Sami Suuriniemi

LÄMPÖTILAN VALVONTA- JA HALLINTAJÄRJESTELMÄ

Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto
2010

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

LÄMPÖTILAN VALVONTA- JA HALLINTAJÄRJESTELMÄ

Suuriniemi, Sami
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Toukokuu 2010
Ohjaaja: Aarinen, Reino
Sivumäärä: 29
Liitteitä:

Asiasanat: JSF, Java EE, MySQL, Digitemp

Työn tarkoitus on rakentaa ohjelmisto digitemp-mittareiden hallintaan ja lämpötilojen valvontaan. Satakunnan ammattikorkeakoululla on käytössä lämpömittareita palvelintiloissa lämpötilojen tarkkailua varten. Tällä hetkellä jokainen mittaritietokone on erillään toisistaan ja hälytysjärjestelmä on jokseenkin alkeellinen sekä hankala hallita.

Ohjelmisto koostuu kolmesta osasta. Asiakasohjelmasta mittarin päähän, palvelinohjelmasta palvelimelle sekä nettikäyttöliittymästä.

Ohjelma toimii siten, että mittarikone lähettää tietoa palvelimelle, joka tallennetaan tietokantaan ja sitä hallitaan ja luetaan web:n kautta. Järjestelmä vaatii käyttäjätunnuksen, johon liitetään puhelinnumero sekä sähköposti. Kun lämpötila nousee yli sallitun rajan aiheutuu hälytys, jolloin lähtee sähköposti ja tekstiviesti mittariin liitetyille henkilöille. Web-käyttöliittymästä voi hallita käyttäjiä sekä katsoa graafista historiaa lämpötiloista.

A TEMPERATURE MONITORING AND MANAGING SYSTEM

Suuriniemi, Sami
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
May 2010
Supervisor: Aarinen, Reino
Number of pages: 29
Appendices:

Keywords: JSF, Java, Digitemp

The purpose of this work was to build an application to manage Digitemp probes and to monitor temperatures. Satakunta University of Applied Sciences uses Digitemp probes to monitor temperatures in server rooms. Currently each computer is apart from each other and the alarm system is somewhat rudimentary and also hard to manage.

The application consists of three parts. These are a client program to probe computers, a server program for the server and a web interface.

The client will send data to the server which saves it into the database. The data is then viewed and controlled through the web interface. The system requires users. Every user has an e-mail address and a phone number. When the temperature of a probe rises above the allowed limits, the system sends an alarm through e-mail and an SMS to users linked with the probe. The web interface is used to manage users and view the temperature data.

SISÄLLYS

1 JOHDANTO

Työn lähtökohtana oli Satakunnan ammattikorkeakoulun (SAMK) lämpötilavalvonnan keskittäminen. SAMK:ssa käytetään lämpötilojen valvontaan DS1820-tyyppisiä lämpötila-antureita, jotka liitetään tietokoneen väylään. Tällä hetkellä jokainen tietokone, johon antureita on liitetty, pitää konfiguroida ja hoitaa erikseen sekä tarvitsee erillisen valvontasovelluksen. Anturit valvovat mm. palvelintilojen lämpötiloja, joissa esimerkiksi jäähdytysjärjestelmän pettäminen voi nopeasti johtaa kalliisiin kustannuksiin hajonneiden laitteiden vuoksi. Näiden antureiden täytyy olla ns. hälyttäviä, eli lämpötilan mennessä rajojen yli, hälyttää järjestelmä tekstiviestillä sekä sähköpostilla. Keskitetty hallinta ja valvontasovellus vähentää huomattavasti työtä, joka kuluu järjestelmän ylläpitoon.

Ohjelman tarkoitus on toimia niin, että tietokoneisiin, joihin lämpötila-anturit kytketään, asennetaan vain ohjelma, joka lähettää lämpötilat verkon yli palvelimelle. Palvelin prosessoi lämpötilat sekä hallitsee tietokantaa. Asiakaskoneiden asetuksia pitäisi myös pystyä hallitsemaan verkon ylitse verkkosivuilta.

