Tampereen Ammattikorkeakoulu
Tietototekniikan koulutusohjelma
Ohjelmistotekniikka
Matti Lehto

Opinnäytetyö

SMSGW-järjestelmä

Työn ohjaaja
Lehtori Jari Mikkolainen
Työn teettäjä
Tampereen ammattikorkeakoulu
Tampere 06/2010

Tampereen Ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tekijä	Matti Lehto
Työn nimi	SMSGW-järjestelmä
Sivumäärä	58 sivua
Valmistumisaika	06/2010
Työnohjaaja	Lehtori Jari Mikkolainen
Työn teettäjä	Tampereen ammattikorkeakoulu
Valvojana	DI Petteri Jekunen

TIIVISTELMÄ

Tutkintotyön aiheena oli suunnitella ja toteuttaa SMS-viestien lähettämiseen ja vastaanottamiseen tarkoitettu järjestelmä Tampereen ammattikorkeakoulun tietokonekeskukselle. Yksi järjestelmän käyttökohteista oli mahdollistaa Nagios-verkonvalvontasovelluksen vikatilannetietojen lähettämisen ja kuittaamisen SMS-viesteillä.

Järjestelmän kehityksessä käytettiin Java-ohjelmointikieltä, Java EE -sovelluskehystä ja AJAX- ja XML-RPC-tekniikoita. Järjestelmän SMS-yhdyskäytävä toteutettiin käyttäen matkapuhelinta ja Gammu-projektin tarjoamia työkaluja.

Tuloksena syntyi pohja, jolta voidaan lähteä toteuttamaan järjestelmään uusia ominaisuuksia ja laajentamaan jo toteutettuja. Työn aikana saatiin arvokasta kokemusta järjestelmän toteuttamisessa käytetyistä monista tekniikoista.

Avainsanat SMS, Java, Gammu, yhdyskäytävä

TAMK University of Applied Sciences
Programme in information technology
Software engineering

Writer	Matti Lehto
Thesis	SMSGW-system
Pages	58 pages
Graduation time	06/2010
Thesis supervisor	Lecturer Jari Mikkolainen
Commissioning company	TAMK, MSc Petteri Jekunen

ABSTRACT

This thesis covers the process of designing and implementing a system for the use of TAMK's Computer Center. The implemented system is used to send and receive SMS messages. One of the aims in developing the system was to enable Nagios computer system and networking monitoring software to send error reports and to acknowledge those error reports using SMS.

Java-programming language, Java EE- platform and AJAX and XML-RPC techniques were used to develop the system. SMS gateway was implemented by using a mobilephone and tools provided by the Gammu project.

As a result a foundation was created, from which further development of the system could continue. During development valuable experience was accumulated of the multiple techniques that were used.

Keywords SMS, Java, Gammu, gateway

TERMIT JA LYHENTEET

Asennuskuvain	*Deployment Descriptor*
AJAX	*Asynchronous Javascript And XML* on joukko toisiinsa liittyviä tekniikoita, joita käytetään luomaan vuorovaikutteisia web-sovelluksia.
Gammu	Sovellus joka sisältää sovelluksia, komentosarjoja ja ajureita matkapuhelimien ja vastaavien laitteiden toimintojen hallitsemiseen.
Java Applet	Asiakaskoneessa web-selaimen yhteydessä suoritettava Java-ohjelma.
JDBC	*Java Database Connectivity* on Java ohjelmointikielen rajapinta, joka tarjoaa tietojen kyselyssä ja päivittämisessä tarvittavat metodit.
Komentotulkki	Tekstipohjainen tietokoneohjelma, jolla ohjataan käyttöjärjestelmää. Komentulkissa ohjelmien käynnistys ja muut komennot suoritetaan kirjoittamalla ne komentoriville.
LDAP	*Lightweight Directory Access Protocol* on hakemistopalveluiden käyttämiseen tarkoitettu protokolla.
Nagios	Avoimen lähdekoodin verkonvalvontasovellus.
Netcat	TCP- ja UDP-yhteyksien luomiseen käytetty työkalu.
Nimetty putki	Tiedostojärjestelmään luotu pysyvä erikoistiedosto, johon voivat eri aikoihin tai samanaikaisesti lukea ja kirjoittaa eri prosessit.
OpenSSH	Kokoelma sovelluksia, jotka tarjoavat salatun kommunikointi tavan tietoverkoissa, käyttäen SSH-protokollaa.

RPC	*Remote Procedure Call* teknologia, jolla toisessa tietokoneessa toimivia palveluja voidaan kutsua Internet-verkon yli.
Servlet	Servlet on Java-ohjelmointikielellä toteutettu luokka, joka prosessoi HTTP-pyyntöjä ja generoi niihin vastauksen.
Shell Script	Komentosarja eli skripti, joka on kirjoitettu käyttöjärjestelmän komentotulkille.
Socket	Mahdollistaa lukemisen ja kirjoittamisen tiettyyn määriteltyyn verkkoyhteyteen.
SSH-protokolla	*Secure Shell* on tietoverkoissa käytetty protokolla liikenteen salaamiseen.
stdin	*Standard input stream* Standardi syötevirta.
UDH	*User Data Header* tapa lähettää moniosaisia SMS-viestejä, jossa viesteihin lisätään tunniste, joka rajoittaa yksittäisten viestien sisällön 153 merkkiin, lopun viestin koostuessa tunnisteesta.
XML	*Extensible Markup Language* on kuvauskieli, jossa tiedon sisältö kuvataan tiedon seassa.
XML-RPC	RPC kutsu käyttäen HTTP-protokollaa. XML-RPC-viesti välitetään HTTP-POST-muotoisena viestinä, jonka rakenne on XML:ää.

Sisällysluettelo

1 Johdanto

Suorittaessani TAMKin tietokonekeskuksella opintoihini sisältyvää työharjoittelua, minulle tarjoutui mahdollisuus tehdä opinnäytetyönäni SMS-viestien lähettämiseen ja vastaanottamiseen tarkoitettu järjestelmä. Järjestelmä sisältää SMS-viestien lähettämisen ja vastaanottamisen toteuttavan palvelinsovelluksen sekä palvelinsovelluksen käyttämiseen toteutetun web-käyttöliittymän tarjoavan asiakassovelluksen.

Yhtenä palvelinsovelluksen käyttötarkoituksena oli mahdollistaa Nagios-verkonvalvontasovellukselta SMS-viestien lähettäminen ja vastaanotettujen viestien palauttaminen Nagios-palvelimelle. TAMKin tietokonekeskus käyttää Nagios-verkonvalvontasovellusta TAMKin verkon ja sen palveluiden valvontaan. Palvelinsovellusta käyttämällä voidaan valvottavassa verkossa ilmenevästä vikatilanteesta lähettää ylläpitäjälle ilmoitus SMS-viestillä sekä ylläpitäjä voi kuitata vikatilanteen lähettämällä SMS-viestin takaisin palvelinsovellukselle.

Työn laajuuden vuoksi työssä keskitytään palvelinsovelluksen suunnitteluun ja toteutukseen sekä sen käyttöön Nagios-verkonvalvontasovelluksen kanssa. Asiakassovelluksesta esitellään toteutuksessa käytetyt tekniikat ja sen toiminta yleisellä tasolla esimerkkien kautta.

2 SMSGW-järjestelmän suunnittelu

Järjestelmän suunnittelu aloitettiin palaverilla, jossa määriteltiin tietokonekeskuksen asettamat vaatimukset työlle. Järjestelmä jaettiin kahteen erilliseen työkokonaisuuteen, palvelin- ja asiakassovellukseen. Seuraavissa luvuissa käsitellään palvelin- ja asiakassovellukselle määritetyt asiakasvaatimukset.

2.1 Järjestelmän asiakasvaatimukset

Suunnittelupalaverissa palvelinsovellukselle määritettiin seuraavanlaiset toiminnalliset ja ei-toiminnalliset vaatimukset:

- SMS-viestien lähettäminen ja vastaanottaminen käyttäen GSM-modeemia.
- Nagios-verkonvalvontasovelluksen vikatilannetiedon lähettäminen SMS-viestillä päivystäjälle. Päivystäjä pystyy kuittaamaan vikatilanteen lähettämällä saapuneen SMS-viestin takaisin sen lähettäneelle palvelinsovellukselle.
- XML-RPC-rajapinnan tarjoaminen asiakassovelluksen käytettäväksi.
- Java-ohjelmointikielen käyttäminen sovelluksen toteuttamisessa.
- Sovellus suoritetaan Linux-sovellusalustalla.
- GSM-modeemina käytetään matkapuhelinta.

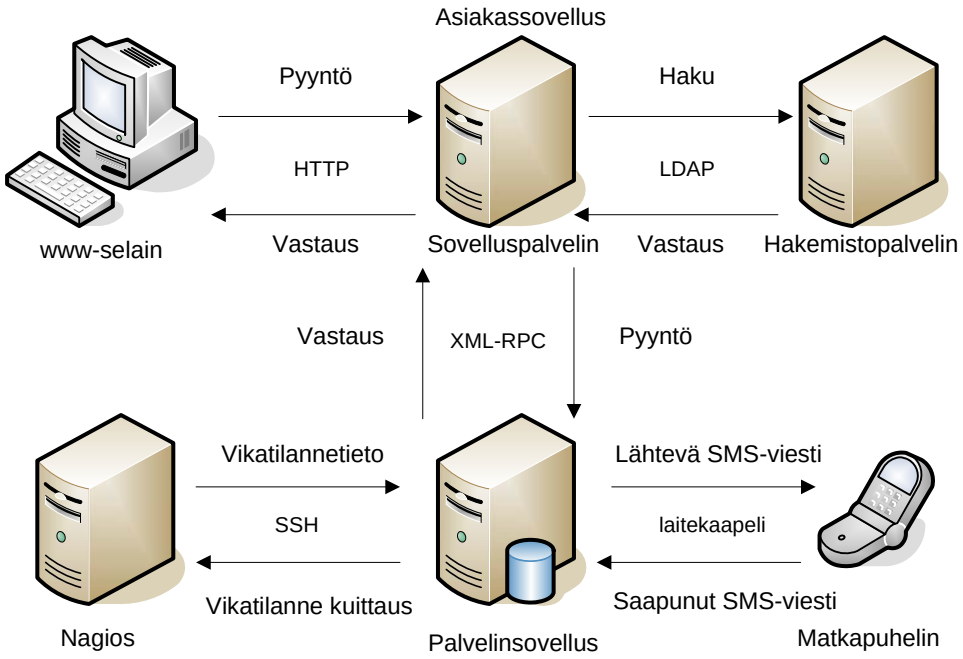
Muutoin palvelinsovelluksen suunnittelussa oli vapaat kädet teknologioiden ja laitteiden osalta sillä poikkeuksella, että laitteisto olisi saatavilla TAMKin tietokonekeskukselta.

Asiakassovellukselle vuorostaan määriteltiin seuraavat toiminnalliset vaatimukset ja ei-toiminnalliset vaatimukset:

- Sovellus tarjoaa Java EE -spesifikaation mukaisen web-käyttöliittymän.

- Kontaktien hakeminen hakemistopalvelusta ja käyttäjäkohtaisten kontaktilistojen luominen sekä SMS-viestien lähettäminen listoista valituille kontakteille.
- Palvelinsovelluksella lähetettyjen ja palvelinsovellukselle saapuneiden SMS-viestien tarkastelu.
- XML-RPC-rajapinnan kautta kommunikointi palvelinsovelluksen kanssa.
- Palvelinsovelluksen tietokannassa säilytettyjen tietojen näyttäminen sovelluksessa.
- TAMKin hakemistopalvelun käyttö LDAP-protokollalla.
- Käyttäjän tunnistaminen käyttäen Sun Java System Access Manageria.

Järjestelmälle asetettujen vaatimusten pohjalta siirryttiin palvelin- ja asiakasovelluksen suunnitteluun. Suunnittelun pohjaksi laadittiin järjestelmän yleiskuva, joka on esitetty kuvassa 1.



Kuva 1: Yleiskuva SMSGW-järjestelmästä.

2.2 Palvelinsovelluksen ominaisuudet

Palvelinsovelluksen SMS-yhdyskäytäväksi valittiin esitutkimuksen ja testauksen perusteella Gammu-projekti. Kannel-yhdyskäytävää harkittiin myös toisena vaihtoehtona, sen tarjotessa samankaltaiset ominaisuudet kuin Gammu-projekti. Ratkaisevina tekijöinä valinnassa olivat Gammu-projektin tukemien laitteiden suurempi lukumäärä ja projektin suurempi päivitystiheys. Gammu-projektin sovelluksia käyttämällä voidaan antaa käskyjä matkapuhelimelle, joka on liitetty laitekaapelilla palvelinsovellukseen.

Palvelinsovelluksen tarve palvella useita asiakkaita samanaikaisesti oli tärkeä suunnittelussa huomioitava ominaisuus. Java-ohjelmointikieli tarjoaa valmiin socket-pohjaisen palvelimen Nagios-palvelimelta tulevien yhteydenottojen kuuntelemiseen. Käyttämällä säikeitä socket-pohjaisessa palvelimessa mahdollistetaan useiden käyttäjien palveleminen samanaikaisesti. Suojattu yhteys Nagios-palvelimen ja palvelinsovelluksen välille voidaan helposti toteuttaa käyttämällä SSH-protokollan tarjoavaa OpenSSH-työkalua. Palvelinsovelluksen ja asiakassovelluksen välisen XML-RPC-protokollaa käyttävän yhteyden toteuttamiseksi valittiin Java-ohjelmointikielellä toteutettu Apache XML-RCP -kirjasto, jonka palvelinkirjastoa käytetään toteuttamaan palvelinsovelluksessa XML-RPC-palvelin.

SMS-viestien ja muiden tietojen säilömiseksi, palvelinsovellukselle sijoitetaan paikallinen tietokanta, joka toimii myös rajapintana palvelinsovelluksessa toteutettujen palvelimien ja Gammu-projektin välillä.

2.3 Asiakassovelluksen ominaisuudet

Asiakassovelluksen suunnittelussa tukeuduttiin Java EE:ssä yleisesti käytettyihin web-sovellus ratkaisuihin. Web-sovellus toteutetaan käyttäen JavaServer Faces -sovelluskehystä yhdessä JavaServer Pages -sivujen kanssa. Web-sovelluksen käyttökokemuksen parantamiseksi JavaServer Facesin kanssa käytetään AJAX-tekniikoita käyttävää RichFaces-käyttöliittymäkomponenttikirjastoa. Kuten

palvelinsovelluksessa, asiakassovelluksen tulee pystyä palvelemaan montaa käyttäjää samanaikaisesti.

