tulisi tehdä? Miltä ohjelman tulisi ulkoisesti näyttää? Miltä ohjelman käytön tulisi tuntua? Palaverissa saatiin aikaiseksi muistio, johon listattiin halutut ominaisuudet. Näiden pohjalta voitiin aloittaa suunnittelu ja vaatimuksia voitiin tarpeen tullen myöhemmin tarkentaa tai lisätä.

Työn tilaajan toiveena oli luoda alkuperäisestä hallintajärjestelmästä irrallaan oleva erillinen sovellus, joka mahdollistaisi järjestelmän tiettyjen toiminnallisten osien käytön aikaisempaa nopeammin, helppokäyttöisemmin ja suoraviivaisemmin. Yrityksen työntekijät olivat jo tottuneet käyttämään järjestelmän tarjoamaa selainpohjaista käyttöliittymää ja apuohjelma päätettiinkin rakentaa selainversiota mukaillen, jotta käyttötuntuma olisi alusta lähtien pysynyt mahdollisimman tuttuna.

4.1.1 Vaatimusmäärittely, QFD

Määrittelypalaverista saatujen vaatimusten pohjalta luotiin vaatimusmäärittely eli QFD (Quality Function Deployment).

Vaatimusmäärittely on tiimipohjainen tekniikka, joka tarjoaa keinot tunnistaa ja kääntää asiakkaan esille tuomat vaatimukset teknisiksi määritelmiksi (spesifikaatiot), joita voidaan sitten käyttää tuotteen suunnitteluun ja tuottamiseen /10, 1/.

Vaikka vaatimusmäärittelystä saisi luotua hyvinkin monimutkaisen kokonaisuuden, pyritään tässä tapauksessa asiat tekemään mahdollisimman yksinkertaisesti. Tässä vaatimusmäärittelyssä listataan asiakkaan vaatimukset kolmiportaisesti priorisoituna: pakolliset (täytyy olla), odotetut (tulisi olla) ja extrat (voisi olla). Pakolliset vaatimukset ovat niitä ominaisuuksia, joita asiakas tietoisesti haluaa tuotteelleen. Odotetut vaatimukset ovat yleensä taustalla olevia yleismaallisia vaatimuksia, joita ei välttämättä asiakkaan taholta erikseen vaadita, mutta jotka tapauksesta riippuen tulisi löytyä lopullisesta tuotteesta erikseen sanomatta. Extrat taasen ovat extraa, eli lisäominaisuuksia, jotka voidaan tarpeen mukaan toteuttaa tai jättää pois.

Yrityksen kanssa käytyjen keskustelujen pohjalta voitiin luoda taulukon 1 mukainen vaatimusmäärittely.

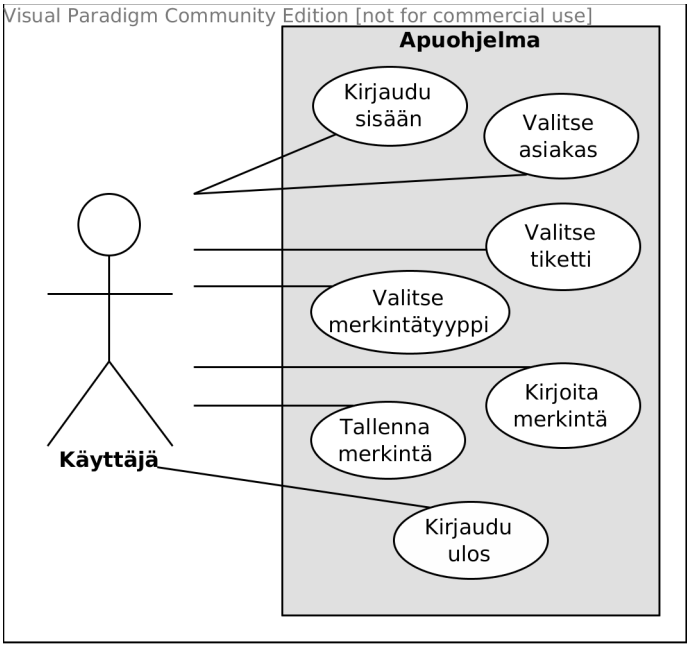
Taulukko 1. Vaatimusmäärittely.

Pakolliset	• merkintöjen lisääminen tiketteihin (pääasiassa Time Note -tyyppi) • graafinen käyttöliittymä • käyttöliittymä selainversion kaltainen • tikettien hakeminen tietokannasta asiakaskohtaisesti • tiketin tietojen näyttäminen
Odotetut	• kevyt sovellus • käytettävissä eri käyttöjärjestelmäalustoilla (Windows, Linux, Mac) • syöttötietojen tarkistus ennen merkinnän tallennusta
Extrat	• mahdollisuus käyttää mobiililaitteilla • apuohjelman käyttö näppäimistöllä • muunlaisten merkintöjen lisääminen tikettiin • taustakuva • vanhojen merkintöjen selaaminen • uusien tikettien luominen

Vaatimusmäärittelyssä mainittu tiketti (ticket) viittaa asiakkaalta tulleeseen ilmoitukseen laite- tai järjestelmäviasta.

4.1.2 Käyttötapauskaaviot

Käyttötapauskaavio on yksi UML-mallinnuksen yksinkertaisimmista kaaviotyypeistä. Yksinkertaisuudestaan huolimatta se on tärkeä, koska se antaa kuvattavasta aiheesta helppolukuisen notaation, jota asiaan perehtymättömänkin on helppo ymmärtää. Apuohjelmaa kuvaamaan luotu käyttötapauskaavio **(Kuvio 4.)** sisältää kaikki tämän kuviotyypin peruselementit: aktorin (esim. käyttäjä), käyttötapaukset (kuplat) ja aktoria ja käyttötapauksia yhdistävät viivat (aktori osallistuu tapaukseen). /2, 77-78/



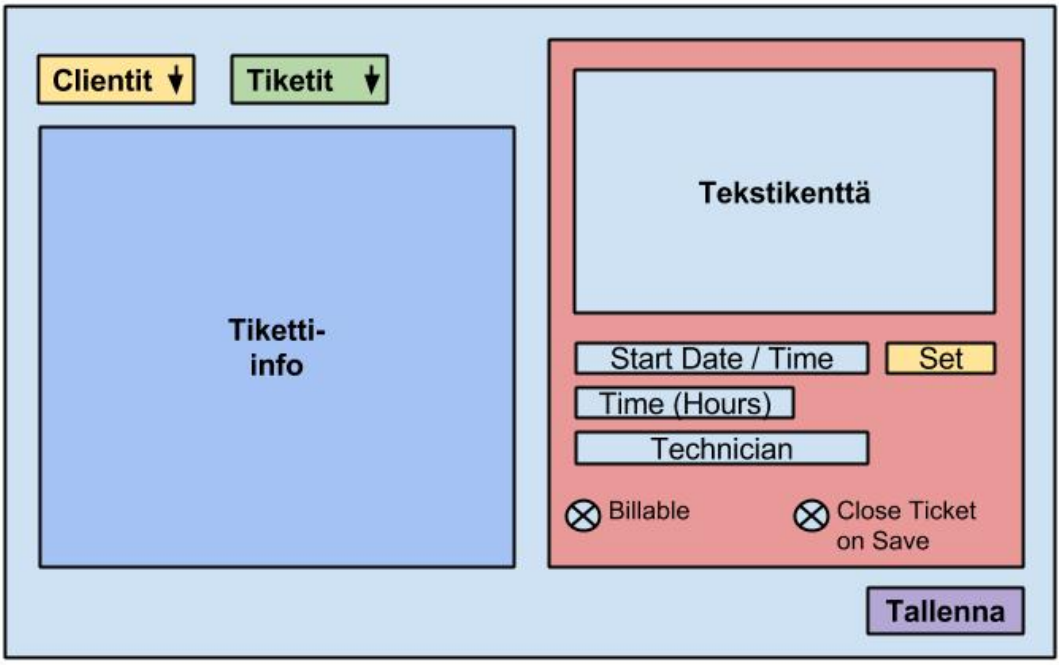
Kuvio 4. Apuohjelman käyttötapauskaavio.