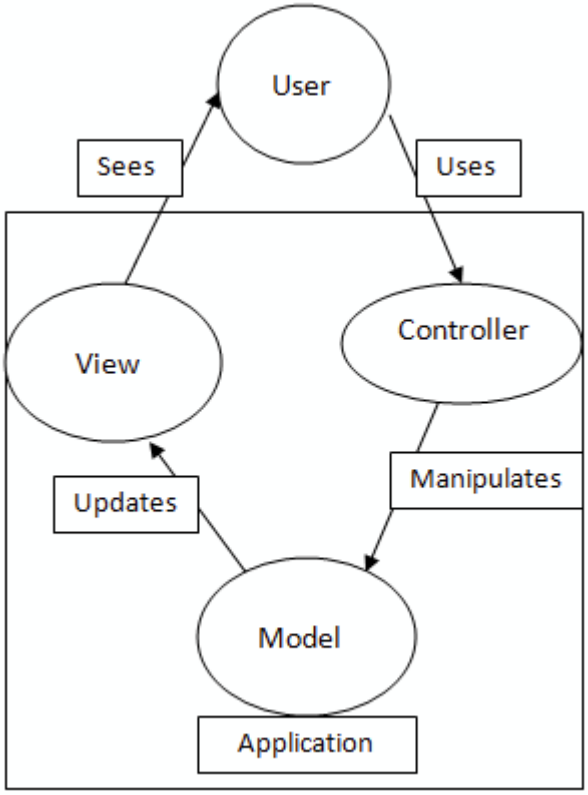
2 KÄYTETTÄVÄT TEKNIIKAT

2.1 Java

2.1.1 Java Server Faces

Java Server Faces on sovelluskehys Java-pohjaisten web-sovellusten luomiseksi. Kehys tarkoittaa nimensä mukaisesti valmista ohjelmistorunkoa, jonka päälle sovellus ohjelmoidaan. Sovelluskehysten käyttö helpottaa, selventää ja nopeuttaa sovellusten tuottamista. Sovelluskehyksen käyttö auttaa myös pitämään ohjelmiston runkoa selvempänä. JSF on pyyntöpohjainen MVC-arkkitehtuuria noudattava sovelluskehys. MVC tulee sanoista Model-View-Controller ja se tarkoittaa ohjelmiston rungon rakentamista nimiensä mukaisiin osiin. MVC-arkkitehtuurin tarkoituksena on jakaa

ohjelmisto osiin siten, että käyttöliittymä ja sovelluksen sisäiset toiminnot sijaitsevat erillään. View eli näkymä on web-sovelluksissa se, mitä käyttäjä näkee selaimellaan. Näkymältä lähetetään pyyntöjä ohjaimelle, controller, joka käsittelee pyynnöt ja ohjaa oikeille sivuille. Model eli malli sisältää business-logiikan eli mm. tiedon käsittelyn, tallennuksen sekä tietokantayhteydet.



Kuva 1. MVC-arkkitehtuurin kuvaus

JSF:n versiota 2 aikaisemmat versiot käyttivät oletuksena käyttöliittymän näkymien esittämiseen Java Server Pages, JSP, -sivuja. JSF 2.0 käyttää oletuksena Facelet-tekniikkaa. Facelet-tekniikka on avoimen lähdekoodin web ohjelmistokehys. Tekniikka vaatii validia XML -tekniikkaa toimiakseen. Tämän vaatimuksen vuoksi sivujen pitää olla validia XHTML formaattia.