Yhteys hakemistopalvelimelle asiakassovelluksesta toteutetaan käyttäen Java-ohjelmointikielen tarjoamaa JNDI-rajapintaa, jota käyttämällä voidaan suorittaa LDAP-protokollalla hakuja hakemistopalvelimeen. XML-RPC yhteys palvelinsovellukseen toteutetaan käyttämällä Apache XML-RPC -kirjaston asiakaskirjastoa.

3 Järjestelmässä käytetyt tekniikat

Järjestelmän toteuttamisessa käytettiin useita eri tekniikoita. Tässä luvussa esitellään niistä tärkeimmät, asiakassovelluksessa käytetyt web-tekniikat käydään läpi omassa luvussaan.

3.1 Socket

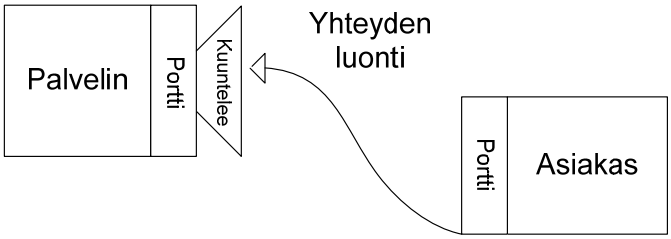
Socket on päätepiste kahdensuuntaisessa prosessien välisessä paikallisessa tai verkon ylitse tapahtuvassa kommunikaatiossa. /1/ Socketit voidaan jakaa kahteen ryhmään, kuunteleviin (palvelin) ja lähettäviin (asiakas). Socket-yhteyden toteuttamisessa asiakkaan ja palvelimen väliseen kommunikointiin voidaan käyttää erilaisia protokollia, seuraavaksi esitellään kaksi yleisesti käytössä olevaa protokollaa: TCP ja UDP.

TCP-protokolla tarjoaa luotettavan tavan lähettäjän ja vastaanottajan väliseen kommunikaatioon. Jotta lähettäjän ja vastaanottajan välillä voidaan lähettää tietoa, on ensiksi avattava yhteys lähettäjän ja vastaanottajan välille käyttäen kolmitiekättelyä. Yhteyden luomisen jälkeen aloitetaan tiedonsiirto, jonka aikana käytetään useita eri mekanismeja siirrettävän tiedon eheyden varmistamiseksi. Nämä mekanismit huolehtivat pakettien oikeasta järjestyksestä, tarkistussummien käytöstä virheiden tarkistuksessa sekä tunnistimien ja ajastimien käytöstä hukatuille paketeille ja yhteydessä esiintyvälle viiveelle. Tiedonsiirron jälkeen yhteys suljetaan käyttäen modifioitua kolmitiekättelyä. /2/

UDP-protokolla eroaa TCP:stä siinä, että tiedonsiirrossa käytetään yhteydetöntä yhteyttä, eikä tiedon eheyttä tarkisteta. UDP tarjoaa epäluotettavan tavan siirtää tietoa lähettäjän ja vastaanottajan välillä, jota käytetään esimerkiksi DNS-pyyntöjen ja reaaliaikaisen videon ja äänen lähettämiseen. /2/

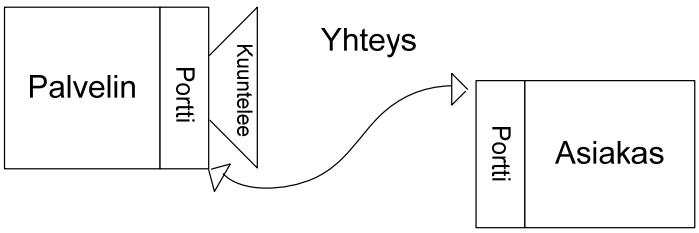
Seuraavaksi on esitetty esimerkki socketin käyttämisestä asiakkaan ja palvelimen välillä käyttäen TCP-protokollaa.

Palvelimella on socket, joka on sidottu tiettyyn porttinumeroon. Palvelin kuuntelee porttiin sidottua socketia, asiakkaan yhteydenottoa odottaen. Asiakas tietää tietokoneen IP-osoitteen, jossa palvelin on käynnissä ja myös palvelimen kuunteleman portin numeron. Asiakas sitoo myös itsensä paikalliseen porttiin, jota se käyttää yhteyden aikana ja yrittää yhdistää itsensä palvelimen kuuntelemaan porttiin (kuva 2). /1/



Kuva 2. Yhteyden luonti asiakkaan ja palvelimen välillä.

Hyväksyessään asiakkaan yhteydenoton palvelin luo uuden socketin. Luotu socket on sidottu palvelimen kuuntelemaan porttiin ja socketin etäpäätepiste asetetaan asiakkaan käyttämään IP-osoitteeseen ja porttiin. Palvelin käyttää luotua sockettia yhteyden ottaneen asiakkaan palvelemiseen, jotta se voi jatkaa asiakkaiden yhteydenottojen kuuntelemista samanaikaisesti. Asiakas on nyt yhdistetty palvelimeen ja socket on onnistuneesti luotu (kuva 3). Asiakas ja palvelin voivat nyt kommunikoida kirjoittamalla ja lukemalla sockettiensa kautta. /1/



Kuva 3. Asiakas ja palvelin on yhdistetty socketilla.

Java-ohjelmointikieli tarjoaa valmiita luokkia TCP- tai UDP-protokollaa käyttävän socket-palvelimen toteuttamiseksi. Järjestelmän yhteyspalvelimen socket-palvelimessa käytetään TCP-protokollaa sen tarjoaman luotettavuuden takia, myös järjestelmässä käytettävän XML-RPC-protokollan käyttämä HTTP-protokolla käyttää TCP:tä.

3.2 XML-RPC

XML-RPC on etäproseduurikutsu-protokolla, joka toimii internetin yli. XML-RPC:n kautta lähetettävä viesti on POST-tyyppinen HTTP-pyyntö, jonka runko on XML-muotoinen. Kutsu suoritetaan palvelimella ja vastaus lähetetään XML-muotoisena HTTP-viestinä kutsun suorittaneelle asiakkaalle. Proseduurikutsussa lähetettävä ja palautettava tieto voi olla muodoltaan skalaareja, numeroita, merkkijonoja sekä monimutkaisia tieto- ja listarakenteita. /3/

XML-RPC-asiakas- ja palvelinsovelluskirjastojen toteutuksia on tehty lukuisille ohjelmointikielille ja teknologioille. Järjestelmässä käytettiin Apache XML-RPC:n tarjoamia asiakas- ja palvelinsovelluskirjastoja, jotka on toteutettu Java-ohjelmointikielellä.

3.3 Observer-suunnitelumalli

Observer-suunnittelumalli perustuu kahdenlaisiin objekteihin, kohteisiin ja tarkkailijoihin. Tarkkailijat rekisteröityvät kohteisiin, jolloin kohteet voivat

ilmoittaa tilansa muutoksesta kaikille rekisteröityneille tarkkailijoille. Kohteet voivat myös irrottautua tarkkailijoistaan. Yleensä rekisteröityminen, ilmoitus ja irrottautuminen tapahtuu metodikutsuilla. Observer-suunnittelumalli soveltuu käytettäväksi tilanteissa, joissa toisistaan riippuvat oliot halutaan toteuttaa mahdollisimman riippumattomasti. /4/

3.4 Singleton-suunnittelumalli

Käyttämällä Singleton-suunnittelumallia varmistetaan, että objektista luodaan ainoastaan yksi instanssi. Jos useamman objektin täytyy käyttää singleton-luokkaa, objektit jakavat saman singleton-luokan instanssin. /5/

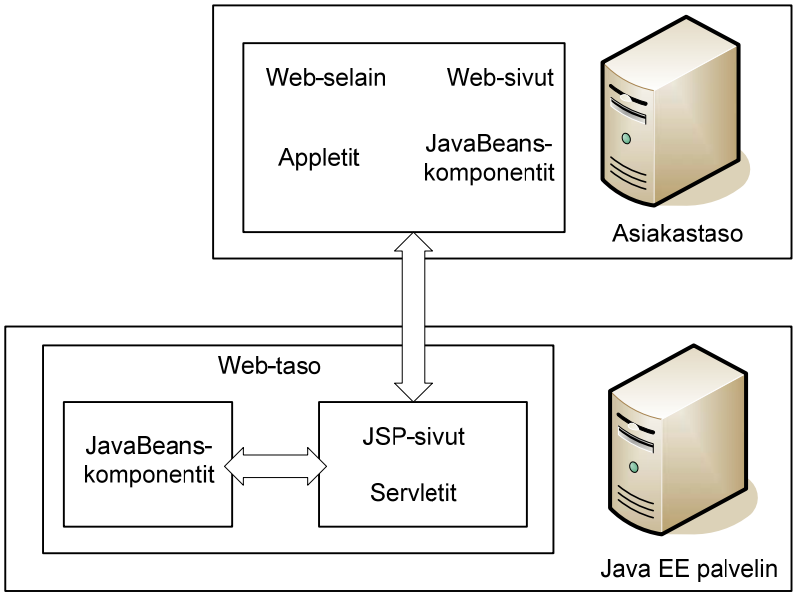
3.5 Java EE

Java Platform, Enterprise Edition (Java EE) on spesifikaatio sovelluspalvelimille ja ohjelmistonkehitysympäristöille web-sovellusten totettamiseksi. Java EE -sovellusmalli rakentuu Java-ohjelmointikielestä ja Java-virtuaalikoneesta. /6/

Java EE -sovellusmalli mahdollistaa web-sovellusten toteuttamisen monitason arkkitehtuuria käyttäen. Sovellusmalli jakaa monitason arkkitehtuuria käyttävän web-sovelluksen toteuttamisen kahteen osaan: sovellus- ja esityslogiikan toteuttamisesta huolehtii kehittäjä, ja Java EE -alusta tarjoaa näitä tukevat palvelut, jotka huolehtivat alemman tason sovelluslogiikan toteuttamisessa. /6/

Java EE -alustan käyttämä monitason arkkitehtuuri jakaa sovelluksessa käytettävän sovelluslogiikan komponentteihin niiden käyttötarkoituksen mukaisesti. Java EE määrittelee komponentin omavaraiseksi toiminnalliseksi ohjelmistoyksiköksi. Komponentit luodaan käyttäen Java-ohjelmointikieltä ja käännetään myös Javan käyttämällä tavalla. Ero komponenttien ja perinteisten Java-luokkien välillä on, että komponentit kootaan Java EE -sovellukseksi. Komponenttien on myös täytettävä niille asetetut syntaksisäännöt ja niiden on vastattava Java EE -spefikaatiossa niille asetettuja vaatimuksia. Komponentit sijoitetaan sovelluspalvelimelle, jossa ne myös suoritetaan sovelluspalvelimen toimesta. /6/

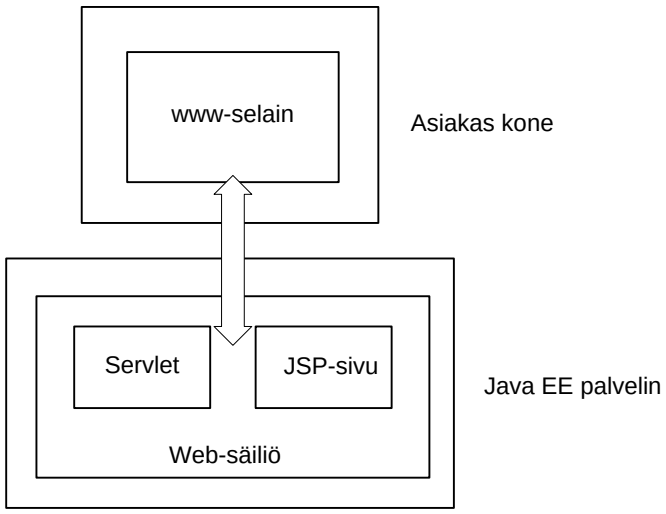
Kuvassa 4 on esitetty monitason arkkitehtuurin asiakas- ja web-tasot. Asiakastaso kommunikoi web-tason kanssa ja sille sijoittuvat web-selain, web-sivut, appletit ja JavaBeans-komponentit. Web-tasolla ovat web-komponentit ja JavaBeans-komponentit. /6/



Kuva 4. Web-komponentit monitason arkkitehtuurissa.

3.5.1 Java EE Container

Java EE tarjoaa jokaiselle komponenttityypille oman säiliönsä (container). Säiliöt toimivat rajapintana komponentin ja komponenttia tukevan alustan matalan tason toiminnallisuuden välissä. Säiliö huolehtii näin ollen säiliössä olevien komponenttien elinkaaresta. Elinkaari kuvaa miten komponentti käyttäytyy sitä kutsuttaessa, kattaen kaikki vaiheet sen luomisesta tuhoamiseen. Ennen kuin komponentti voidaan suorittaa, pitää se koota Java EE -moduuliksi ja sijoittaa säiliöönsä. Moduuliksi kokoamisessa määritetään säiliön asetukset jokaiselle komponentille ja itse suoritettavalle sovellukselle, joka koostuu komponenteista. /6/ Kuvassa 5 on esitetty web-säiliöön sijoittuvat komponentit.



Kuva 5. Web-säiliö.

3.5.2 JavaBeans

JavaBeans-komponentit ovat Java-luokkia, joita käytetään säilömään ja käsittelemään tietoa web-sovelluksissa. Vaikkakin JavaBeans on komponentti, sitä ei määritellä Java EE -spesifikaatiossa Java EE -komponentiksi Mikä tahansa Java-luokka, joka seuraa JavaBeans-komponenteille asetettuja suunnitteluperiaatteita, on JavaBeans-komponentti. /6/

JavaBeans-komponentille asetetut suunnitteluperiaatteet ovat seuraavat:

- Parametritön oletusrakentaja.
- Tietojäsenten arvot ovat asetettavissa julkisella metodilla, joka on muotoa: `setTietojäsen(TietojäsenenLuokka tj) { ... }`.
- Tietojäsenten arvot ovat palautettavissa julkisella metodilla joka on muotoa: `TietojäsenenLuokka getTietojäsen() { ... }`.