4.1.3 Käyttöliittymä

Käyttöliittymän haluttiin olevan mahdollisimman samankaltainen alkuperäisen selainpohjaisen käyttöliittymän kanssa. Uudenlaisen ulkoasun tekeminen olisi joka tapauksessa ollut huono valinta, koska työntekijät olivat jo tottuneet tietynlaiseen ulkoasuun. Pieniä eroavaisuuksia näiden käyttöliittymien välillä tuli kuitenkin olemaan, mutta tuntuma pysyi suurimmalta osin hyvin samankaltaisena.

Kuviossa 5 on esitetty projektin alkuvaiheilta varhainen piirros käyttöliittymän ul-

koasusta. Tätä kuviota on mielenkiintoista verrata myöhemmin esitettyyn valmiiseen käyttöliittymään **(Kuvio 15.)** ja tehdä johtopäätöksiä siitä, kuinka hyvin alkuperäinen visio säilyi projektin aikana. Voinen tässä vaiheessa jo paljastaa, että suuria linjamuutoksia ei ollut odotettavissa, olihan ulkoasu suunniteltu alkuperäisen selainversion pohjalta.



Kuvio 5. Varhainen GUI-hahmotelma.

4.2 Apuohjelman suunnittelu

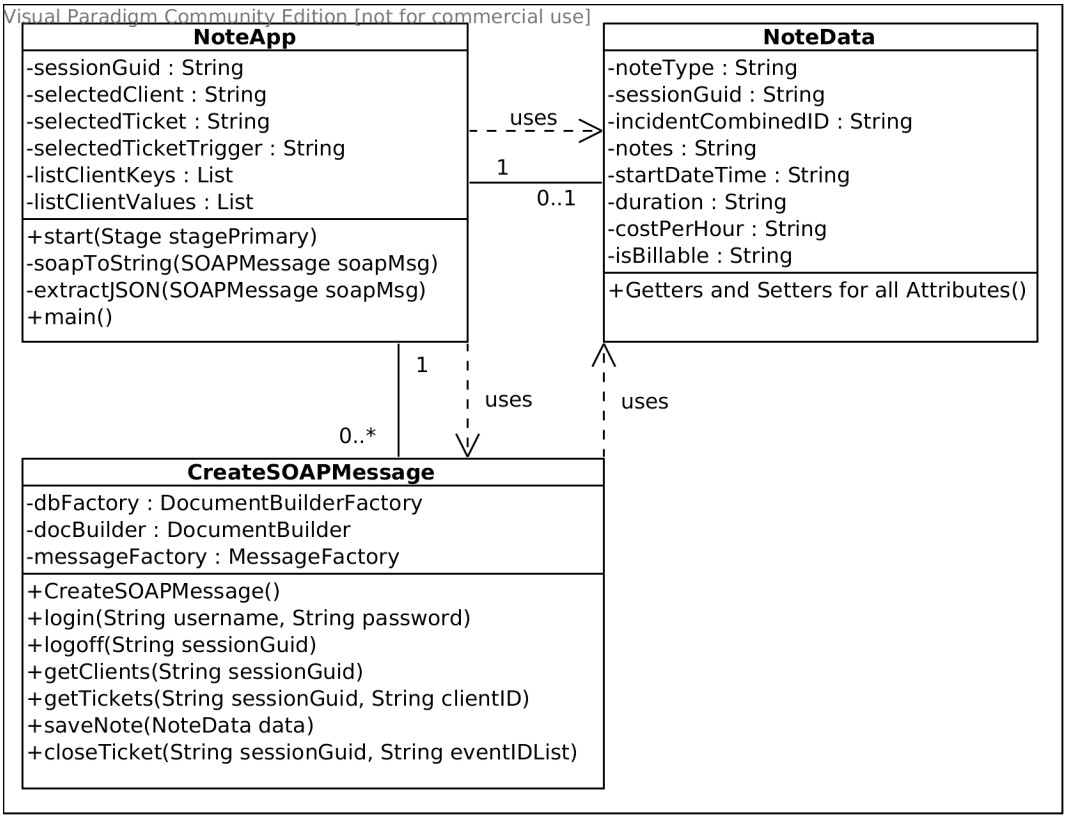
Seuraavaksi esitetään joukko eri tyyppisiä UML-kaavioita kuvastamaan apuohjelman rakenteita ja toimintaa. UML on vuodesta 1997 lähtien ollut ainakin ohjelmistoalalla korvaamaton työkalu, jonka avulla suurien ohjelmien ja ohjelmistojen suunnittelu ja kuvaaminen on ollut mahdollista. /2, 71-72/

4.2.1 Luokkakaaviot

Luokkakaaviot ovat ehkä yksi UML:än tärkeimmistä kaavioista niiden tarkkuuden johdosta. Luokkakaavioilla voidaan hyvin helposti mutta samalla detaljimaisesti kuvata erilaisten ohjelmakoodissa käytettävien luokkien sisältö (attribuutit ja metodit) ja mm. niiden interaktiot, assosiaatiot sekä lukumääräsuhteet toistensa kans-

sa. Tässä tarkkuudessa on mahdollista päästä jopa niin pitkälle, että luokkakaavioiden pohjalta voidaan generoida valmiita ohjelmakoodia sisältäviä luokkia tai miksei vaikka valmiita tietokantarakenteitakin. /2, 95/

Apuohjelman luokkakaavio **(Kuvio 6.)** sisältää kolme luokkaa. NoteApp-luokka on tämän ohjelman sydän, josta kaikki muu toiminta hajautuu. Luokkien mittasuhteita katsomalla ei välttämättä saa kuvaa NoteApp-luokan laajuudesta, joka on näistä kolmesta luokasta ehdottomasti suurin. Luokka sisältää paljon koodia graafisen käyttöliittymän rakentamista, SOAP-viestien lähettämistä, vastaanottamista ja virheentarkistuksia varten. Ohjelman käytön aikana lähetetyt SOAP-viestit luodaan ennen niiden lähettämistä CreateSOAPMessage-luokassa, jossa jokaista viestityyppiä varten on oma luontimetodinsa. Tätä luokkaa tutkimalla saa kuvan siitä määrästä palvelinkutsuja, joita ohjelmaa joutuu suorituksensa aikana tekemään. NoteData-luokkaa käytetään tallennettaessa uutta merkintää tikettiin.