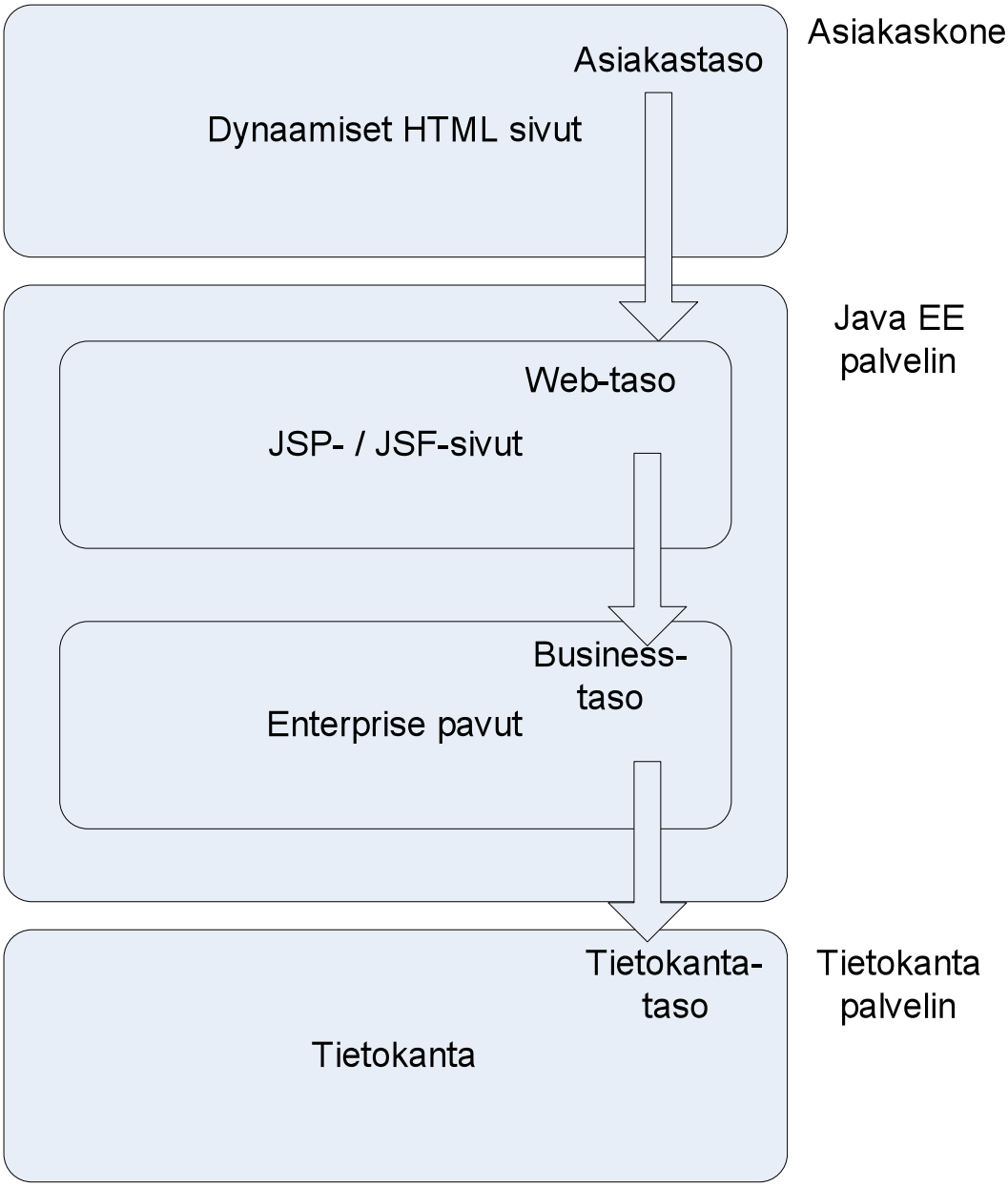
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Tempcontrol</title>
    </h:head>
    <h:body>
      <h:outputText value="Hei Maailma"/>
    </h:body>
</html>
```

Esimerkki JSF merkkauksesta

Työssä päädyttiin käyttämään JavaServer Faces tekniikkaa, koska JSF merkkaus on selväkielistä ja helposti ylläpidettävää, vaikka ohjelmistopuoli olisi hieman oudompaa. /2, s.261/

2.1.2 Java Enterprise Edition

Java Enterprise Edition tai Java EE on kehitysalusta monitasoisten Java -palvelinsovellusten kehittämiseen ja ajamiseen. Ennen versiota 2.0 Java EE:stä käytettiin nimityksiä Java 2 Platform, Enterprise Edition ja J2EE. Java EE on siis sarja ohjelmointirajapintoja monitasoisten komponenttipohjaisten ohjelmistojen rakentamiseen. Yleisesti Java EE ohjelmistojen toteutus jakautuu neljään tasoon: asiakastaso, web- ja business-taso ja tietokanta-taso. Asiakastason komponentit ajetaan asiakkaan koneella, kuten esimerkiksi verkkoselaimen dynaamiset komponentit, esim. javascript. Web-tason komponentteihin kuuluu mm. JSP- ja JSF –sivut. Business-tasossa on ohjelman yleinen logiikka, pavut. Web- ja business-tason komponentit ajetaan Java EE palvelimella. Tietokantasoon kuuluu mikä tahansa tietojen tallennuskohde, esim. relaatiotietokanta. /1/ /2, s.1/



Kuva 2. Java EE tasot

2.1.3 Java Persistence API

Java Persistence API eli JPA on tietokantarajapinta, jota käytetään tiedon pysyvään tallennukseen Java EE tai Java SE -pohjaisissa ohjelmissa. JPA on Javan standardirajapinta tietokantojen käyttämän relaatiomallin sekä olio-ohjelmoinnin oliomallin yhdistämiseksi. Nykyään yleistyneessä tekniikassa rakennetaan ns. objekti-relaatio-mapping -kerros tietokannan päälle, jonka avulla tietokantaa käytetään kuin mitä tahansa oliota. JPA:ssa näitä olioita kutsutaan entiteeteiksi, joka on kevyt Java-luokka,

jonka tila tyypillisesti säilytetään relaatiotietokannassa. Entiteetin instanssit vastaavat relaatiotietokannan taulun yhtä riviä. /1/ /2, s.41/