3.5.3 Java Servlet

Java Servletit ovat Java-luokkia, jotka suoritetaan sovelluspalvelimella. Niitä käytetään tyypillisesti laajentamaan sovelluspalvelimella sijaitsevia web-sovelluksia. Servletit huolehtivat sovelluspalvelimelle tulleiden pyyntöjen

palvelemisesta ja pyyntöihin vastaamisesta. Servlet käy läpi elinkaarensa vaiheet, servlettiä kutsuttaessa ensimmäisen kerran tai kun sovellus käynnistetään sovelluspalvelimessa. Jos servletistä ei ole luotu instanssia, web-säiliö lataa servlet-luokan, luo instanssin siitä ja alustaa sen. Alustuksen jälkeen servlet on valmis palvelemaan asiakkaiden pyyntöjä. /6/

3.5.4 JavaServer Pages

JavaServer Pages on teknologia, jolla voidaan luoda web-sivulle staattisia ja dynaamisia komponentteja sisältävää sisältöä. JSP tarjoaa kaikki Java Servlet -teknologian dynaamiset ominaisuudet, mutta tarjoaa luonnollisemman lähestymistavan staattisen sisällön luomiseen. /6/

JSP-sivu on tekstidokumentti, joka sisältää kahdenlaista tekstiä: staattista dataa, joka voidaan ilmaista tekstipohjaisella formaatilla (muun muassa HTML, SVG, WML ja XML) ja JSP-elementtejä, jotka koostavat dynaamisen sisällön. JSP-elementit JSP-sivulla voidaan toteuttaa kahdella tavalla, standardilla tai XML-syntaksilla, mutta yksittäinen sivu voi sisältää vain yhtä syntaksia. JSP tukee myös JavaBeans-komponenttien käyttöä JSP-elementtien kanssa. /6/

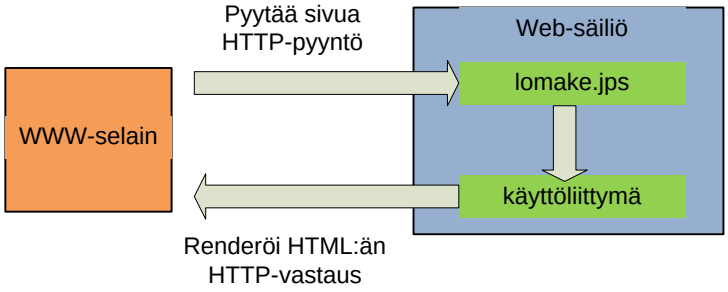
JSP-sivu palvelee sille osoitettuja pyyntöjä servlettinä, joten JSP-sivujen elinkaari ja dynaamiset ominaisuudet määräytyvät käytetyn servletin mukaan. Kun asiakkaan lähettämä pyyntö ohjataan JSP-sivulle, web-säiliö tarkistaa onko JSP-sivun servlet vanhempi kuin JSP-sivu. Servletin ollessa vanhempi, web-säiliö muuntaa JSP-sivun servlet-luokaksi ja kääntää sen. JSP-sivujen kehityksessä tämä ilmenee kääntämisprosessin automaattisena suorittamisena. /6/

3.5.5 JavaServer Faces

JavaServer Faces on palvelinpuolen sovelluskehys Java-pohjaisille web-sovelluksille. JavaServer Faces koostuu kahdesta pääkomponentista, JavaServer Faces APIsta ja kahdesta JSF-tagikirjastosta. JavaServer Faces APIa käytetään käyttöliittymäkomponenttien esittämiseen ja niiden tilanhallintaan. Tilanhallinta

sisältää tapahtumakuuntelijoiden toteutuksen, palvelinpuolella tapahtuvan tiedon validoinnin ja muunnokset, kansainvälisyyden toteuttamisen sekä sivujen navigaation toteuttamisen. JSF-tagikirjastot sisältävät valmiita käyttöliittymäkomponentteja ja niihin liitettäviä toimintoja. Kirjastoja käyttämällä luodaan liitokset käyttöliittymäkomponenttien ja palvelinpuolen objektien välille. /6/

Kuvassa 6 on esitetty asiakkaan ja palvelimen vuorovaikutusta tyypillisessä JSF -sovelluksessa. Asiakkaan pyynnöstä web-säiliö renderöi web-sivun, joka käyttää JavaServer Facesia. Kuvassa esitetty `lomake.jsp` on JSP-sivu, joka sisältää JSF-tageja. Kuvassa esitetty `ui` on web-sovelluksen JSF:llä luotu käyttöliittymä, joka hallinnoi JSP-sivulla viitattuja objekteja. Näihin objekteihin lukeutuvat JSP-sivulla olevat käyttöliittymäkomponentit, komponentteihin liitetyt tapahtuman kuuntelijat ja muuntajat sekä JavaBeans-komponentit. /6/



Kuva 6. JavaServer Faces JSP-sivulla.

Tyypillinen JavaServer Faces -sovellus on suurimmilta osin kuin mikä tahansa muu Javalla toteutettu web-sovellus. Tyypillinen JSP-sivuja käyttävä JSF-sovellus koostuu seuraavista osista:

- JSP-sivuista, JSF:n käyttäminen ei kuitenkaan rajoitu JSP-sivujen käyttämiseen vaan muitakin formaatteja voidaan käyttää.
- Kokoelmasta backing beans -komponentteja, jotka määrittävät sivuilla olevien käyttöliittymäkomponenttien tietojäsenet ja toiminnot käyttäen JavaBeans-komponentteja. JSF-sovelluksen asetustiedostossa backing beans -komponentteja kutsutaan managed bean -nimellä.

- Sovelluksen asetustiedostossa (`faces-config.xml`), jossa määritetään sivujen navigointisäännöt ja asetetaan backing beans -komponentit ja muut valinnaiset komponentit.

- Asennuskuvaimesta (`web.xml`), jossa määritetään sovelluksen asetukset ja sovelluksessa käytettävät Java EE -komponentit.

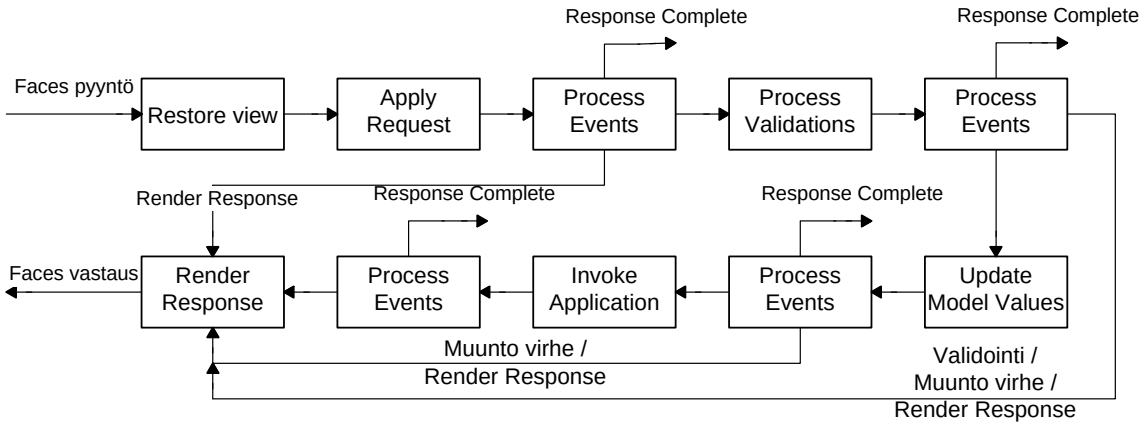
- Käyttöliittymäkomponenteista ja muista JSP-sivuille sijoitetuista objekteista. /6/

FacesServlet

Kaikkien JavaServer Faces -sovellusten tulee määrittää sovelluksen asennuskuvaimessa FacesServelet-luokka. Jokaisella JSF-sovelluksella sovelluspalvelimen web-säiliössä on oma FacesServelet. FacesServlet on servletti, joka hallinnoi pyyntöjen prosessointia JSF-sovelluksen elinkaaressa ja alustaa JSF:n tarvitsemat resurssit. /6/

Elinkaari

JavaServer Faces -sivun elinkaari kattaa yksinkertaistettuna kaikki vaiheet asiakkaan suorittamasta HTTP-pyynnöstä sivulla siihen, että palvelin vastaa muuntamalla sivun HTML:ksi. Saadessaan pyynnön FacesServlet luo FacesContext-objektin, joka sisältää tarvittavat tiedot pyynnön prosessoimiseksi. Kuvassa 7 on esitetty JSF-sivun elinkaari. Elinkaaren aikana sivun näkymä rakennetaan (Restore View), käyttöliittymäkomponentit saavat pyynnössä niille välitetyt arvot (Apply Request), käyttöliittymäkomponentteihin liitetyt validaattorit tarkastavat komponenttien arvot (Process Validations), käyttöliittymäkomponenttien arvot asetetaan niihin viittaavien palvelinpuolen objektien tietojäsenten arvoiksi (Update Model Values), sovelluslogiikka suoritetaan aiemmissa vaiheissa käsitellyllä tiedolla (Invoke Application) ja sivun näkymä näytetään sen uusilla arvoilla (Render Response). Vaiheiden jälkeen tulevissa Process Events -kohdissa voidaan elinkaaren suoritus lopettaa (Response Complete) tai edetä suoraan Render Response -vaiheeseen. /6/



Kuva 7. JSF-sivun elinkaari.

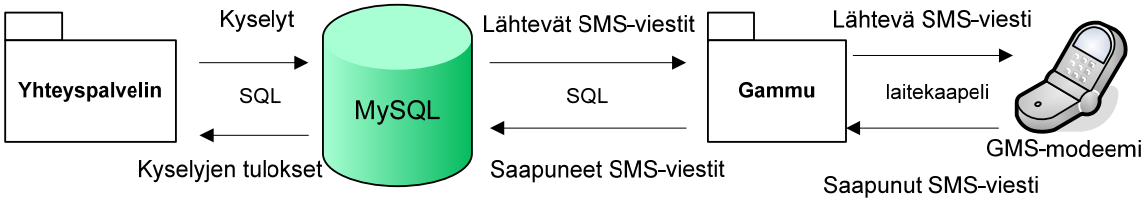
3.5.6 RichFaces

RichFaces on avoimen lähdekoodin käyttöliittymäkomponenttikirjasto AJAX-toiminnallisuuksien lisäämiseen JavasServer Faces -sovelluksiin. Richfaces käyttää JavaServer Faces -kehyksen tarjoamaa elinkaarta, validointia, muunnoksia ja staattisten ja dynaamisten resurssien hallinnointia.

RichFaces tarjoaa kaksi komponenttikirjastoa: Core ja UI. Core-kirjastolla voidaan lisätä JSF-komponentteihin AJAX-toiminnallisuuksia ilman, että olemassa olevia komponentteja tarvitsee korvata tai kirjoittaa JavaScript-koodia. UI-kirjasto sisältää valmiita AJAX-toiminnallisuuksia omaavia RichFaces-komponentteja. /7/

4 Palvelinsovellus

Tässä luvussa tarkastellaan palvelinsovelluksen toimintaa ja toteutusta. Tärkeimmät osat palvelinsovelluksesta käsitellään omissa kappaleissaan. Palvelinsovellus rakentuu Java-ohjelmointikielellä toteutetusta yhteyspalvelimesta, MySQL-tietokannasta, Gammusta ja matkapuhelimesta. Kuvassa 8 on esitetty palvelinsovelluksen osat ja osien väliset kommunikointitavat.



Kuva 8. Palvelinsovelluksen rakenne.

4.1 Palvelinsovelluksen kehitysympäristö

Palvelinsovellus kehityksessä käyttöjärjestelmänä käytettiin Debian GNU/Linux -jakelupaketin 5.0.1 Lenny -versiota.

Palvelinsovelluksen yhteyspalvelin toteutettiin käyttäen Java 1.5 -ohjelmistoalustaa. Yhteyspalvelimen tietokantayhteyksien toteuttamiseen käytettiin Javan tarjoamaa JDBC-rajapintaa. XML-RPC-palvelimen toteuttamiseen yhteyspalvelimessa käytettiin Apache XML-RPC 3.1.2 -kirjastoa.

Yhteyspalvelimen ohjelmointiympäristönä käytettiin avoimen lähdekoodin Eclipse Java 3.2.2-6.1 standard -versiota. Eclipse on yleisesti käytetty ohjelmointiympäristö, joka tukee Javan lisäksi myös muita ohjelmointikieliä, kuten C, C++ ja PHP. Ohjelmointiympäristöä käytettiin yhteyspalvelimen kehityksessä lähinnä editorina, sovelluksen kääntäminen ja suorittaminen tehtiin komentoriviltä.

4.2 Tietokanta

Palvelinsovellus käyttää tietokantanaan avoimen lähdekoodin MySQL-tietokantaa, jonka versio on 5.0.51a. MySQL valittiin siksi, että sitä voitiin käyttää Gammun ja JDBC:n kanssa. MySQL oli myös järjestelmän toteuttajalle ja tietokonekeskuksen henkilöstölle ennestään tuttu. MySQL-tietokanta asennettiin kehityksessä käytetylle Debianille, joten yhteys tietokantaan luotiin localhostin kautta. Tietokannan hallinnointiin käytettiin avoimen lähdekoodin phpMyAdmin-hallintatyökalua, joka mahdollistaa tietokannan hallinnoinnin selaimen kautta.

MySQL-tietokanta toimii SMSGW-järjestelmässä tietovarastona ja rajapintana yhteyspalvelimen ja Gammun välillä. Tietokannassa säilytetään muun muassa lähteneitä ja saapuneita SMS-viestejä.

Tietokannan taulut luotiin Gammun mukana tulleesta SQL-tiedostosta, joka sisältää SMS Daemon -sovelluksen käyttämät tietokantataulut. SMS Daemon on esitelty luvussa 4.4.3 ja sen käyttämät taulut ovat esitelty luvussa 4.4.4. Tietokantaan lisättiin myös asiakassovelluksen tarvitsemia tauluja käyttäjäkohtaisten kontaktilistojen toteuttamiseksi. Nämä taulut ovat esitelty seuraavassa kappaleessa.

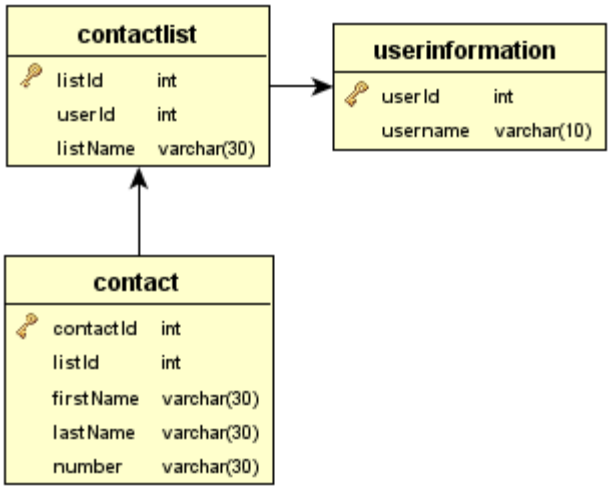
Käyttäjäkohtaiset kontaktilistat

Kuvassa 9 on esitetty käyttäjäkohtaisten kontaktilistojen toteuttamiseksi käytettyjen taulujen rakennekaavio, taulujen käyttöä esitellään luvussa 6.5. Rakennekaaviossa on esitetty taulujen pääavaimet, tauluissa käytetyt tietotyypit ja niiden viittaukset toisiinsa. Taulut ovat InnoDB-tyyppisiä, joten ne tukevat viite-eheyttä.