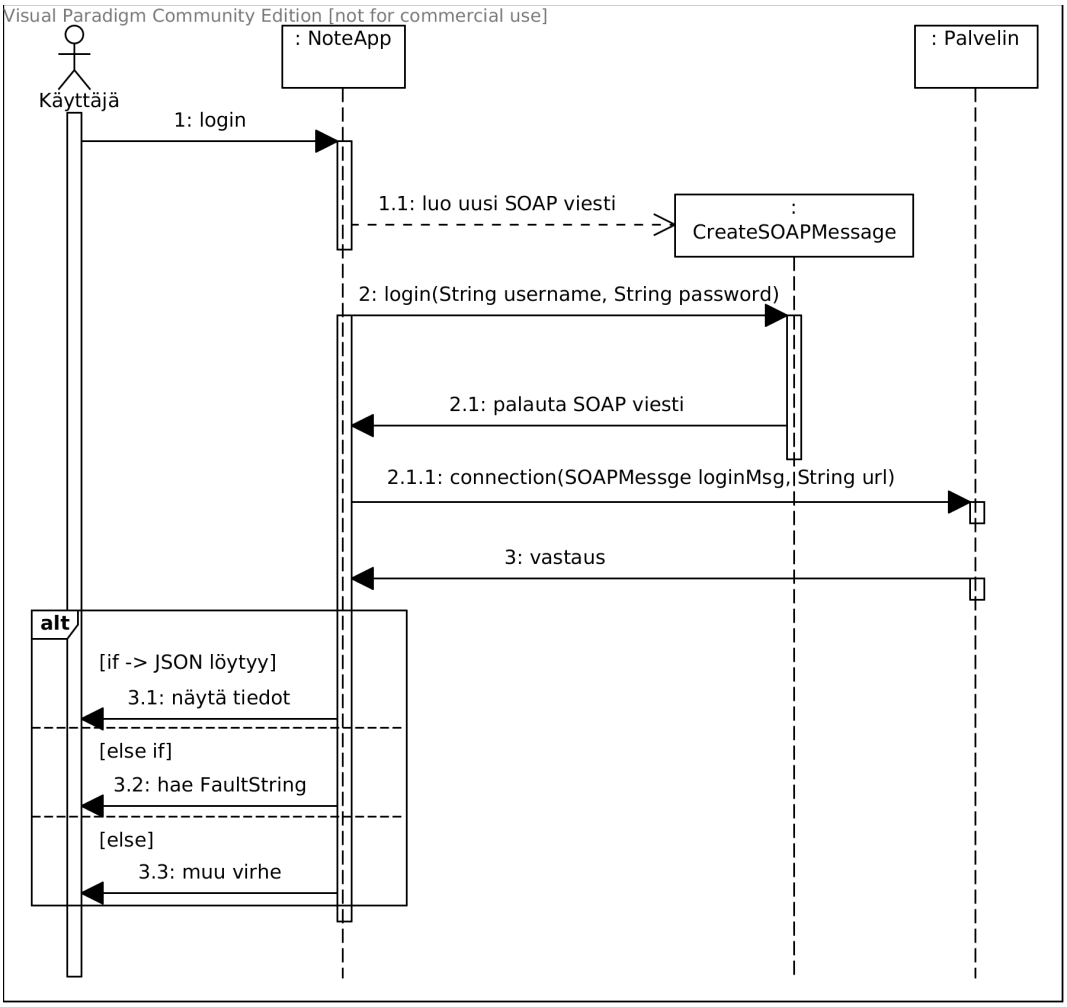


Kuvio 6. Apuohjelman luokkakaavio.

4.2.2 Sekvenssikaaviot

Sekvenssikaaviot (myös tapahtumasekvenssikaaviot) ovat luokkakaavioiden ohella yksi tärkeimmistä kaaviotyypeistä niiden yksinkertaisuuden ja havainnollisuuden johdosta, erityisesti oliopohjaisia ohjelmia suunniteltaessa. Sekvenssikaavioiden tarkoituksena on kuvata olioiden (tai luokkien) välistä kommunikointia, jossa vaakasuuntaisten nuolien voidaan ajatella olevan olioiden metodeja. Metodikutsun ollessa synkroninen (täytetty nuoli), olio jää odottamaan vastausta kutsulleen ennen ohjelmakoodin suorituksen jatkamista. Asynkronisessa kutsussa (avoin nuoli) vastausta ei jäädä odottamaan. Oliot kuvataan yleensä kaavion yläreunassa olevilla laatikoilla, poikkeuksena ne oliot, jotka voidaan luoda aina tarvittaessa ohjelman suorituksen aikana. Olioista lähteviä katkopystyviivoja kutsutaan elämänlangoiksi, jotka kuvastavat olion elinaikaa. /2, 97-99/

Otetaan sekvenssikaaviosta esimerkkinä apuohjelman sisäänkirjautumisprosessi **(Kuvio 7.)**. Käyttäjän yrittäessä kirjautua sisään, ohjelma luo uuden instanssin CreateSOAPMessage-luokasta ja kutsuu tämän login-metodia. Käyttäjän syöttämät käyttäjätunnus ja salasana liitetään SOAP-viestiin ja lähetetään palvelimelle käsiteltäväksi. Vastauksen tullessa takaisin ohjelma yrittää kolmella eri tavalla käsitellä saamansa informaation. ”Yritä ensin hakea saadusta vastauksesta serialisoitu JSON-merkkijono ja tulkitse se. Jos tämä ei onnistu, hae SOAP-vastauksen FaultString. Jos tämäkään ei onnistu, kyseessä on jokin muu virhe.”