```
<h:outputText value="Käyttajätunnus" />
<h:outputText value="#{UsersController.user.username}" />
<h:outputText value="Nimi" />
<h:outputText value="#{UsersController.user.name}" />
<h:outputText value="Sähköposti" />
<h:inputText value="#{UsersController.user.email}" />
<h:outputText value="Puhelin" />
<h:inputText  value="#{UsersController.user.phone}" />
<h:commandButton action="#{UsersController.saveUser}" val-
ue="Tallenna" />
```

JPA entiteetin käyttö sivulla

```
public String saveUser() {
    users.edit(user);
    return "profile";
}
```

JPA entiteetin lähetys sivulta eteenpäin mallille

```
public void edit(Users users) {
    em.merge(users);
}
```

JPA entiteetin päivitys tietokantaan

2.2 MySQL

MySQL on erittäin suosittu, ilmainen ja alustasta riippumaton SQL- relaatiotietokannan hallintajärjestelmä. MySQL on saatavilla vapaalla GNU GPL tai kaupallisella lisenssillä. GNU General Public Licence, GNU GPL, on vapaa ohjelmistolisenssi. GPL antaa käyttäjälle mahdollisuuden kopioida, muuttaa ja jakaa edelleen ohjelmia ja niiden lähdekoodia. MySQL-tietokanta on hyvin suosittu web-sovellusten tietokantana varsinkin aloittelevien ohjelmoijien käytössä suuren materiaalin ja täten helpon avunsaannin vuoksi. MySQL:ää hyödyntää suurista yrityksista mm. Google, Wikipedia ja Yahoo. /3/

2.3 Python

Python on monipuolinen, tulkattava ohjelmointikieli, jonka kehitysfilosofia painottaa koodin luettavuutta. Pythonin syntaksi on helppoa ymmärtää ja sitä suositellaan monesti ensimmäiseksi ohjelmointikieleksi /4/. Pythonin vakiokirjasto on myös todella laaja ja sillä on erittäin helppoa ja yksinkertaista toteuttaa monta asiaa, jotka esimer-

kiksi C kielellä vaativat asioiden syvällistä tuntemista. Työssä käytettiin Pythonia verkkoprotokollan ja palvelimen sekä asiakaskoneiden taustaprosessien luomiseen. Python valittiin juuri sen helpon luettavuuden vuoksi, joka auttaa järjestelmän ylläpitäjää mm. lisäämään ominaisuuksia tarpeen mukaan, ilman syvää ohjelmoinnin asiantuntemista. Koska Python on tulkattava kieli, on sitä myös helppo testata komentotulkilla.