Taulussa `userinformation` pääavaimena toimii juoksevalla numeroinnilla kasvava `userid`-kenttä, taulussa ylläpidetään käyttäjän käyttäjänimeä (`username`).

Taulussa `contactlist` pääavaimena on juoksevalla numeroinnilla kasvava `listid`-kenttä ja vierasavaimena käytetään `userinformation`-taulun `userid`-kenttää. Taulussa ylläpidetään käyttäjän luomia kontaktilistojen nimiä (`listname`).

Taulussa `contact` pääavaimena käytetään juoksevalla numeroinnilla kasvavaa `contactid`-kenttää ja vierasavaimena käytetään `contactlist`-taulun `listid`-kenttää. Taulussa ylläpidetään käyttäjän kontaktilistoissa käytettävien kontaktien etunimeä (`firstName`), sukunimeä (`lastName`) ja puhelinnumeroita (`number`).



Kuva 9. Käyttäjäkohtaisten kontaktilistojen toteuttamiseksi käytettyjen taulujen rakennekaavio.

4.3 Yhteyspalvelin

Yhteyspalvelin huolehtii palvelinsovelluksen ja siihen yhteyttä ottavien asiakkaiden välisestä tiedonsiirrosta. Yhteyspalvelin tekee tietokantakyselyjä asiakkaiden pyynnöstä palvelinsovelluksen MySQL-tietokantaan ja palauttaa tarvittaessa kyselyn tulokset asiakkaille. Palvelinsovellukseen liitettyyn matkapuhelimen saadessa Nagios-palvelimelle osoitetun SMS-viestin, yhteyspalvelin huolehtii sen lähettämisestä Nagios-palvelimelle. Nagios-verkonvalvontasovellus on esitelty luvussa 5.

Luvuissa 4.3.1, 4.3.2, 4.3.3 ja 4.3.4 käsitellään säikeitä ja Java-ohjelmointikielen ja Apache XML-RPC -kirjaston tarjoamia luokkia. Suoritus säikeessä aloitetaan kutsumalla `java.lang.Thread`-luokan rakentajaa, jonka jälkeen suoritetaan `java.lang.Runnable`-rajapinnan toteuttavan luokan `run()`-metodia kunnes säie pysäytetään tai `run()`-metodin suoritus loppuu. Esityksen yksinkertaistamiseksi, säikeen suorittamisen aloittamisesta käytetään vertausta ”`run()`-metodia kutsutaan”.

Valmiiden kirjastojen kohdalla, kerran esiteltyyn luokkaan ei enää viitata sen koko paketin pituudelta, esimerkiksi kerran mainittuun `java.lang.string` luokkaan

viitataan seuraavan kerran `string`. Yhteyspalvelimen toteuttamiseksi kirjoitettuihin luokkiin ei viitata paketeilla ja ne ovat esitelty luvussa 4.3.1.

4.3.1 Yhteyspalvelimen-luokat

Tämä luku käsittelee yhteyspalvelimen toteuttamisessa käytettyjä luokkia, luokista esitellään niiden käyttötarkoitus, tärkeimmät toiminnallisuudet ja mahdolliset yhteydet toisiin luokkiin. Yhteyspalvelimen toteutus sisältää myös joukon `java.io.Serializable`-rajapinnan toteuttavia luokkia, joita ei käsitellä. Näitä luokkia käytetään välittämään tietoa yhteyspalvelimen sisällä ja XML-RPC-palvelimen ja asiakassovelluksen välillä.

MultiThreadedServer

`MultithreaderServer`-luokka toteuttaa `Runnable`-rajapinnan. `MultithreaderServer`-luokalla on jäsenmuuttujina socket-pohjaisen palvelimen toteuttavan `java.net.ServerSocket`-luokan ja `ServerObserver`-luokan oliot. `MultithreaderServer`-luokka huolehtii socket-pohjaisen palvelimen suorittamisesta.

WorkerRunnable

`WorkerRunnable`-luokka periytyy `java.util.Observable`-luokasta, tämä mahdollistaa tarkkailijoiden rekisteröitymisen luokan kuuntelijoiksi. Luokka toteuttaa myös `Runnable`-rajapinnan. Luokkaa käytetään lukemaan socket-pohjaiseen palvelimeen yhdistyneiden asiakkaiden socket-yhteyteen kirjoittamia viestejä.

ServerObserver

`ServerObserver`-luokka toteuttaa `java.util.Observer`-rajapinnan, joka mahdollistaa luokan rekisteröityä kohteiden kuuntelijoiksi. Kohteen, eli `WorkerRunnable`-luokan ilmoittaessa tilansa muuttuneen, saa `ServerObserver`-luokka käsiteltäväkseen asiakkaan palvelinsovellukselle välittämän viestin. Luokka

muodostaa viestistä SQL-lauseen, jonka se välittää `DatabaseInterface`-luokalle tietokantakyselyn suorittamista varten.

`ServerObserver`-luokka toimii myös Nagios-pavelimelle osoitettujen saapuneiden SMS-viestien lähettäjänä. Tätä tarkoitusta varten on luokalla jäsenmuuttujana `java.util.Timer`-luokan olio, jota käyttämällä luodaan taustaprosessina suoritettava säie. Tasaisin väliajoin säie kutsuu `DatabasePoller`-luokan `run()`-metodia.

DatabaseInterface

`DatabaseInterface`-luokka toteuttaa Singleton-suunnitelumallin. Luokka käyttää JDBC-ajuria luomaan yhteyden palvelinsovelluksen tietokantaan ja sen kanssa kommunikoitiin. Luokka suorittaa `ServerObserver`-, `DatabasePoller`- ja `XmlRpcListener`-luokkien pyynnöistä kyselyjä palvelinsovelluksen tietokantaan ja palauttaa niille tarvittaessa vastauksia.

DatabasePoller

`DatabasePoller`-luokka periytyy `java.util.TimerTask`-luokasta, luokkaa käytetään tasaisin väliajoin kutsumaan `DatabaseInterface`-luokan metodia. Kutsuttu metodi noutaa Nagios-palvelimelle osoitetut prosessoimattomat viestit tietokannasta. Kutsuttu metodi noutaa myös viestejä lähettäneiden henkilöiden numeroita vastaavat käyttäjätunnukset.

XRPCServer

`XRPCServer`-luokan käyttötarkoituksena on luoda ja käynnistää XML-RPC-palvelin.

DBQuery

`DBQuery`-luokka on rajapintaluokka, jota käytetään XML-RPC-palvelimen tarjoamien metodien rekisteröinnissä.

DBQueryImpl

`DBQuery`-rajapinnan toteuttavassa `DBQueryImpl`-luokassa määritetään XML-RPC-palvelimeen rekisteröityjen metodien toteutukset. Luokan metodeja kutsutaan XML-RPC-asiakkaasta, jolloin luokan metodit kutsuvat `XmlRpcListener`-luokan vastaavia metodeita ja palauttavat `XmlRpcListener`-luokan metodien paluuarvot niitä kutsuneelle XML-RPC-asiakkaalle XML-muotoisena HTTP-viestinä.

XmlRpcListener

`XmlRpcListener`-luokan metodeja kutsutaan `DBQueryImpl`-luokan metodeista muodostamaan SQL-lauseet ja suorittamaan kyselyt käyttäen `DatabaseInterface`-luokkaa, joka palauttaa kyselyn tulokset sitä kutsuneelle `XmlRpcListener`-luokan metodille.

4.3.2 Viestin välittäminen Nagios-palvelimelta

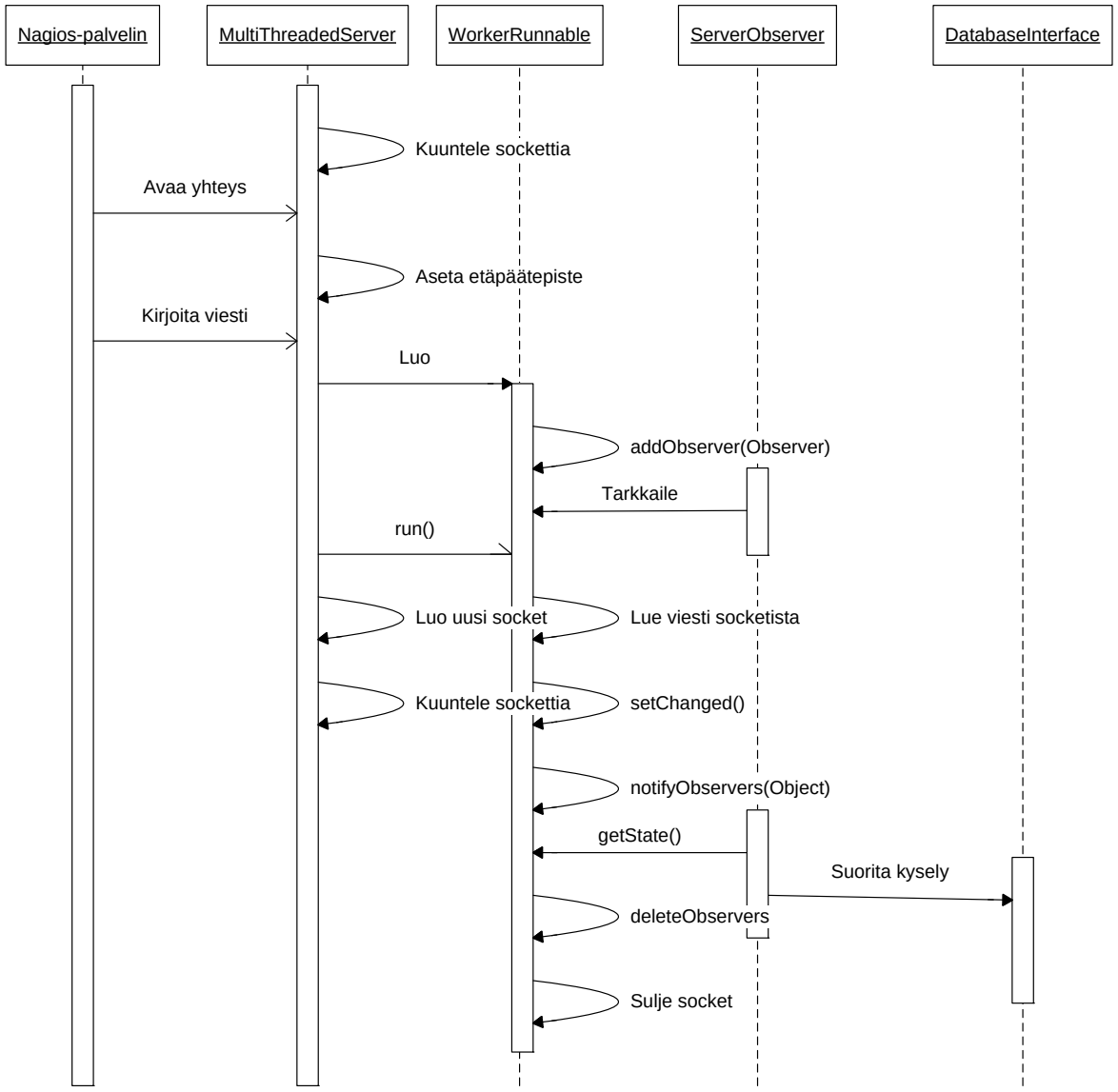
Kuvassa 10 on sekvenssikaaviolla havainnollistettu socket-palvelimen toimintaa. Nagios-palvelimelta avataan SSH-asiakasohjelmaa ja Netcat-työkalua käyttäen yhteys `MultiThreadedServer`-luokassa kuunneltuun sockettiin. `MultiThreadedServer`-luokassa yhteys hyväksytään ja luodaan uusi socket, jonka etäpäätepiste asetetaan Nagios-palvelimen SSH-asiakasohjelman käyttämään osoitteeseen ja porttiin. Nagios-palvelimella välitettävä vikatilanneilmoitus kirjoitetaan `stdin`-komennolla yhteyspalvelimeen yhdistettyyn sockettiin. `MultiThreadedServer`-luokassa luodaan uusi `WorkerRunnable`-luokan olio, jolle välitetään Nagios-palvelimeen yhdistetty socket.

`MultithreaderServer`-luokan jäsenmuuttujana oleva `ServerObserver`-luokan olio rekisteröidään luodulle `WorkerRunnable`-luokan oliolle tarkkailijaksi kutsumalla `addObserver(Observer o)-metodia`. Tämän jälkeen `WorkerRunnable`-luokan olion `run()`-metodia kutsutaan. `MultiThreadedServer`-luokassa luodaan uusi socket ja jatketaan socket-palvelimen porttiin sidotun socketin kuuntelemista.

`WorkerRunnable`-luokan `run()`-metodissa luetaan Nagios-palvelimelta välitetty viesti ja asetetaan luokan tila muuttuneeksi kutsumalla `setChanged()`-metodia.

Seuraavaksi kutsutaan `notifyObservers(Object arg)`-metodia, jolle annetaan parametrinä luettu viesti. `WorkerRunnable`-luokan olion `run()`-metodissa irrottaudutaan tarkkailijasta, suljetaan socket ja suoritetaan `run()`-metodi loppuun.

`ServerObserver`-luokan `update(Observable obj, Object arg)`-metodissa saadaan Nagios-palvelimelta välitetty viesti luokan käytettäväksi `arg`-parametristä. Viestiä käyttämällä muodostetaan SQL-lause ja kutsutaan `DatabaseInterface`-luokan singletonin metodia, joka lisää viestin tietokantaan. Vikatilanne ilmoitus lisätään `outbox`-tauluun odottamaan lähetystä SMS Daemonin toimesta.



Kuva 10. Socket-palvelimen toimintaa kuvaava sekvenssikaavio.