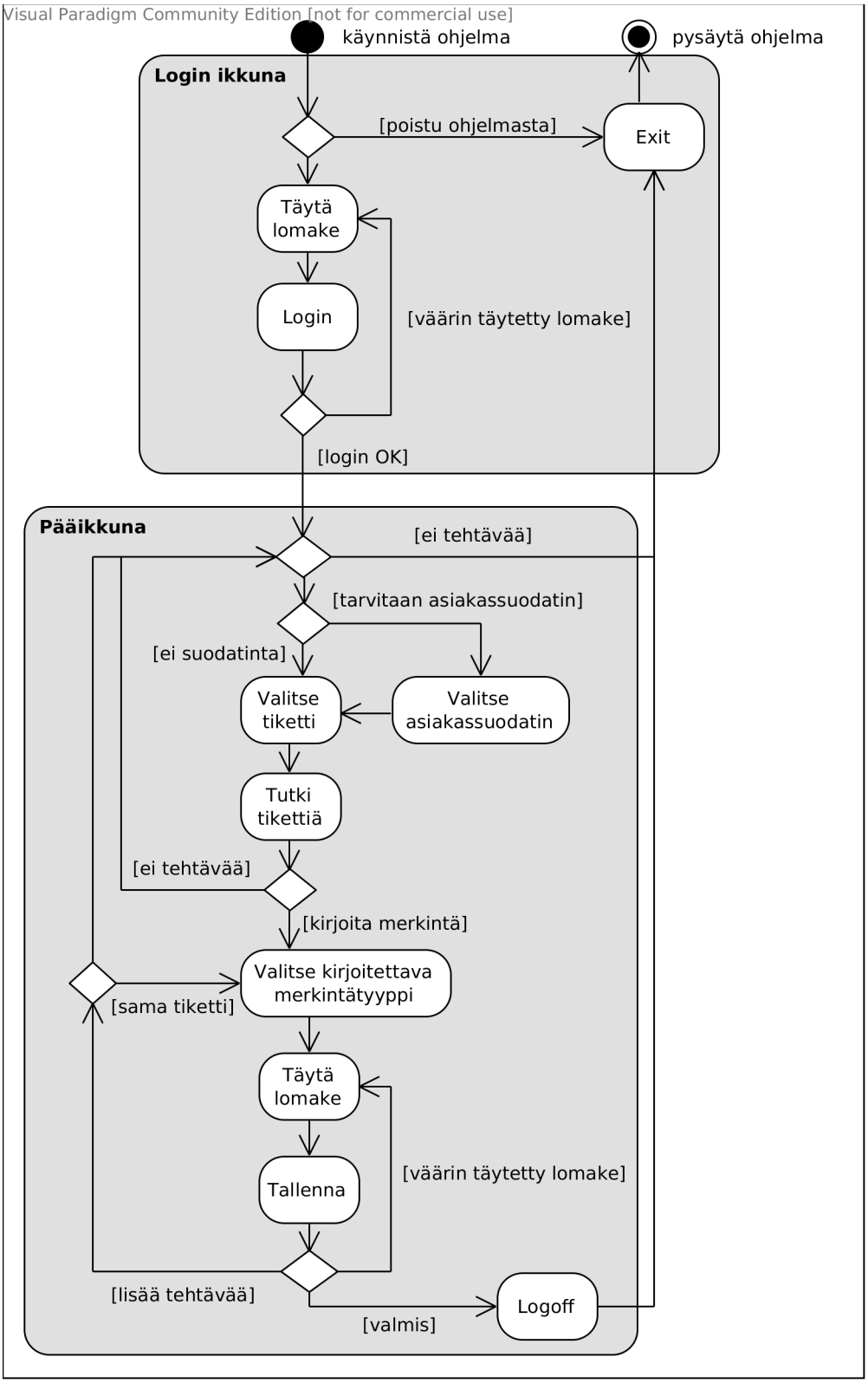


Kuvio 7. Sekvenssikaavio sisäänkirjautumisesta.

Apuohjelman muistakin palvelinkutsuista **(Kuvio 6.)** voisi toteuttaa samanlaisen sekvenssikaavion, mutta koska nämä tapahtumat ovat joidenkin metodien pieniä eroavaisuuksia lukuun ottamatta identtisiä, ei tämä ole tarpeellista. Hallintajärjestelmän palvelinpuoli on sen verran yksinkertaisesti toteutettu, että kaikki tuleva kommunikointi, oli kyseessä sitten sisäänkirjautuminen tai tikettiin lisättävien merkintöjen tallennus, voidaan hoitaa lähes samalla ohjelmakoodilla. Suurin ero tulee vastaan vasta silloin, kun palvelimelta saatua tietoa aiotaan jalostaa eteenpäin.

4.2.3 Aktiviteettikaaviot

Aktiviteettikaavio (myös tilakaavio tai ns. *flow chart*, vuokaavio) kuvastaa jonkin järjestelmän, laitteen tai kokonaisuuden toimintaa tai prosessia /2, 105/. Tämän projektin tiimoilta aktiviteettikaaviot eivät ehkä kilpaile aivan samassa tärkeysluokassa luokka- ja sekvenssikaavioiden kanssa, mutta ne antavat kuitenkin hieman lisäinformaatiota apuohjelman pintapuolisesta toiminnasta käytön aikana **(Kuvio 8.)**.



Kuvio 8. Apuohjelman aktiviteettikaavio.

5 APUOHJELMAN TOTEUTUS

Tämän kappaleen tarkoituksena on johdattaa lukija läpi apuohjelman luontiprosessin sen tärkeimmiltä osin. Tässä kappaleessa esitellään apuohjelman rakennusympäristö, käyttöliittymän ulkoasun luonti, apuohjelman taustalla pyörivä liiketoimintalogiikka ja muu apuohjelman toiminnan kannalta tärkeä materiaali. Kappaleessa esitellään koodiesimerkein ratkaisuja erilaisiin pulmatilanteisiin, joihin apuohjelman luonnin aikana törmättiin, ja harjoitetaan samalla ehkä hieman itsekritiikkiä käytettyjä ratkaisuja kohtaan.

5.1 Rakennusympäristö

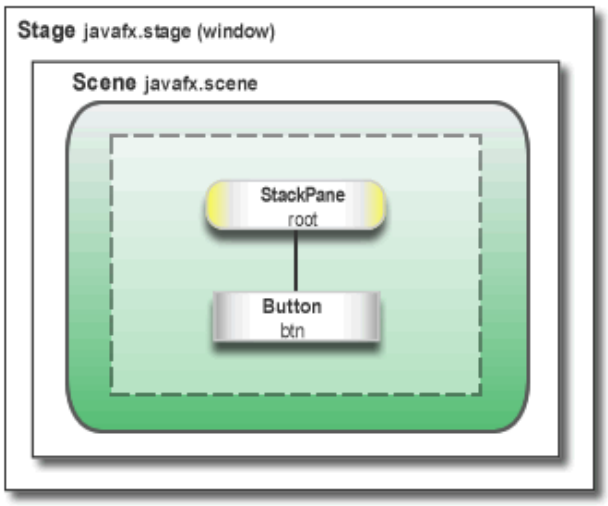
Käyttöliittymä rakennettiin JavaFX-kehystä hyväksikäyttäen, joka on Oraclen Javan pohjalta kehittämä kokoelma grafiikka- ja mediapaketteja. Työssä käytettiin JDK versiota 8 (kirjoitushetkellä 1.8.0_20) ja NetBeans IDE:tä (Integrated Development Environment) versiota 8.0.1. Käyttöjärjestelmäympäristö oli Linux.

5.2 Ulkoasu

Kuten edellisessä kappaleessa tuli esille, ulkoasu päätettiin luoda alkuperäisen sovelluspohjaisen käyttöliittymän kaltaiseksi.

JavaFX toimii hyvin samoin periaattein kuin edeltäjänsä Java Swing. Ohjelmakoodissa luodaan tarvittavat GUI komponentit **(Kuvio 10.)**, asetetaan niille halutut ominaisuudet **(Kuvio 11.)** ja lopuksi komponentit sijoitetaan näytettäväksi ulkoasuun **(Kuvio 12.** ja **Kuvio 13.)**. Lopputulos on nähtävissä kuvioista 14 ja 15.