```
def hello(s):
      print "Hello,", s

hello(”sami”)
```

Esimerkki Pythonin syntaksista

2.4 Digitemp

Digitemp on ohjelma, jolla mitataan ja näytetään tietokoneen väylään liitettyjen antureiden lämpötiloja. Tietokoneeseen liitetyt anturit ovat DS1820-tyyppisiä ja ne toimivat one-wire väylässä. Digitemp on yksinkertainen ohjelma, jolla ensin haetaan tietokoneeseen liitetyt anturit ja tämän jälkeen ohjelma on valmis tulostamaan lämpötilan anturilta. Työssä käytetään Digitemp-ohjelmaa mittareiden lukemiseen sen yksinkertaisuuden ja helppouden vuoksi.

2.5 Gammu

Gammu on vapaa, GNU GPL lisenssin alla julkaistu ohjelmisto puhelimien monien toimintojen hallintaan /5/. Gammulla pystyy mm. lähettämään ja vastaanottamaan tekstiviestejä, soittamaan ja vastaanottamaan puheluita, käyttämään puhelimen kalenteria sekä tiedostojärjestelmää. Työssä käytetään Gammua tekstiviestihälytysten tekemiseen.

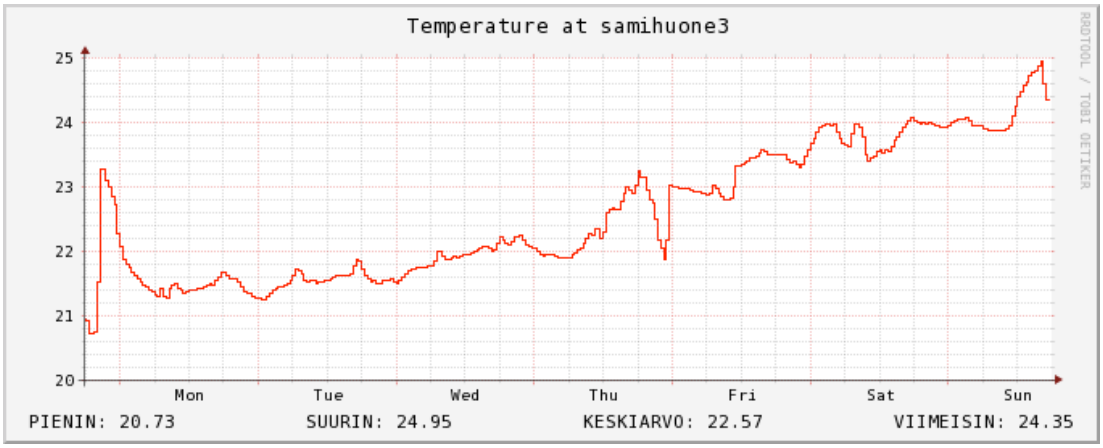
2.6 Glassfish

Glassfish on avoimen lähdekoodin sovelluspalvelin. Glassfish-projektia johtaa Sun Microsystems ja se on tarkoitettu Java EE alustalle /4/ /2, s.31/. Glassfish valittiin työhön sovelluspalvelimeksi sen vapauden sekä hyvän Java EE alustan yhteensopi-

vuuden vuoksi. Glassfish:lle on myös laajat ja kattavat dokumentaatiot. Glassfish sisältää Apache Derby tietokannan. Derby on myös avoimen lähdekoodin projekti, jota olisi voinut tässä työssä käyttää /2, s.29/. Derbyn sijaan kuitenkin valittiin tietokannaksi MySQL, koska työssä piti ottaa huomioon sovelluksen ylläpitäjän mahdollinen tietotaso. Asiakas myös toivoi käytettävän MySQL tietokantaa.

2.7 RRDtool

RRDtool, Round Robin Database tool, on sovellus, joka on kehitetty aika-sarjaisen informaation kuvaamiseen. Informaatio talletetaan kiertävään tietokantaan ja tästä informaatiosta piirretään kuvaajaa. RRDtool kehitettiin ensisijaisesti verkon solmupisteiden tarkkailuun, mutta nykyään se on käyttössä lähes minkä tahansa aikasarjaisen informaation, kuten lämpötilojen, verkon kapasiteetin ja prosessorin käytön esittämiseen graafisesti. Työssä käytetään RRDtool:ia lämpötilojen kuvaamiseen ja lämpötilatiedon tallennukseen.