4.3.3 Viestien välittäminen Nagios-palvelimelle

Kuvassa 11 on esitetty sekvenssikaaviolla vikatilanteen kuittaus viestien lähettämistä Nagios-palvelimelle tietokannasta. `ServerObserver`-luokassa luodaan taustaprosessina suoritettava säie ja sen suoritus aloitetaan määritetyn viiveen jälkeen. `DatabasePoller`-luokan `run()`-metodi kutsuu `DatabaseInterface`-luokan singletonin metodia, jossa suoritetaan SQL-lause. Suoritettu lause noutaa kaikki Nagios-palvelimelle osoitetut käsittelemättömät viestit ja viestien lähettäneitä numeroita vastaavat käyttäjätunnukset tietokannasta. Tulokset kyselystä parsitaan vikatilanne kuittausviesteiksi, huomioiden myös moniosaiset viestit. Moniosaisen viestin kohdatessaan parsija noutaa UDH-tunnuksen sekä saapumisajan perusteella viestin muut osat ja yhdistää viestien sisällöt yhdeksi merkkijonoksi.

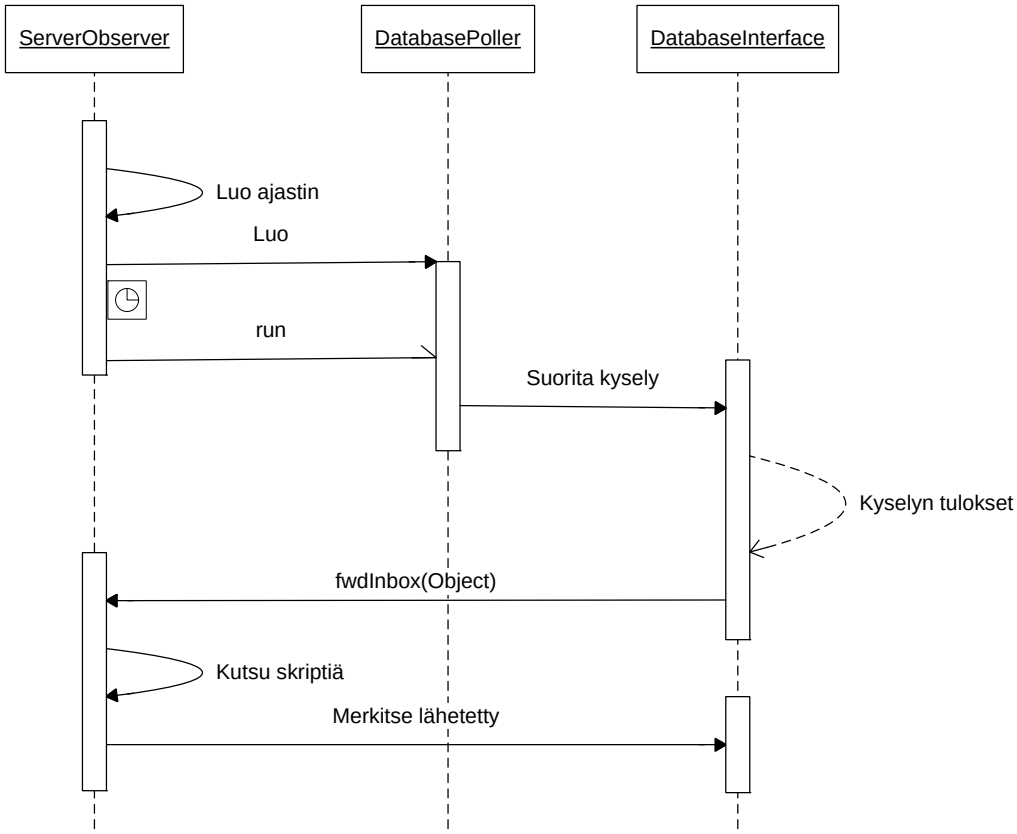
Parsitut viestit välitetään parametrinä kutsuttaessa `ServerObserver`-luokan olion `fwdInbox(Object obj)`-metodia. Metodissa jokaista välitettävää viestiä kohden kutsutaan paikallisella levyllä sijaitsevaa Bourne Shell -skriptiä, joka on esitetty esimerkissä 1. Parametreinä skriptille annetaan viestistä parsittu kuitattavan palvelun palvelin, palvelu ja ylläpitäjän tunniste ja ylläpitäjän kommentti vikatilanteesta. Skripti avaa SSH-asiakasohjelmaa käyttäen yhteyden Nagios-palvelimella ja suorittaa siellä toisen skriptin itse saamillaan parametreillä. Skirptin suorittamisen jälkeen tietokantaan merkitään välitetyt viestit käsitellyiksi.

Esimerkki 1. Palvelun vikatilanteen kuittaus Bourne Shell -skriptillä.

```
#!/bin/sh
HOST=$1
SVC=$2
AUTHOR=$3
REPLY=$4

ssh user@nagios.server.com  "/path/to/script/ackSvc $HOST $SVC
$AUTHOR \"$REPLY\""

exit 0
```



Kuva 11. Nagios-palvelimelle viestin välittämistä kuvaava sekveknssikaavio.

4.3.4 XML-RPC-palvelimen toiminta

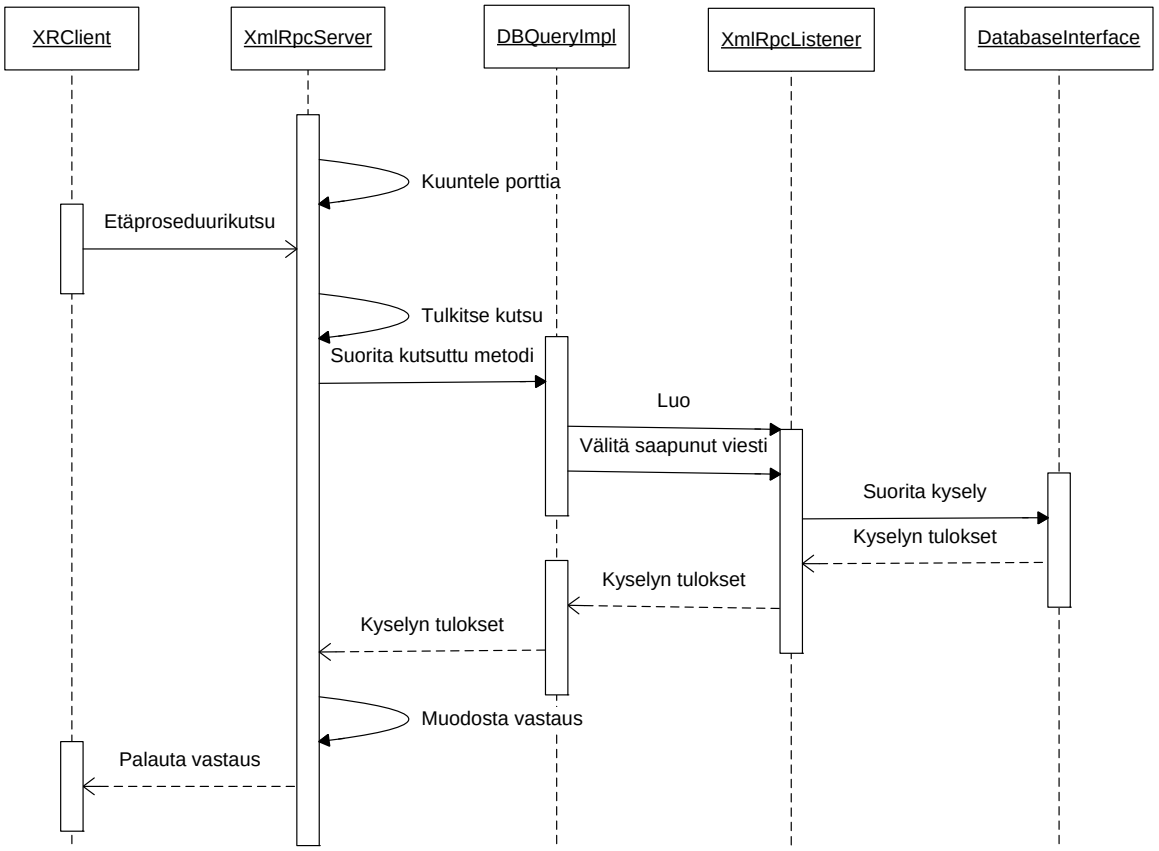
Luotaessa `XRPCServer`-luokan olio, välitetään sille rakentajan parametrinä kuunneltava porttinumero. Samalla luodaan `org.apache.xmlrpc.webserver`-luokan olio, jolle välitetään kuunneltava portti. `Webserver`-luokka toteuttaa minimalistisen HTTP-palvelimen. Kutsuttaessa luokan `getXmlRpcServer()`-metodia palauttaa se `org.apache.xmlrpc.server.XmlRpcServer`-luokan olion. `XmlRpcServer`-luokka toteuttaa XML-RPC-protokollaa toteuttavan palvelimen, joka palvelee asiakkaita omissa säikeissään.

XML-RPC-palvelimen asiakkaille tarjoamat metodit kirjataan käyttäen `org.apache.xmlrpc.server.PropertyHandlerMapping`-luokan oliota, kutsumalla sen `addHandler(java.lang.String pKey, java.lang.Class`

pClass)-metodia. Parametri pKey on rajapintaluokka DBQuery ja pClass-parametri on rajapintaluokan toteuttava DBQueryImpl-luokka. Palvelimen tarjoamiin metodeihin kirjataan kaikki metodit, joilla on public-näkyvyys, mutta ne eivät ole luokkakohtaisia (static) ja niiden paluuarvo on muu kuin void. Kirjatut palvelimen metodit asetetaan palvelimeen kutsumalla XmlRpcServer-luokan olion, setHandlerMapping(XmlRpcHandlerMapping pMapping)-metodia. Parametrinä välitetään PropertyHandlerMapping-luokan olio, johon palvelimen ominaisuudet kirjattiin.

Seuraavaksi asetaan palvelimen käyttämät asetukset kutsumalla XmlRpcServer-luokan olion getConfig()-metodia, joka palauttaa palvelimen oletusasetukset sisältävän org.apache.xmlrpc.XmlRpcConfig-luokan olion. Olio sijoitetaan tyyppimuunnosta käyttäen org.apache.server.XmlRpcServerConfigImpl-luokan olioon. Kutsumalla XmlRpcServerConfigImpl-luokan olion setEnabledForExtensions(boolean pExtensions)-metodia, pExtensions-parametrin arvolla true, asetetaan palvelin tukemaan java.io.Serializable-rajapinnan toteuttavien olioiden vastaanottamista metodikutsuissa ja palauttamista paluuarvoissa. Tämä asetus mahdollistaa myös muiden XML-RPC-spefikaation ulkopuolisten tietotyyppien käyttämisen. Kutsumalla webserver-luokan olion start()-metodia palvelin käynnistetään omassa säikeessään ja se on valmis palvelemaan asiakkaita.

Kuvassa 12 on kuvattu sekvenssikaaviolla yhteyspalvelimen XML-RPC-palvelimen toimintaa. Asiakkaan kutsuessa palvelimen tarjoamaa metodia, palvelin tulkitsee asiakkaan kutsun ja suorittaa pyydetyn metodin. DBQueryImpl-luokassa luodaan XmlRpcListener-luokan olio, josta kutsutaan asiakkaan välittämän tiedon vastaavaa metodia. XmlRpcListener-luokan metodissa luodaan asiakkaalta saatujen tietojen perusteella SQL-lause, joka suoritetaan kutsumalla Databaseinterface-luokan singletonin tietokantakyselyn suorittavaa metodia. Kyselyn tulokset palautetaan takaisin XmlRpcServer-luokalle, joka muodostaa vastauksen ja palauttaa sen asiakkaalle XML-muotoisena HTTP-viestinä.



Kuva 12. XML-RPC-palvelimen toimintaa kuvaava sekvenssikaavio.

4.4 Gammu

Gammu-projektin tarjoamat työkalut toteuttavat SMSGW-järjestelmän palvelinsovelluksen SMS-yhdyskäytävän. Järjestelmän kehityksessä käytettiin Gammun versiota 1.24.0.

4.4.1 Yleistä

Gammu-projekti on Michal Čihařin ylläpitämä C-ohjelmointikielellä kirjoitettu avoimen lähdekoodin projekti, joka sisältää sovelluksia, komentosarjoja ja ajureita matkapuhelimien ja muiden GMS-modeemien hallintaan. Gammusta on saatavilla

lähdekoodit, windows-binääri, Debian- ja Ubuntu-paketit ja RPM-paketti. Gammu-projekti tukee useita markkinoilla olevia laitemalleja ja projektin pitkän tähtäimen tarkoituksena on luoda mahdollisimman monia eri malleja tukeva API. /8/ Tämän johdosta, Gammu-projekti ei välttämättä tue kaikkia tietyn puhelinmallin ominaisuuksia, mutta mahdollistaa projektin jatkuvan käytön, vaikka markkinoille tulee jatkuvasti uusia laitemalleja.

Gammu perustuu samankaltaiseen Gnokii-sovellukseen, mutta tarjoaa lisäominaisuuksina SMS Daemonin, libGammun ja Python-sidokset. SMS Daemon on Gammu-projektin tarjoama sovellus ja sitä tarkastellaan luvussa 4.4.3. Gammu-projektin ydin on sen libGammu-kirjasto, joka tarjoaa samat ominaisuudet, jotka löytyvät Gammu-komentorivi työkalusta ja SMS Daemonista. /8/ Ensin harkittiin yhdyskäytävän toteuttamista käyttäen libGammu-kirjastoa. Tästä kuitenkin luovuttiin hyvin pian sen tuodessa lisää työtä, nyt jo laajan järjestelmän toteuttamiseen. Myös SMS Daemon ja sen kanssa käytettävä tietovarasto, tarjosivat kätevän rajapinnan käyttää yhdyskäytävää Java-sovelluksesta.

Gammu tarjoaa seuraavat perusominaisuudet, jotka toimivat riippuen käytetystä puhelimen mallista, kaapelista ja yhteysprotokollasta:

- Puheluiden listaus, käynnistäminen ja käsittely laitteessa.
- SMS-viestien noutaminen, tallentaminen ja lähettäminen laitteessa.
- MMS-viestien noutaminen laitteesta.
- Puhelinmuistion noutaminen ja vieminen laitteesta.
- Kalenterin ja tehtävien noutaminen ja vieminen.
- Laitteen ja verkko tietojen noutaminen.
- Pääsyn laitteen tiedostojärjestelmään. /8/

4.4.2 GSM-modeemiin yhdistäminen

Gammu voidaan liittää matkapuhelimeen käyttäen joko laitekaapelia, infrapunaa tai Bluetoothia. /8/ Yhteyden toimivuus riippuu matkapuhelimen tai GSM-modeemin mallista sekä yhteydessä käytettävästä laitteesta tai kaapelista.