Kaiken kaikkiaan JavaFX ulkoasu koostuu neljästä osasta: stage, scene, pane ja komponentit **(Kuvio 9.)**. Yksi stage (periaatteessa siis ikkuna) sisältää yhden scenen, joka vastaavasti voi sisältää yhden (tai useamman) panen, joka voi sisältää halutun määrän käytössä olevia komponentteja. /6/



Kuvio 9. JavaFX GUI-rakenne /6/.

```
/*****************************/
/*** COMPONENTS - LOGIN ***/
/*****************************/
        Label lblAppName = new Label(appName);
        Label lblUsername = new Label("Username:");
        Label lblPassword = new Label("Password:");
        Label lblNotificationsLog = new Label("WELCOME!");
        TextField tfUsername = new TextField();
        PasswordField pwdfPassword = new PasswordField();
        Button btnExit = new Button("  Exit  ");
        Button btnLogin = new Button("  LOGIN  ");
```

Kuvio 10. Login-ikkunan komponentit.

```
/*********************************/
/*** COMPONENT SETUPS - LOGIN ***/
/*********************************/
        lblAppName.setStyle("-fx-font-size:18;"
                + "-fx-font-weight:bold;"
                + "-fx-font-style:italic");
        lblNotificationsLog.setWrapText(true);
        lblNotificationsLog.setStyle("-fx-text-fill:green;"
                + "-fx-font-weight:bold");

        tfUsername.setTooltip(new Tooltip("Write your username here."));
        pwdfPassword.setTooltip(new Tooltip("Write your password here."));
```

Kuvio 11. Login-ikkunan komponenttien esiasetukset.

Komponenttien asetuksia ja toimintaa voidaan ohjelman suorituksen aikana muuttaa ja muokata vapaasti tarpeiden mukaan. Yllä olevan kuvion asetukset tulevat

käyttöön heti apuohjelman käynnistyessä. Huomaa myös CSS:ltä peritty tyylisyntaksi määriteltäessä tekstin tyyliä ja väriä. JavaFX tukee tyylisivujen käyttöä ja suoraan koodiin upotettujen tyyliasetusten lisäksi tyylit voidaan määritellä erillisessä tiedostossa.

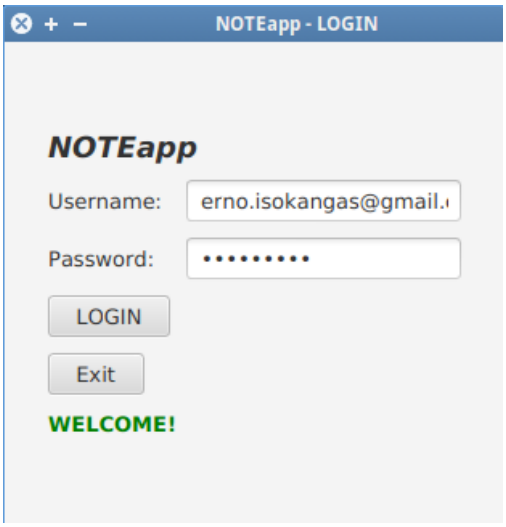
```
/************************/
/*** LAYOUT - LOGIN ***/
/************************/
        GridPane gridLogin = new GridPane();
        gridLogin.setAlignment(Pos.CENTER);
        gridLogin.setPadding(new Insets(inset,inset,inset,inset));
        gridLogin.setHgap(gap);
        gridLogin.setVgap(gap);

        // The order in which these components appear here is a must!
        // This affects the behaviour of pressing the Tab key from keyboard.
        gridLogin.add(lblAppName,0,0,2,1);
        gridLogin.add(lblUsername,0,1);
        gridLogin.add(lblPassword,0,2);
        gridLogin.add(lblNotificationsLog,0,5,2,1);
        gridLogin.add(tfUsername,1,1);
        gridLogin.add(pwdfPassword,1,2);
        gridLogin.add(btnLogin,0,3);
        gridLogin.add(btnExit,0,4);
```

Kuvio 12. Login-ikkunan komponenttien asettelu.

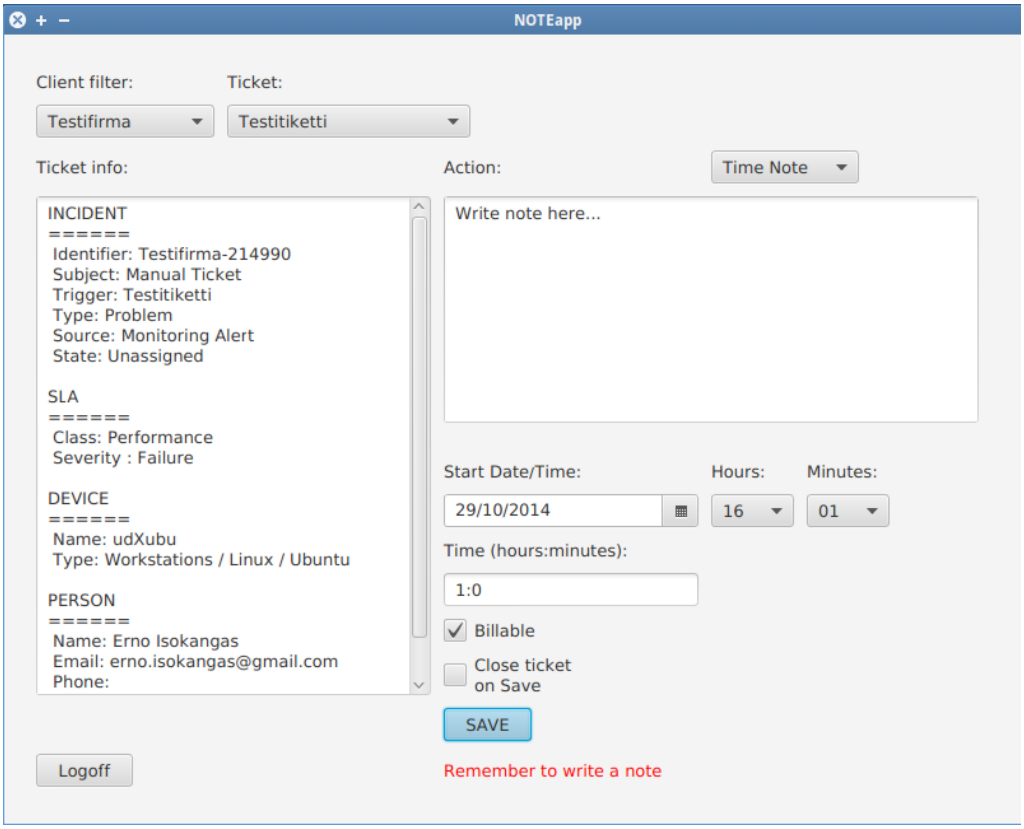
```
/*************************/
/*** STAGES & SCENES ***/
/*************************/
        Scene scenePrimary = new Scene(gridPrimary, priWinWidth, priWinHeight);
        stagePrimary.setTitle(appName);
        stagePrimary.setScene(scenePrimary);
        stagePrimary.hide();

        Scene sceneLogin = new Scene(gridLogin, logWinWidth, logWinHeight);
        stageLogin.setTitle(appName+" - LOGIN");
        stageLogin.setScene(sceneLogin);
        stageLogin.show();

        // Screw Windows and it's backslash. >_<
        // The StyleManager doesn't like backslashes so replace them.
        scenePrimary.getStylesheets().add(
                ("file:"+new File(appRoot).getParent()+"/CSS/Primary.css")
                        .replace("\\","/"));
        scenePrimary.getRoot().getStyleClass().add("main");
        sceneLogin.getStylesheets().add(
                ("file:"+new File(appRoot).getParent()+"/CSS/Login.css")
                        .replace("\\","/"));
        sceneLogin.getRoot().getStyleClass().add("main");
```

Kuvio 13. Stagen ja scenen asetukset ja asettelu, taustakuvan lisäys.