Laitteeksi valittiin Nokia 3120 matkapuhelin, joka yhdistettiin palvelimen USB-porttiin Nokia CA-42 -laitekaapelilla ja SMS Daemon asetettiin käyttämään fbuspl2303-protokollaa yhteydessä. Myös muita Nokian puhelinmalleja ja kaapeleita testattiin eri protokollilla, mutta edellä mainitulla yhdistelmällä saavutettiin parhaat tulokset.

4.4.3 SMS Daemon

SMS Daemon on itsenäisesti suoritettava sovellus, joka lähettää SMS-viestejä tietovarastosta ja tallentaa saapuneita SMS-viestejä tietovarastoon. SMS Daemon käynnistetään käyttäen komentoa `gammu-smsd –c configfile`, jossa optionaalisena parametrinä annetaan asetustiedoston sijainti, muutoin asetustiedostoa haetaan oletuksena polusta `/etc/gammu-smsdrc` tai `~/gammu-smsdrc`. SMS Daemonissa käytetty asetustiedosto on esitetty kokonaisuudessaan liitteessä 1, tärkeimmät kohdat asetustiedostossa ovat listattu esimerkissä 2.

Arvoon `port` asetetaan tietokoneen portti, johon GSM-modeemi on yhdistetty laitekaapelilla, `connection`-arvo määrittää SMS Daemonin ja matkapuhelimen kommunikoinnissa käytetyn protokollan, `service` asettaa käytettävän tietokannan tyypin ja `PIN` on matkapuhelimen PIN-koodi. Arvo `commtimeout` määrittää odotettavan ajan sekunneissa, jonka jälkeen matkapuhelimen ja tietovaraston tiedustelujakso suoritetaan uudelleen. Kohta `sendtimeout` asettaa SMS-viestien lähetyksen epäonnistuessa odotettavan ajan ennen kuin viestiä yritetään lähettää uudelleen. Tietokannan käyttäjätunnus tulee kohtaan `user`, `password` on tietokannan käyttäjän salasana, `pc` määrittää mistä tietokanta löytyy ja `database` tietokannan nimen.

Esimerkki 2: SMS Daemon asetustiedoston tärkeimmät kohdat.

```
port = /dev/ttyACM0
connection = fbuspl230
service = MYSQL
PIN = 4321
commtimeout = 30
sendtimeout = 30
user = gammu
password = gammupassword
pc = localhost
database = sms
```

Käynnistettäessä SMS Daemon, asetustiedosto luetaan ja SMS Daemon luo yhteyden asetustiedostossa määritettyyn laitteeseen. Yhteyden luonnin onnistuessa, SMS Daemon aloittaa asetustiedostossa määritetyn laitteen ja tietovaraston tiedustelun, asetetun jakson välein. SMS Daemonin löytäessä tietovarastosta lähettämistä odottavan viestin, SMS Daemon siirtää viestin laitteelle lähettämistä varten ja suorittaa laitteessa lähetyskäskyn. Laitteeseen saapuneet SMS-viestit SMS Daemon noutaa ja tallentaa tietovarastoon.

4.4.4 Tietovarasto

SMS Daemonin kanssa käytettäväksi tietovarastoksi valittiin MySQL. Seuraavat tietovarastot ovat myös tuettuja SMS Daemonissa:

- Tiedostoon tallentaminen
- PostgreSQL
- Perl DBI

Gammu-projekti sisältää SMS Daemonin käyttämiseen tarvittavat tietokantataulut, taulut ovat MyISAM-tyyppisiä, jotta SMS Daemonin niihin kohdistamat lukemis- ja kirjoitusoperaatiot olisivat mahdollisimman nopeita. Taulujen rakennekaaviot ovat esitetty liitteessä 2. Seuraavissa kappaleissa käydään läpi tietokantataulujen käyttötarkoitukset:

daemons ja gammu

Taulu `daemons` sisältää tietoa suoritettavista SMS Daemon-sovelluksista. Järjestelmässä käytetty Gammun versio jätti taulun tyhjäksi. Taulu `gammu` sisältää käytettävän Gammu-projektin versiossa käytettyjen tietokantataulujen version (`Version`).

inbox-taulu

Taulussa `inbox` säilytetään SMS Daemonin laitteesta noutamia saapuneita SMS-viestejä. Taulun tietokentissä säilytetään seuraavanlaisia tietoja:

- Viestin viimeinen muokkausajankohta (`UpdatedInDB`)
- Ajankohta jolloin viesti saapui laitteelle (`ReceivingDateTime`)
- Viestin sisältö muunnettuna heksadesimaaliksi (`Text`)
- Viestin lähettäjän puhelinnumero (`SenderNumber`)
- Viestissä käytetty pakkausmenetelmä (`Coding`)
- Moniosaisen viestin UDH-tunnus (`UDH`)
- Viestin lähettämisessä käytetyn viestikeskuksen numero (`SMSCNumber`)
- Gammun määrittelemä viestin luokka (`Class`)
- Viestin sisältö ihmisen ymmärtämässä muodossa (`TextDecoded`)
- Viestin identifioinnissa käytettävä juoksevalla numeroinnilla kasvava perusavain (`ID`)
- Viestin laitteesta noutaneen SMS Daemonin tunniste (`RecipientID`)
- Viestin prosessoinnin tilaa ilmaiseva arvo (`Processed`)

outbox taulu

Useampi kuin yksi SMS Daemon voi käyttää samaa tietovarastoa SMS-viestien lähettämiseen, taulun `Creator`-kentässä määritetään mitä SMS Daemonia käytetään viestin lähettämiseen. Tauluun `outbox` tallennetaan lähetystä odottavat SMS-viestit,

viestin ollessa moniosainen muut osat täytyy tallentaa `outbox_multipart`-tauluun. Taulu sisältää seuraavanlaisia tietoja:

- Viestin viimeinen muokkausajankohta (`UpdatedInDB`)
- Ajankohta jolloin viesti lisättiin tauluun (`InsertIntoDB`)
- Ajankohta, jonka jälkeen viesti lähetetään (`SendingDateTime`)
- Viestin sisältö muunnettuna heksadesimaaliksi (`Text`)
- Viestin vastaanottava puhelinnumero (`DestinationNumber`)
- Viestissä käytetty pakkausmenetelmä (`Coding`)
- Moniosaisen viestin UDH-tunnus (`UDH`)
- Gammun määrittelemä viestin luokka (`Class`)
- Viestin sisältö ihmisen ymmärtämässä muodossa (`TextDecoded`)
- Viestin identifioinnissa käytettävä juoksevalla numeroinnilla kasvava perusavain (`ID`)
- Moniosaista viestiä ilmaisevaa arvo (`Multipart`)
- GSM-standardissa määritetty viestin validi elinikä (`RelativeValidity`)
- Viestin lähettävän SMS Daemonin tunnus (`SenderID`)
- Ajankohta, jolloin viestin lähetys epäonnistui (`SendingTimeOut`)
- SMS-viestin välitystietojen käyttämistä ilmaiseva arvo (`DeliveryReport`)
- Viestin lähettäjän tunnus (`CreatorID`)

outbox_multipart

Taulu sisältää lähettävien moniosaisten SMS-viestien osat. Taulussa ylläpidetään osan sisältöä muunnettuna heksadesimaaliksi (`Text`), viestissä käytettyä pakkausmenetelmää (`Coding`), viestin UDH-tunnistetta (`UDH`), Gammun määrittelemää viestin luokkaa (`Class`), viestin sisältöä ihmisen luettavassa muodossa (`TextDecoded`), viestin identifioinnissa käytettävää juoksevalla numeroinnilla kasvavaa perusavainta (`ID`) ja moniosaisen viestin osan määrittävää arvoa (`SequencePosition`).

phones

Taulu sisältää tietoja Gammuun liitetyistä laitteista, taulun tiedot päivitetään SMS Daemonin tiedustelujaksoissa. Taulussa ylläpidetään puhelimelle määritettyä tunnusta (`ID`), aikaa jolloin tiedot laitteesta päivitettiin (`UpdatedInDB`), aikaa jolloin laitteeseen luotiin yhteys Gammusta (`InsertIntoDB`), aika jolloin taulussa listatut tiedot laitteesta vanhenevat (`TimeOut`), viestien lähettämisen sallivan arvon (`Send`), viestien vastaanottamisen sallivan arvon (`Receive`), perusavaimena toimivaa laitteen IMEI-koodia (`IMEI`), käytettävän Gammun versionumeroa (`Client`), laitteen akun tasoa (`Battery`), laitteen verkkosignaalin voimakkuutta (`Signal`), kokonaislukumäärää lähetetyistä viesteistä (`Sent`) ja kokonaislukumäärää vastaanotetuista viesteistä (`Received`).

sentitems

Taulussa ovat listattuna `outbox-` ja `outbox-multipart`-tauluista SMS Daemonin noutamat lähetystä odottaneet viestit. Taulu sisältää onnistuneesti lähetetyt viestit ja myös viestit, joita ei onnistuttu lähettämään. Taulussa ylläpidetään seuraavia tietoja:

- Viestin päivitysaikaa(`UpdatedInDB`)
- Aikaa jolloin viesti lisättiin tauluun (`InsertIntoDB`)
- Viestin lähettämisen yrityksen aikaa (`SendingDateTime`)
- Aikaa jolloin viestin välitystieto vastaanotettiin (`DeliveryDateTime`)
- Viestin sisältöä muunnettuna heksadesimaaliksi (`Text`)
- Viestin vastaanottajan puhelinnumeroa (`DestinationNumber`)
- Viestissä käytettyä pakkausmenetelmä(`Coding`)
- UDH-tunnusta viestin ollessa moniosainen (`UDH`),
- Viestin lähettämisessä käytettyä viestikeskuksen numeroa (`SMSCNumber`)

- Gammussa määriteltyä viestin luokkaa(`Class`)
- Viestin sisältöä ihmisen luettavassa muodossa(`TextDecoded`)
- Viestin identifioinnissa käytettävää juoksevalla numeroinnilla kasvavaa perusavainta(`ID`)
- Viestin lähettäneen SMS Daemonin tunnusta(`SenderID`)
- Moniosaisen viestin osan määrittävää arvoa (`SequencePosition`)
- Viestin lähetyksestä syntynyt tila(`Status`)
- Välitystiedosta saatua GSM-standardissa määritettyä tilaa viestin välittämisestä vastaanottajalle (`StatusError`) /9/
- GSM-standardissa määritettyä viestin lähettämisestä syntynyttä viitenumeroa (TP-MR) /9/ (`TPMR`)
- GSM-standardissa määritettyä viestin elinikää (Relative Validity) /9/
- Viestin lähettämisessä käytetty SMS Daemonin tunnus (`CreatorID`)

pbk_groups ja pbk

SMS Daemon ei käytä pbk- ja pbk_groups-tauluja, mutta ne ovat Gammu-projektin mukana tulevissa tietokantatauluissa. Taulut on lisätty käytettäväksi sovelluksissa, joissa hyödynnetään SMS Daemonia. Järjestelmässä tauluja käytetään tunnistamaan Nagios-palvelimelle osoitetun vikatilannekuittauksen lähettäjä. Taulussa `pbk_groups` ylläpidetään käyttäjäryhmän nimeä (`GroupName`) ja käyttäjäryhmän identifioinnissa käytettävää juoksevalla numeroinnilla kasvavaa perusavainta (`ID`). Taulussa `pbk` ylläpidetään ryhmäntunnusta (`GroupID`) (`pbk_groups-taulu`), käyttäjän nimeä (`Name`) ja käyttäjän puhelinnumeroa (`Number`).

5 Nagios verkonvalvontasovellus

Nagios on avoimen lähdekoodin verkonvalvontasovellus, jonka alkuperäinen kehittäjä on Ethan Galstad. Tästä eteenpäin Nagios-verkovalvontasovellusta kutsutaan Nagios-sovellukseksi. Nagios-sovelluksella voidaan valvoa lähiverkoissa toimivia verkko-osoitteen omaavia fyysisiä laitteita ja palveluita. Nagios määrittää valvottavat kohteet kahteen ryhmään, palvelimiin (hosts) ja palveluihin (services). Nagios ilmoittaa vikatilanteista ylläpitäjille sille määritetyllä tavalla, useimmiten tämä tapahtuu sähköpostilla tai Nagios web-käyttöliittymän kautta, joilla myös vikatilanteet voidaan kuitata. /10, 11, s. 22, s. 24/ Ajantasaisten vikatilannetietojen saaminen edellä mainituilla tavoilla on mahdollista vain, jos päivystäjällä on käytettävissä internet-yhteys. SMS-viestin käyttäminen vikatilannetietojen lähettämisessä ja vikatilanteiden kuittaamisessa vapauttaa päivystäjän internet-yhteyden saatavuuden asettamista rajoitteista, GSM-verkon tarjotessa suuremman kattavuuden Suomessa ja ulkomailla.

Seuraavat luvut käsittelevät ratkaisuja, joilla SMS-viesteillä tapahtuva vikatilannetietojen lähettäminen ja vikatilanteiden kuittaaminen toteutettiin järjestelmässä.

5.1 Vikatilanne ilmoituksen lähettäminen

Valvottavan verkon palvelimissa ja palveluissa tapahtuville vikatilanteille voidaan määritellä komentoja, jotka Nagios suorittaa niiden tapahtuessa, /10/ tämä mahdollistaa SMS-viestin lähettämisen vikatilanteesta ylläpitäjälle. Esimerkissä 3 on esitetty Nagios-palvelimella suoritettava komento vikatilanteen tapahtuessa palvelussa, komento lähettää vikatilanne ilmoituksen palvelinsovellukselle SMS-viestinä lähetystä varten.

Komento avaa yhteyden määriteltyyn etäpalvelimen porttiin käyttäen SSH-asiakasohjelmaa ja suorittaa etäpalvelimella Netcat-komennon, joka avaa yhteyden palvelinsovelluksen kuuntelemaan porttiin, johon viesti kirjoitetaan. Viestissä välitetään ilmoituksen vastaanottavan ylläpitäjän puhelinnumero(`$CONTACTPAGER$`), palvelu(`$SERVICEDESC$`), jossa vikatilanne tapahtui, palvelun palvelin

($HOSTALIAS$), palvelun sen hetkinen tila($SERVICESTATE$), palvelun palvelimen IP-osoite($HOSTADDRESS$) sekä päivämäärä ja kellonaika ($SHORTDATETIME$), jolloin vikatilanne tapahtui. Viestin lopussa on `Re:` -merkkijono, jonka perään ylläpitäjä voi kirjoittaa kommentin vikatilanteesta matkapuhelimellaan.

Esimerkki 3: Vikatilanne ilmoituksen palvelinsovellukselle lähettävä Nagios-komento.

```
define command{
command_name notify-service-by-sms
command_line ssh –f –L remoteport:localhost:localport
user@remotehost sleep 10; \nc remotehost remoteport <
/usr/bin/printf “%b” “$CONTACTPAGER$#Svc:$SERVICEDESC$
Host:$HOSTALIAS$ State:$SERVICESTATE$ ($HOSTADDRESS$)
Date:$SHORTDATETIME$ Re:”
}
```

Komennon nimi on `notify-service-by-sms`, komento on lisättävä Nagios-sovelluksessa ylläpitäjän vikatilanneilmoituksien lähetystapoihin, jotta päivystäjä saisi vikatilannetietoja SMS-viesteillä. /10/. Merkkijonon `command_line` jälkeen alkaa itse Nagios-palvelimella suoritettava komento, jonka palvelimen komentotulkki ajaa. Esimerkissä 4 on esitetty Nagios-palvelimelta ylläpitäjän matkapuhelimeen lähetetty vikatilanneilmoitus. Vikatilanne voidaan kuitata lähettämällä viesti takaisin palvelinsovellukseen liitettyyn matkapuhelimeen.

Esimerkki 4: Matkapuhelimeen saapunut vikatilanneilmoitus.

```
Svc:https-cal
Host:cal.tpu.fi
State:CRITICAL
(193.167.68.228)
Date:07-08-2009 10:37:35
Re:
```

5.2 Vikatilanteen kuittaaminen

Nagios-sovelluksen tarjoama nimetty putki `nagios.cmd` mahdollistaa komentojen antamisen sovellukselle /10/, käyttämällä putkea voidaan toteuttaa vikatilanteiden kuittaaminen SMS-viesteillä. Nimettyyn putkeen voidaan kirjoittaa

komentoja suoraan tai vaikkapa suorittaa kirjoitus skriptistä. Esimerkissä 5 on esitetty ylläpitäjän järjestelmään lähettämä vikatilanteen kuittaus.

Esimerkki 5: Matkapuhelimesta lähetetty vikatilanteen kuittaus.

```
Svc:https-cal
Host:cal.tpu.fi
State:CRITICAL
(193.167.68.228)
Date:07-08-2009 10:37:35
Re: Korjataan lounaan jälkeen
```

Esimerkissä 6 on esitetty palvelun vikatilanteen kuittaaminen järjestelmässä Bourne Shell -skriptillä. Skripti sijaitsee Nagios-palvelimella ja sitä kutsutaan palvelinsovelluksesta. Skriptille annetaan parametreinä SMS-viestillä lähetetyt palvelun vikatilanneilmoituksessa lähetetyt tiedot palvelusta(`$2`) ja sen palvelimesta(`$1`). Uusina tietoina saadaan ilmoituksen kuitanneen ylläpitäjän tunnus(`$3`) ja kommentti vikatilanteesta(`$4`). Nämä tiedot sijoitetaan vikatilanteen kuittaus komentoon `ACKNOWLEDGE_SVC_PROBLEM.`

Esimerkki 6: Palvelun vikatilanteen kuittaus Bourne Shell -skriptillä.

```
#!/bin/sh
HOST=$1
SVC=$2
AUTHOR=$3
REPLY=$4
now=’date +%s’
commandfile=’/usr/local/nagios/var/rw/nagios.cmd’
/bin/printf "[%lu] ACKNOWLEDGE_SVC_PROBLEM;
$HOST;$SVC;1;1;1;$AUTHOR;$REPLY\n" $now > $commandfile
exit 0
```

Vikatilanteen kuittaus komennossa on kolme parametriä, joiden arvoja ei anneta SMS-paluuviestissä. Esimerkissä 7 on esitetty palvelun vikatilanteen kuittaus -komento tarkemmin.

Esimerkki 7: Palvelun vikatilanteen kuittaus-komento

```
ACKNOWLEDGE_SVC_PROBLEM;<host_name>;<service_description>;<sticky>;
<notify>;<persistent>;<author>; <comment>
```

Parametrin `<sticky>` ollessa 1 ,vikatilanne ilmoitusten lähettäminen jatkuu kunnes palvelun tila palautuu `OK`:ksi. Muutoin palvelun tilan säilyessä, ilmoituksia ei enään lähetetä. Parametrin `<notify>` ollessa 1, lähetetään muille ylläpitäjille ilmoitus palvelun vikatilanteen kuittauksesta. Jos parametrille `<persistent> on` asetettu arvoksi 1, päivystäjän kommentti säilytetään kuittauksen yhteydessä Nagioksen uudelleen käynnistyksessä. Muutoin kommentti poistetaan Nagios-sovelluksen uudelleen käynnistyksessä. /10/

6 Asiakassovellus

Tässä luvussa tarkastellaan asiakassovelluksen toteutusta ja toimintaa yleisellä tasolla. Asiakassovellus on JavaServer Faces –sovelluskehystä käyttävä web-sovellus, joka suoritetaan Java EE –spesfikaatiota toteuttavalla sovelluspalvelimella.

Asiakassovellus rakentuu JSP-sivuista, niitä tukevista JavaBeans-komponenteista ja Java-ohjelmointikielellä toteutetusta `DataTransfer`-kirjastosta, joka toteuttaa sovelluksen yhteydet palvelinsovellukseen ja käytettyyn hakemistopalveluun. Tärkeimmät osat asiakassovelluksesta käsitellään omissa luvuissaan.

6.1 Kehitysympäristö

Tässä luvussa käsitellään järjestelmän asiakassovelluksen toteuttamisessa käytettyjä alustoja, kehyksiä, kirjastoja sekä työkaluja.

Asiakassovelluksen alustana oli käytössä Sun Microsystemsin Java EE 5 SDK ja sovelluspalvelimena Sun Microsystemsin GlassFish v3 Prelude. Käyttöliittymän toteuttamisessa käytettiin JavaServer Faces 1.2 -sovelluskehystä ja RichFaces 3.3.1 -komponenttikirjastoa. Sovelluksen debuggaus suoritettiin käyttäen Java Debuggeria.

Sovelluksen ulkoiset yhteydet toteuttava `DataTransfer`-kirjasto toteutettiin käyttämällä Java 1.6 -ohjelmistoalustaa. Kirjastossa käytetään XML-RPC-asiakkaan toteuttavaa Apache XML-RPC 3.1.2 –kirjastoa ja JNDI-rajapintaa.

Web-käyttöliittymän kehittämisessä käytettiin Sun Microsystemsin Netbeans 6.7 IDE -ohjelmointialustaa. Netbeans ladattiin Java-pakettina, jolloin mukana tulivat myös Java EE, GlassFish ja JavaServer Faces. Näin kehitysympäristö saatiin nopeasti pystyyn kehittämisen aloittamista varten.

Netbeans mahdollisti kehitetyn koodin testataamisen ja debuggauksen, sen tukiessa käytettyjä tekniikoita täydellisesti. Netbeans asennettiin MicroSoft Windows XP -käyttöjärjestelmään.

6.2. Asetustiedostot

Tässä luvussa käydään läpi asiakassovelluksessa käytetyt asetustiedostot ja niissä käytetyt tärkeimmät asetukset. Asetustiedostoina käytettiin `WEB-INF`-kansiossa sijaitsevia `web.xml-` ja `faces-config.xml`-tiedostoja.

Asennuskuvain, eli `web.xml`-tiedosto, on XML-syntaksia käyttävä koko asiakassovelluksen laajuinen asetustiedosto. Asetustiedostossa määritellään asiakassovelluksessa käytettävät asetukset ja komponentit.

Esimerkissä 8 määritetään käyttöliittymäkomponentteihin tallennetun tiedon säilyttämisestä asiakkaassa tai sovelluspalvelimella. Asetettaessa tiedot tallentumaan sovelluspalvelimelle, tiedot tallentuvat käytettyyn HTTP-sessioon. Tallennettaessa tieto asiakkaalle, tieto tallentuu asiakkaalle takaisin lähetetyn sivun piilotettuihin kenttiin. Asiakkaalle tallentaminen vähentää muistin kulutusta sovelluspalvelimella, mutta kutsuissa välitetyn tiedon määrää kasvaa. /6/

Esimerkki 8. Käyttöliittymäkomponenttien tiedon tallentaminen.

```
<context-param>
   <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
   <param-value>server</param-value>
</context-param>
```

RichFaces-käyttöliittymäkomponenttikirjasto otetaan käyttöön seuraavilla asetuksilla. Esimerkissä 9 on esitetty RichFaces-filtterin asettaminen, filtterille määritetään nimi ja sille asetetaan polku, josta luokka ladataan. Filtteri asetetaan kuuntelemaan asiakkaan lähettämiä AJAX-pyyntöjä FacesServletille. Asetus mahdollistaa RichFaces-käyttöliittymäkomponenttien käyttämisen asiakassovelluksen JSP-sivuilla.

Esimerkki 9. RichFacesin käyttöön ottaminen JSF-sovelluksessa.

```
<filter>
   <display-name>RichFaces Filter</display-name>
   <filter-name>richfaces</filter-name>
   <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>

<filter-mapping>
   <filter-name>richfaces</filter-name>
   <servlet-name>Faces Servlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Käyttäjän tunnistamiseen ja pääsynhallinnan toteuttamiseen käytetty Sun Java System Access Manager asetetaan kuuntelemaan kaikkia web-sovellukselle lähettyjä pyyntöjä esimerkissä 10. Sun Java System Access Manageria on esitelty luvussa 6.5.

Esimerkki 10. Sun Java System Access Manager -asetukset web-sovelluksessa.

```
<filter>
    <filter-name>Agent</filter-name>
    <filter-class> com.sun.identity.agents.filter.AmAgentFilter
</filter-class>
</filter>

<filter-mapping>
    <filter-name>Agent</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Esimerkissä 11 käyttämällä `<load-on-startup>`-elementtiä varmistetaan, että FacesServlet ladataan suoritettaessa web-sovellus ensimmäisen kerran. Asiakassovellus asetetaan löytyväksi `palvelimenOsoite/smsClient`-muotoisesta URL-osoitteesta ja kaikki pyynnöt `smsClient`-osan jälkeen osoitetaan FacesServletille.

Esimerkki 11 FacesServlet-asetusten määrittäminen.

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/smsClient/*</url-pattern>
</servlet-mapping>
```

JavaserverFacesin asetustiedostossa `faces-config.xml` listataan sovelluksessa käytettävät JavaBeans-komponentit ja sovelluksessa käytettävät navigointisäännöt. Sovelluksessa ei hyödynnetty navigointisääntöjen käyttöä, joten tiedostoon listattiin vain käytettävät JavaBeans-komponentit.

Esimerkissä 12 on esitelty `sendSMSBean` JavaBeans-komponentin esittely asetustiedostossa. Esittelyssä asetetaan JavaBeans-komponentille JSP-sivuilla käytettävä nimi, komponentin toteuttava luokka ja komponentille asetettu näkyvyys. Näkyvyys on `session`, joten komponentin tila tallennetaan HTTP-istuntoon ja se säilyttää sille asetetut arvot istunnon ajan. Näin komponentin arvot ja metodit ovat käytettävissä kaikilla JSP-sivuilla. Esimerkissä 8 asetettu käyttöliittymäkomponenttien tiedon tallettaminen ei vaikuta JavaBeans-komponenttien näkyvyyteen.

Esimerkki 12: JavaBeans-komponentin määrittely.

```
<managed-bean>
  <managed-bean-name>sendSmsBean</managed-bean-name>
  <managed-bean-class>backingBeans.SendSmsBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

6.3 JSP-sivut

Tässä luvussa käsitellään JSP-sivujen käyttöä asiakassovelluksessa esimerkkien kautta. Asiakassovellus koostui useista JSP-sivuista, kuitenkaan kaikkia sivuja ei esitellä, vaan esimerkkien pohjana käytetään `sendSMS.jsp`-sivua. Tarkoituksena on esitellä miten asiakassovelluksen toteuttamisessa käytettiin JSP- ja JavaServer Faces -teknologioita.

6.3.1 JSF- ja RichFaces-käyttöliittymäkomponentit

Asiakassovelluksessa käytetyillä JSP-sivuilla yhdistellään JSF- ja RichFaces-tageja käyttöliittymän toteuttamiseksi. RichFaces-komponenttikirjastolla lisättiin web-käyttöliittymän toimintoihin AJAX-toiminnallisuuksia. AJAX-toiminnallisuuksilla parannettiin käyttöliittymän käytettävyyttä, suorittamalla asynkronisia tiedonsiirtoja selaimen ja palvelimen välillä.

JSF- ja RichFaces-tagikirjastot esitellään JSP-sivulla käyttäen esimerkissä 13 käytettyä esittelyä. Esittelyn jälkeen JSF- ja RichFaces-käyttöliittymäkomponentteja voidaan käyttää sijoittamalla niiden tageja JSP-sivulle.

Esimerkki 13 JSF- ja RichFaces -tagikirjastojen esittely JSP-sivulla.

```
<!-- JavaServer Faces -tagikirjaston esittely -->
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<!-- RichFaces-tagikirjaston esittely -->
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
```

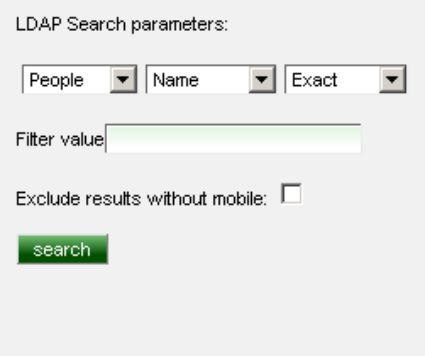
Kuvassa 13 on näytetty JSF- ja RichFaces-tageilla toteutettu lomake LDAP-protokollaa käyttävien hakujen suorittamiseksi hakemistopalvelimella. RichFaces-tageja käytetään toteuttamaan lomakkeeseen AJAX-toiminnallisuuksia. Seuraavaksi käydään lomakkeen toimintaa kahden esimerkin kautta. Lomakkeessa käytettyä `SendSMSBean` JavaBeans-komponenttia tarkastellaan luvussa 6.3.2.

Riveillä 7-13 esitellyn alasvetovalikon `onchange`-tapahtumaan liitetään riveillä 9-11 suorittettava AJAX-kutsu. Kutsu tarkistaa alasvetovalikon valinnan, käyttäen

`SendSMSBean` JavaBeans-komponentin metodia, joka päivittää riveillä 14-18 esitellyn alasvetovalikon `disabled`-arvon.

Riveillä 32-42 esitellään lomakkeen lähettämiseen käytettävä painike. Painettaessa painiketta suoritetaan `SendSMSBean` JavaBeans-komponentin metodi, joka kokoaa lomakkeen alasvetovalikkojen, tekstikentän ja valintalaatikon arvoilla suoritettavan haun hakemistopalvelimelle ja suorittaa sen.