Alkuperäisen apuohjelman login-ikkunassa ja pääikkunassa on taustakuvana tilaajayrityksen logo. Koska yritys ei halua identiteettiään julki, taustakuvat on poistettu kuvakaappauksista.



Kuvio 14. Login-ikkuna.



Kuvio 15. Pääikkuna.

Pääikkunassa ikkunan sulkunäppäin (X-näppäin) on otettu ohjelmallisesti pois käytöstä. Tarkoituksena on ohjata käyttäjä sulkemaan ohjelma painamalla ensin vasemmassa alanurkassa olevaa Logoff-näppäintä ja palaamaan tätä kautta takaisin Login-ikkunaan, mistä käsin ohjelma voidaan sitten sulkea.

Työn tilaajan toiveena oli mahdollisuus kyetä hallitsemaan apuohjelman toimintaa näppäimistöpohjaisesti, mutta tämä tavoite saavutettiin vain osittain; hiiren käyttö on edelleen tarpeellista paikoitellen. Löydetyt ja osittain testatut ratkaisut ns. näppäimistötuntuman parantamiseen olivat suurimmalta osin melko vaivalloisia ja tarpeettoman monimutkaisia toteuttaa ja muutenkin työaikaa päätettiin käyttää pääasiassa toimintalogiikan rakentamiseen ja hiomiseen.

5.3 Toimintalogiikka

Apuohjelman ulkoasun ollessa suurin piirtein kunnossa, aloitettiin toimintalogiikan rakentaminen.

5.3.1 Tiedostojaottelu

Toimintalogiikan tukena löytyy joukko erinäisiä teksti- ja XML-tiedostoja, jotka ovat apuohjelman käytössä ns. ulkoisina resursseina (vastakohtana jar-paketin sisäiset resurssit). Ulkoisten resurssien etuna on mahdollisuus muokata näiden tiedostojen sisältöä tarvittaessa koskematta itse ohjelman jar-pakettiin. Tämä ratkaisu tekee apuohjelmasta joustavan ja ad hoc -tyyppiset toimintalogiikan muutokset ovat tällöin mahdollisia ilman ohjelman uudelleenrakentamista. Apuohjelman tiedostojaottelu rakentuu seuraavasti.

```
Parent folder                                                    (3)
   |- apuohjelma.jar
   |- apuohjelma.html
   |- apuohjelma.jnlp
   |- lib
      |- json-simple-1.1.1.jar
   |- URL
      |- authentication
      |- baseurl
      |- client
      |- ticket
```

```
|- XML
   |- closeTicket.xml
   |- getClients.xml
   |- getTickets.xml
   |- login.xml
   |- logoff.xml
   |- saveNote.xml
|- CSS
   |- Login.css
   |- Primary.css
   |- login-page-logo.png
```

Apuohjelma on mahdollista suorittaa kolmessa erilaisessa moodissa. Tarvittavat tiedostot Java luo automaattisesti. Ensimmäinen moodi, ”standalone”, tarkoittaa yksinkertaisesti ohjelman suorittamista ajamalla jar-paketin sisältö paikallisessa järjestelmässä. Toinen moodi, ”browser”, on periaatteessa sama kuin edellä, mutta ajettavaksi valitaan ohjelman html-tiedosto, joka suorittaa ohjelman verkkoselaimessa näytettynä. Kolmas ja viimeinen moodi, ”webstart”, mahdollistaa ohjelman suorittamisen ajamalla ohjelman jnlp-tiedosto palvelimelta käsin.

5.3.2 NoteApp-luokan muuttujat

Lähdekoodissa logiikan kulmakivenä toimii joukko NoteApp-luokan muuttujia **(Kuvio 16.)**, joita kutsutaan ja manipuloidaan tarvittaessa tiedon lähettämisestä palvelimelle aina käyttäjälle näytettävän tiedon esittämiseen käyttöliittymässä. Tärkeimpiä näistä muuttujista ovat sisään kirjauduttaessa saatu GUID-tunnus (Globally Unique Identifier), vastaanotettu tikettilista ja käyttöhetkellä valittuna olevan tiketin id-tunnus.

```
private String sessionGuid = "";
// Client keys and values could be placed in a Map, but because the keys
// already correspond the right client values when they come from
// the service, a Map is not that necessary to be had.
private List listClientKeys = null;
private List listClientValues = null;
// List of tickets as JSON objects.
private List listTickets = null;
// This string should contain the client's name value (unmodified).
private String selectedClient = "";
// This string should contain the ticket's incident combined ID.
private String selectedTicket = "";
// This string should contain the ticket's incident trigger
// (a.k.a IncidentDescription) (unmodified).
private String selectedTicketTrigger = "";
```

Kuvio 16. Apuohjelman tärkeitä muuttujia.

5.3.3 Sisäänkirjautumisprosessi

Seuraavissa kuvakaappauksissa esitellään pääkohdittain sisäänkirjautumisprosessi, joka noudattaa samoja toimintalogiikkaperiaatteita kuin apuohjelman muutkin toiminnot, jotka vaativat tietokantayhteyttä. Sisäänkirjautuminen vaatii käyttäjää antamaan ohjelmalle käyttäjätunnuksen ja salasanan. Näiden tietojen avulla voidaan luoda palvelimelle lähetettävä SOAP-viesti Javan SAAJ-työkaluja hyväksikäyttäen. **(Kuvio 17.)**.

```
public SOAPMessage login(String username, String password) throws
        SAXException, IOException, SOAPException {

    // Fetch SOAP skeleton for login.
    Document doc = docBuilder.parse(new File(appRoot).getParent()+"/XML/login.xml");
    DOMSource domSource = new DOMSource(doc);

    SOAPMessage loginMsg = messageFactory.createMessage();
    SOAPPart soapPart = loginMsg.getSOAPPart();
    soapPart.setContent(domSource);

    // Place login credentials into the message.
    NodeList uname = loginMsg.getSOAPBody().getElementsByTagName("username");
    uname.item(0).setTextContent(username);
    NodeList pwd = loginMsg.getSOAPBody().getElementsByTagName("password");
    pwd.item(0).setTextContent(password);

    loginMsg.saveChanges();
    /*
    System.out.println("");
    loginMsg.writeTo(System.out);
    System.out.println("");
    */
    return loginMsg;
}
```

Kuvio 17. Login-viestin luonti.

Viestiä luotaessa ohjelma lukee ulkoisista resursseista vastaavan XML-pohjaisen viestirungon ja liittää tähän viestiin tarvittavat parametrit. Ennen viestin lähetystä ohjelman täytyy totta kai myös tietää, minne viesti kuuluu lähettää. Ulkoisista resursseista käydään jälleen lukemassa tarvittavat osoitteet **(Kuvio 18.)**.

```
BufferedReader br;
// Fetch server address.
br = new BufferedReader(new FileReader(
        new File(appRoot).getParent().replace("%20"," ")+"/URL/baseurl"));
String baseurl = br.readLine();

// Fetch authentication service name.
br = new BufferedReader(new FileReader(
        new File(appRoot).getParent().replace("%20"," ")+"/URL/authentication"));
String authService = br.readLine();
```

Kuvio 18. Osoitteiden haku.

Viestin ollessa kunnossa ja osoitteet tiedossa, ohjelma lähettää SAAJin avulla viestin palvelimelle ja vastaanottaa palvelimelta SOAP-vastauksen ja käsittelee sen. Käsittelyvaiheessa tarvitaan json-simple-1.1.1 -kirjastoa **(Kuvio 19.)**.

```
if(tfUsername.getText().isEmpty() || pwdfPassword.getText().isEmpty()) {
    lblNotificationsLog.setText("Missing login information!");
}
else {
    SOAPMessage loginRes = null;
    SOAPMessage getClientsRes = null;
    SOAPMessage getTicketsRes = null;
    boolean loginOk = false;
    boolean clientsOk = false;
    boolean ticketsOk = false;

    // Send login message.
    try {
        CreateSOAPMessage createMessage = new CreateSOAPMessage();
        SOAPMessage loginMsg = createMessage.login(
                tfUsername.getText(), pwdfPassword.getText());
        loginMsg.getMimeHeaders().removeHeader("SOAPAction");

        System.out.println("CALL SERVICE...login");
        loginRes = connection.call(loginMsg, baseurl+authService);

        JSONObject loginJSON = extractJSON(loginRes);

        if((boolean)loginJSON.get("Success")) {
            System.out.println("...OK!");
            sessionGuid = (String)loginJSON.get("SessionGuid");
            loginOk = true;
        }
        else {
            String error = (String)loginJSON.get("Error");
            System.out.println(error);
            lblNotificationsLog.setText(error);
        }
    }
```

Kuvio 19. Login-viestin lähetys ja prosessointi.

SOAP-vastauksen käsittely tapahtuu lähettämällä vastaus metodille, joka muuttaa koko viestin ensin tekstijonoksi ja erottaa tästä tekstistä sitten JSON-osuuden **(Kuvio 20.)**. Palautettu JSON kantaa mukanaan ”Success”-nimistä muuttujaa, joka kertoo, onko SOAP-viestillä toteutettu toimenpide onnistunut vai ei. Jos sisäänkirjautuminen onnistui, JSON-tekstijonosta poimitaan käyttäjän GUID-tunnus, jota tullaan tästä lähtien käyttämään kaikissa palvelimelle lähetettävissä SOAP-viesteissä. Jos toiminto epäonnistui, ”Error”-kenttä sisältää syyn tähän.

```
// Turn SOAP message into a string.
private String soapToString(SOAPMessage soapMsg) throws
        SOAPException, IOException {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    if(soapMsg != null) {
        soapMsg.writeTo(out);
        return out.toString();
    }
    else {
        return "";
    }
}

// Extract serialized JSON from SOAP message.
private JSONObject extractJSON(SOAPMessage soapMsg) throws
        SOAPException, IOException, ParseException,
        StringIndexOutOfBoundsException {
    // The original SOAP response cannot be worked on with SAAJ tools as the
    // message starts with an html tag. The message is first turned into
    // a string and the JSON part is then extracted.
    // The JSON "{" will always be the first one to appear and the string
    // always ends just before "</body>" where the last "}" is included.
    String sSoapMsg = soapToString(soapMsg);
    String serializedJSON =
            sSoapMsg.substring(sSoapMsg.indexOf("{"), sSoapMsg.indexOf("</body>"));
    JSONParser parser = new JSONParser();
    //System.out.println("\nSerializedJSON: "+serializedJSON+"\n");
    Object obj = parser.parse(serializedJSON);

    return (JSONObject)obj;
}
```

Kuvio 20. Serialisoidun JSON-tekstijonon haku SOAP-viestistä.

Tässä vaiheessa tarkkaavaisempi lukija on saattanut huomata, että en käytä SOAP-vastauksen käsittelyssä SAAJin tarjoamia työkaluja. Miksi? Jotta vastausta voitaisiin käsitellä SAAJin kautta, tulisi viesti alkaa ns. XML-julistuksella, jotta SAAJ ymmärtäisi viestin olevan SOAP-viesti. Mutta viesti alkaakin HTML-merkkikielisellä osiolla ja XML-julistus tulee vasta paljon myöhemmin. Viesti täytyy täten käsitellä jollain muulla tavalla.

Jos palvelimelta saapuva vastaus ei sisällä odotetun kaltaista tietoa tai viestiä ei ole virhetapahtuman johdosta lähetetty lainkaan, tapaus täytyy käsitellä jotenkin **(Kuvio 21.)**. Erilaisia virhetilanteita ovat mm. väärä osoite ja virheellisesti tai puutteellisesti rakennettu SOAP-viesti. Itse JSON-tekstijono sisältää oman virheilmoituksena (sisään kirjauduttaessa esim. virheellinen käyttäjätunnus tai salasana), mutta tämä käsitellään lähdekoodissa jo aikaisemmin, kuten jo mainittua.

```
catch (SOAPException | SAXException | IOException |
        ParserConfigurationException | ParseException |
        StringIndexOutOfBoundsException ex) {
    System.out.println("LOGIN ERROR: "+ex.getMessage());
    lblNotificationsLog.setText(ex.getMessage());
    // In case there is no serialized JSON, try get SOAPFault.
    try {
        String fault = loginRes.getSOAPBody().getFault().getFaultString();
        System.out.println("LOGIN FAULT: "+fault);
        lblNotificationsLog.setText("LOGIN FAULT:\n"+fault);
    }
    catch(SOAPException ex2) {
        // Empty catch.
    }
}
```

Kuvio 21. Login-viestin lähetyksen virheenkäsittely.

Koko edellä esitetty toimintaketju käydään läpi aina, kun ohjelma hakee tietoa palvelimelta. Toteutuksessa on pyritty ottamaan huomioon kaikki mahdolliset virhetilanteet ja virheet näytetään apuohjelman käyttöliittymässä. Vaihtoehtoisesti ohjelma voidaan käynnistää komentotulkista, jolloin on mahdollista saada tarkentavaa tietoa, milloin jokin tietty virhe on tarkalleen tapahtunut, jos virheilmoitus ei selkeästi identifioi ongelman laatua ja tapahtumahetkeä.