Kuvassa 14 esitetään miltä lomake näyttää käyttäjän selaimessa.



Kuva 14. Lomake LDAP-hakujen suorittamiseen.

```
<h:outputText value="LDAP Search parameters:"/>
<a4j:form id="ldapQuery" >
    <h:selectOneMenu id="filter1SelectOneMenu"
                     value="#{sendSmsBean.treeSelection}">
        <f:selectItems value="#{sendSmsBean.treeFilters}"/>
    </h:selectOneMenu>
    <h:selectOneMenu id="filter2SelectOneMenu"
                     value="#{sendSmsBean.attrSelection}">
        <a4j:support event="onchange"
                     action="#{sendSmsBean.checkAttrOperatorSelectOneMenu}"
                     reRender="filter3SelectOneMenu"/>
        <f:selectItems value="#{sendSmsBean.attributeFilters}"/>
    </h:selectOneMenu>
    <h:selectOneMenu id="filter3SelectOneMenu"
                     value="#{sendSmsBean.attrOperator}"
                     disabled="#{sendSmsBean.disableAttrOperatorSelectOneMenu}">
        <f:selectItems value="#{sendSmsBean.attributeOperators}"/>
    </h:selectOneMenu>
    <h:outputText value="Filter value"/>
    <h:inputText id="searchString"
                 size="25"
                 value="#{sendSmsBean.inputText}">
        <rich:hotKey id="filterValueEnter"
                     selector="#ldapQuery"
                     key="return"
                     handler="#{rich:element('queryLDAPButton')}.click()" />
    </h:inputText>
    <h:outputText value="Exclude results without mobile: "/>
    <h:selectBooleanCheckbox id="excludeNoMobile"
                             value="#{sendSmsBean.excludeNoMobiles}">
    </h:selectBooleanCheckbox>
    <a4j:commandButton id="queryLDAPButton"
                       value="search"
                       action="#{sendSmsBean.queryLDAP}"
                       reRender="ldapTable">
        <a4j:support event="onclick"
                     action="#{sendSmsBean.showSearching}"
                     reRender="hideShow, queryLDAPButton" />
        <a4j:support event="oncomplete"
                     action="#{sendSmsBean.hideSearching}"
                     reRender="hideShow, queryLDAPButton, ldapQueryResults" />
    </a4j:commandButton>
    <a4j:outputPanel id="hideShow">
        <h:graphicImage value="images/ajax-loader2.gif"
                        height="12"
                        width="12"
                        rendered="#{sendSmsBean.search}"/>
    </a4j:outputPanel>
</a4j:form>
<h:outputText id="ldapQueryResults"
              value="#{sendSmsBean.ldapQueryResultText}"/>
```

Kuva 13. JSF- ja RichFaces-tageilla toteutettu lomake.

6.3.2 JavaBeans-komponentin käyttö

Luvussa 6.3.1 käsitellyn lomakkeen käyttöliittymäkomponentit saavat arvonsa `SendSMSBean` JavaBeans-komponentin jäsenmuuttujista ja käyttävät komponentille määriteltyjä metodeita käyttöliittymän toimintojen suorittamiseen.

Kuvassa 15 näytetään lomakkeen käyttämiä `SendSMSBean` JavaBeans-komponentin jäsenmuuttujien get- ja set-metodeita ja hakemistopalvelimeen suorittavan haun metodi. Seuraavaksi käydään läpi metodien toimintaa kolmella esimerkillä.

Riveillä 1-2 on `SendSMSBean` JavaBeans-komponentin `treeSelection`-jäsenmuuttujan set-metodi. Lähetettäessä lomake, kutsutaan jäsenmuuttujan set-metodia, joka asettaa jäsenmuttujan arvoksi lomakkeen alasvetovalikosta valitun arvon.

Riveillä 4-5 on `SendSMSBean` JavaBeans-komponentin `disableAttrOperatorSelectOneMenu`-jäsenmuuttujan get-metodi. Metodia kutsutaan, kun luvun 6.3.1 kuvan 13 riveillä 14-18 olevan alasvetovalikon tila päivitetään.

Riveillä 38-51 on `sendSMSBean` JavaBeans-komponentin `queryLDAP`-metodi, jota kutsutaan lomakkeen lähettämiseksi. Metodissa kootaan lomakkeen alasvetovalikkojen, tekstikentän ja valintalaatikon arvoilla hakemistopalvelimessa suoritettava haku.

```
1  public void setTreeSelection(String treeSelection){
2      this.treeSelection = treeSelection;}
3
4  public String checkAttrOperatorSelectOneMenu(){
5      if(attrSelection.compareTo("free") == 0){
6          disableAttrOperatorSelectOneMenu = true;
7      }else{
8          disableAttrOperatorSelectOneMenu = false;}
9      return "";}
10
11 public boolean getDisableAttrOperatorSelectOneMenu(){
12     return disableAttrOperatorSelectOneMenu;}
13
14 public String queryLDAP(){
15     LDAPConnector ldapConnector = new LDAPConnector();
16     if(attrSelection.compareTo("free") == 0){
17         ldapConnector.setLDAPTree(treeSelection);
18         ldapConnector.setFreeLDAP(inputText);
19         ldapConnector.setExcludeNoMobiles(excludeNoMobiles);
20     }else{
21         ldapConnector.setLDAPTree(treeSelection);
22         ldapConnector.setFilter(attrSelection, inputText, attrOperator);
23         ldapConnector.setExcludeNoMobiles(excludeNoMobiles);}
24     setLdapResultslist(ldapConnector.Connect());
25     ldapQueryResultText = (new StringBuilder()).append("Query produced ").
26     append(ldapResultslist.size()).append(" results.").toString();
27     return null;}
```

Kuva 15. JavaBeans-komponentin metodeja.

6.4 DataTransfer-kirjasto

Asiakassovellukseen toteutettu DataTransfer-kirjasto on kokoelma luokkia, joita käytetään luomaan yhteyksiä ulkoisten järjestelmien ja sovelluspalvelimessa suoritettavan asiakassovelluksen välille. Ulkoisina järjestelminä asiakassovelluksessa käytettiin palvelinsovellusta ja TAMKin hakemistopalvelinta.

6.4.1 Hakemistopalvelin

Hakemistopalvelimelle suoritetaan hakuja käyttämällä LDAP-protokollaa, haun tuloksista voidaan koostaa asiakassovelluksesta lähetettävien tekstiviestien vastaanottajalistoja. Luvun 6.3.1 kuvassa 14 on esitetty asiakassovelluksessa

käytettävä lomake hakujen suorittamiseksi. Haut asiakassovelluksen käyttämään hakemistopalvelimeen suoritetaan käyttäen Java-ohjelmointikielen JNDI-rajapintaa haun suorittamiseksi.

6.4.2 Palvelinsovellus

Yhteys palvelinsovellukseen toteutetaan käyttäen Apace XML-RPC –asiakaskirjastoa. Jotta palvelinsovelluksen XML-RPC-palvelimen metodeita voidaan kutsua, on XML-RPC-asiakkaan tiedettävä palvelimen tarjoamat metodit. Asiakkaassa käytetään samaa `DBQuery`-rajapintaluokkaa kuin palvelimessa, joka esiteltiin luvussa 4.3.1. Asiakassovelluksesta kutsutaan palvelinsovelluksen XML-RPC-palvelimen metodeita SMS-viestien lähettämiseksi ja palvelinsovelluksen tietokannan tietojen näyttämiseksi asiakassovelluksessa.

6.5 Käyttäjän tunnistaminen

Käyttäjän tunnistamisen ja pääsyn hallinnan toteuttamiseen asiakassovelluksessa käytettiin sovelluspalvelimella Sun Java System Access Manageria. Selaimella navigoitaessa URL-osoitteeseen, josta asiakassovellus löytyy, ohjataan käyttäjä kirjautumissivulle ellei käyttäjä ole jo kirjautunut sisään järjestelmään. Kirjautumisen jälkeen käyttäjä uudelleen ohjataan asiakassovelluksen tarjoavalle web-sivulle.

Käyttäjän tunnistaminen mahdollistaa käyttäjäkohtaisten kontaktilistojen käyttämisen, kirjautumisen jälkeen asiakassovellus noutaa palvelinsovelluksen tietokannasta käyttäjän luomat kontaktilistat. Joita voidaan käyttää koostamaan SMS-viestien vastaanottaja listoja.

7 Yhteenveto

Työssä toteutettu järjestelmä tarjosi monipuolisen ja haastavan aiheen opinnäytetyölle, sen käsitellessä web-sovelluksia, tiedonsiirtoa ja laitekohtaisia ominaisuuksia. Työ sisälsi paljon itsenäistä työtä ja esitutkimusta työssä tarvittavien tekniikoiden käyttämiseksi. Varsinkin Java EE ja sen tarjoamat tekniikat olivat täysin uusia järjestelmän toteuttajalle ja näin sitoivat paljon aikaa järjestelmän toteuttamisessa.

Gammu-projektin tarjoama SMS Daemon -sovellus ja MySQL:n tukeminen käytettävänä tietovarastona sopivat hyvin järjestelmän tarkoituksiin. Gammu-projektin saadessa tasaisesti päivityksiä, sen käyttäminen myös pitkällä tähtäimellä oli perusteltua. Gammu-projektin tarjoama mahdollisuus lähettää myös moniosaisia SMS-viestejä oli kaivattu ominaisuus järjestelmässä.

Gammu-projektista on riittävästi dokumentaatiota ja saatavilla on myös listaus, jossa Gammu-projektin käyttäjät ovat listanneet testaamiaan puhelinmalleja. Puhelinmallista on mainittu käytetty yhteysprotokolla ja ominaisuudet, jotka ovat onnistuneesti testattu sillä kokoonpanolla. Tästä oli suuri apu testattaessa eri laitteita ja kaapeleita.

Esitutkimuksen viedessä paljon aikaa, toteuttamisessa ei ehditty tekemään kunnollisia testauksia järjestelmälle, vaan testaaminen tapahtui järjestelmää käyttämällä. Työn aikana kertyi arvokasta kokemusta järjestelmässä käytetyistä tekniikoista.

Järjestelmän toteuttaminen antoi hyvän kuvan mitä vaaditaan toteuttaessa monista eri osista koostuvia järjestelmiä. Suunnittelu on hyvin tärkeässä asemassa, jolloin vältytään aikaa vieviltä korjaavilta ratkaisuilta. Työn tuloksena syntyi järjestelmän konsepti, jota jatkokehittämällä voidaan toteuttaa yleiseen käyttöön tarkoittettu järjestelmä.

Lähteet

/1/ Oracle Corporation. Lesson: All About Sockets. [www-sivu] [Viitattu 29.5.2010]. Saatavilla: http://java.sun.com/docs/books/tutorial/networking/sockets/definition.html

/2 / Teknillinen Korkeakoulu Tietokoneverkot-kurssin aineisto [online] [viitattu 29.5.2010]. Saatavilla: http://www.tml.tkk.fi/Opinnot/T-110.4100/2007/Luennot/tkv20070914_4.pdf

/3/ XML-RPC Home Page. [www-sivu]. [viitattu 21.5.2010]. Saatavilla: http://www.xmlrpc.com/

/4/ Westerholm, Mika & Kyppö, Jarmo. 2006. Java-Ohjelmointi. Helsinki: Talentum.

/5/ Geary, David. Simply Singleton. [www-sivu]. [viitattu 30.5.2010]. Saatavilla: http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html

/6/ Jendrock, Eric; Ball, Jennifer; Carson, Debbie; Evans, Ian; Fordin, Scott; Haase, Kim. The Java EE 5 Tutorial. [www-sivu] [viitattu 21.5.2010]. Saatavilla: http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

/7/ RichFaces Developer Guide. [www-sivu]. [viitattu 22.5.2010]. Saatavilla: http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/

/8/ Wammu and Gammu. [www-sivu]. [viitattu 23.5.2010]. Saatavilla: http://wammu.eu/gammu/

/9/ European Telecommunications Standards Institute. GSM Technical Spefication Version 5.3.0. July 1996. [pdf-tiedosto]. [viitattu 2.6.2010]. Saatavilla: ftp://ftp.infradead.org/pub/gsm/GSM-03.40.pdf

/10/ Nagios Enterprises. Nagios Core Documentation. 16.6.2009. [www-sivu]. [viitattu 24.5.2010]. Saatavilla: http://nagios.sourceforge.net/docs/3_0/

/11/ Gröning, Harri & Hernberg, Ilja. Nagios verkonvalontaohjelmiston asennus ja käyttöönotto Laurea-ammattikorkeakoulussa. Laurea-ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Leppävaara 2009. 50 s. + 32 liites.

Liitteet

Liite 1: SMS Daemon -asetustiedosto

```
# Gammu configuration, this section is like section "gammu" in
"gammurc" file,
# see gammurc(5) for documentation.
[gammu]
port = /dev/ttyACM0
#model = 6110
connection = fbuspl2303
#synchronizetime = yes
#use_locking = yes
#gammuloc = gammu.us
#startinfo = yes

# When uncomment this section and insert numbers here, smsd will
process
# incoming sms only from numbers written here (incoming sms from
all other
# numbers will be deleted)
#[include_numbers]
#number1 = 1234

# When uncomment this section and insert numbers here, smsd will
process
# incoming sms from all numbers not written here (incoming sms from
numbers
# written here will be deleted). This is "black" list.
# Note: after using "include_numbers" section this one will be
ignored
#[exclude_numbers]
#number1 = 1234

# General SMSD settings, see gammu-smsdrc(5) for detailed
description.
[smsd]
# SMSD service to use, one of FILES, MYSQL, PGSQL, DBI
service = MYSQL
# PIN for SIM card
PIN = ********
# File (or stderr, syslog, eventlog) where information will be
logged
#logfile = smsdlog
# Amount of information being logged, each bit mean one level
debuglevel = 0
# Configuration for using more phones on same database
phoneid = Tietokonekeskus3120
# Script to be executed when new message has been received
#runonreceive = /some/script
# Commication frequency settings
commtimeout = 30
```

```
sendtimeout = 30
#receivefrequency = 0

# Phone communication settings
#checksecurity = 1
#resetfrequency = 0

# Delivery report configuration
#deliveryreport = no
#deliveryreportdelay = 10

# Ignoring broken SMSC
#skipsmscnumber = +48602123456

# Database backends congfiguration
user = ********
password = ********
pc = localhost
# pc can also contain port or socket path after colon (eg.
localhost:/path/to/socket)
database = sms
```

Liite 2: SMS Daemonin käyttämien taulujen rakennekaaviot