Vaikka palvelinkutsuihin käytetyt toimintaperiaatteet ovatkin tapauksesta riippumatta hyvin identtiset, pieniä eroavaisuuksia löytyy. Palvelimelta vastaanotettu data on aina hieman erilaista ja tämä vaatii vaihtelevan määrän eri käsittelytapoja. Jotta tarpeettomalta saman jo kirjoitetun koodin toistolta oltaisiin vältytty, ajatuksena oli saada kaikki toiminnallisuus pilkottua kouralliseen toisiaan kutsuvia metodeja, mutta tämä ei valitettavasti ollut täysin mahdollista. Lopullisessa toteutuksessa päästiin kuitenkin varsin tyydyttävään lopputulokseen.

6 TESTAUS

Testauksen tarkoituksena on rakentaa suunnitelmallinen ympäristö sovelluksessa olevien mahdollisten virheiden löytämiseksi. Suunnitelmallisuus on tärkeää, mutta toisinaan virheitä voidaan löytää myös täysin sattumanvaraisesti. Testaus on ainoa keino, jolla ohjelman toiminnasta voidaan löytää virheitä. Näitä virheitä voidaan sitten verrata määriteltyihin vaatimuksiin ohjelman toimintaperiaatteista (spesifikaatio) ja todeta, että ohjelma ei toimi odotetusti. /2, 205/

Testaus toteutettiin Linux- ja Windows-ympäristössä. Suurin osa testauksesta tapahtui aina ohjelman kirjoitushetkellä Linux-ympäristössä. Ohjelmaa testattiin käyttämällä sitä tarkoitetulla tavalla odottaen sen toimivan halutun kaltaisesti. Virhetilanteet korjattiin aina niiden esiin tullessa ja tämän jälkeen virheeseen johtaneet toimenpiteet tehtiin uudelleen ja katsottiin, toistuiko virhe. Jos virhettä ei tullut, korjaukset olivat onnistuneet. Virheitä etsittiin myös tarkoitushakuisesti käyttämällä ohjelmaa tarkoituksella väärin ja havainnoimalla, käsittelikö ohjelma virhetilanteet odotetulla tavalla.

Erilaisia virhetilanteita tuli projektin aikana vastaan useita, joista tässä seuraavaksi muutama tärkeämpi.

Kuten edellisessä kappaleessa tuli kerrottua, palvelimelta saatua SOAP-vastausta ei voitu lukea käyttämällä Javan SAAJ APIa. Tämä asia ei kuitenkaan ollut selvä silloin, kun ongelma ensimmäisen kerran nosti päätään. Veikkaan, että tämän ongelman kanssa painimisessa meni vajaa viikon päivät, ennen kuin nykyiseen ratkaisuun päädyttiin.

Kun ohjelmaa testattiin Windows-ympäristössä, huomattiin taustakuvien puuttuvan. Linuxissa ongelmaa ei esiintynyt. Ongelma korjattiin korvaamalla ohjelmakoodissa Windowsin tiedostopoluissa käyttämä kenoviiva ("\") kauttaviivalla ("/").

Myöhemmissä testauksissa huomattiin, että joidenkin tikettien tiedot eivät näkyneet tiketti-info -kentässä lainkaan. Syynä ongelmaan oli yritys käsitellä sellaisia

JSON-objektista saatuja muuttujia, jotka omasivat arvon ”null”. Tästä viisastuneena tiettyjä muuttujia luettaessa testataan, sisältääkö muuttuja mahdollisesti arvon ”null”, minkä perusteella muuttuja voidaan jättää käsittelemättä.

Koska JSON-tekstijono tulee SOAP-vastauksessa HTML-osion sisällä HTML-enkoodattuna, skandinaaviset aakkoset eivät näkyneet vetovalikoissa ja tiketti-info -kentässä oikein (ä => ”ä” jne.). Koska oikeanlaista dekoodaustapaa ei löytynyt pitkällisten etsintöjen ja testauksien jälkeen, päädyttiin luomaan Hash-taulukko, joka sisälsi HTML-enkoodatun merkkijonon ja sitä vastaavan skandinaavisen aakkosen. Taulukkoa käytettiin tämän jälkeen korvaamaan tekstistä HTML-enkoodatut osiot oikeilla aakkosilla.

7 YHTEENVETO

Apuohjelman toteutus oli mielestäni onnistunut ja tämän suuntaista palautetta sain myös työn tilaajalta. Vaatimusmäärittely saatiin toteutettua vaadittavilta osin ja hieman ylimääräistäkin saatiin aikaiseksi. Käytännön osuudelle antamani aikataulutuskin, yksi kuukausi, piti hyvin paikkansa, mistä olin tyytyväinen.

Projektin alku oli totta kai kankea kuten aina, useampaan kertaan jouduin olemaan eri tahoihin yhteydessä ratkaisuja hakiessani, erityisesti ohjelmointiin liittyvissä ongelmatapauksissa, mutta loppu olikin sitten melko sujuvaa suurimpien esteiden ollessa poissa tieltä. Käyttöliittymän luonti Javan työkaluilla oli tuttua puuhaa, vaikka en sitä koskaan ollutkaan toteuttanut JavaFX:llä. SOAP oli protokollana myös tuttu, mutta Javan SAAJ APIn käyttö tarvitsi hieman opettelua alkuvaiheessa. JSON-formaatin käsittely oli myöskin osittain tuttua.

Tämän yhteenvedon kirjoitushetkellä en osaa vielä sanoa, missä mittakaavassa apuohjelma on otettu käyttöön, jos on otettu. Ohjelma oli julkaisuhetkellä täysin toimiva ja koska kyseessä oleva tilaajayritys ei ole työntekijämäärältään suuren suuri, käyttöönotto voidaan toteuttaa hyvin nopeasti ja helposti.

Kaiken kaikkiaan projekti oli erittäin opettavainen ja valaiseva kokemus. Pääsin mielestäni syventämään tietämystäni juuri sellaisista asioista, joista olin kiinnostunut ja yhteistyö oikean työelämän yrityksen kanssa oli mielenkiintoista.

LÄHTEET

/1/ McLaughlin, B., O'Reilly and Associates Inc. 2001. Java & XML. Second Edition. Suomennettu painos. Helsinki. Talentum. 2002.

/2/ Haikala, I. & Mikkonen T. 2011. Ohjelmistotuotannon käytännöt. 12. uud. painos. Helsinki. Talentum.

/3/ X System Reference. System Guide. V2013 R3. Joulukuu 2013. A guide to the deployment and usage of the X IT Management System.

/4/ Introducing JSON. Viitattu 18.9.2014.
http://json.org

/5/ JavaFX Overview. Viitattu 27.9.2014.
http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

/6/ Hello World, JavaFX Style. Viitattu 31.10.2014.
http://docs.oracle.com/javafx/2/get_started/hello_world.htm

/7/ JavaFX Frequently Asked Questions. Viitattu 27.9.2014.
http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html

/8/ The Java EE 5 Tutorial. Chapter 19. SOAP with Attachments API for Java. Viitattu 27.10.2014.
http://docs.oracle.com/javaee/5/tutorial/doc/bnbhf.html

/9/ Extensible Markup Language (XML). Introduction. Viitattu 27.10.2014.
http://www.w3.org/XML/

/10/ Quality Function Deployment (Draft). Viitattu 29.10.2014.
http://www.di.ufpe.br/~req_case/Seminarios/Prioridades/qfd.pdf