Kuva 3. Esimerkki RRDTool:lla tehdystä kuvaajasta

3 SUUNNITTELU

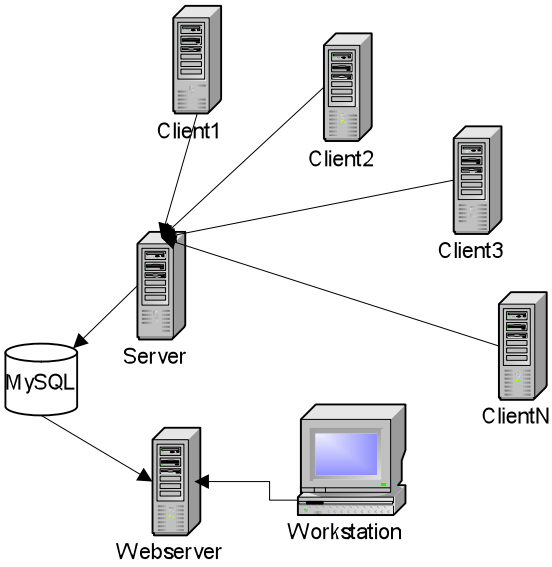
Ohjelmistotuotannossa hyvä suunnittelu on sulavan tuotannon kannalta tärkein osa. Ohjelman kehityksessä oli tarkoitus noudattaa ohjelmistotuotannon V-Mallia, jossa on tarkoituksena ensin suunnitella koko ohjelma ja sen jälkeen toteuttaa se. Uuden ohjelmoijan on tosin käytännössä mahdotonta noudattaa ohjelmistotuotannon V-mallia kokemuksen ja puutteellisen osaamisen vuoksi. Uuden ohjelmoijan on erittäin

vaikea onnistua suunnittelussa täydellisesti ilman, että on aikaisempaa kokemusta kyseisenlaisen ohjelman kehityksessä. Tästä johtuen ohjelmiston tuotannossa noudatettiin ns. ketterää menetelmää, jonka perustana on useat pienet kehitysiteraatiot. Jokaiseen iteraatioon kuuluu suunnittelu, toteutus ja testaus ja tämän jälkeen katsotaan, onko iteraatio onnistunut vai epäonnistunut, jonka jälkeen aloitetaan sykli uudelleen.

3.1 Määrittely

Ohjelmiston toiminnallinen määrittely on vaihe, jossa ohjelman vaatimukset käydään läpi ja listataan asiakkaan kanssa. Määritellyssä dokumentissa tulee olla selväkielisesti kaikki asiat, mitä ohjelmistosta halutaan löytyvän. Määrittelyn tulisi olla mahdollisimman täydellinen ohjelmiston suunnittelua varten. Mitä pidemmälle ohjelmiston kehitys on ehtinyt kun määrittelyä aletaan muuttamaan, sitä enemmän aikaa muutoksiin yleensä kuluu ja sitä kallimmaksi ohjelmiston tuotanto tulee. Määrittely kuvaa, mitä valmiin järjestelmän pitää pystyä tekemään, ei miten asiat pitää järjestelmän sisällä tehdä.

Tekninen määrittely tehdään toiminnallisen määrittelyn pohjalta, sen tarkoituksena on kuvata ohjelmiston tekninen arkkitehtuuri tarkasti. Tekninen määrittely kattaa kaikki komponentit, joilla toiminnallisen määrittelyn ominaisuudet saadaan toteutettua. Teknisestä määrittelystä löytyy mm. käytetyt ohjelmointikielet, ohjelmistokomponentit, tietorakenteet ja niiden väliset rajapinnat. Tekninen määrittely pitää jakaa moneen tasoon laajemmissa järjestelmissä, joista esimerkiksi ylimmällä tasolla kuvataan järjestelmän eri ohjelmien väliset suhteet.



Kuva 4. Ohjelmiston karkea kuvaus

4 ASIAKASSOVELLUS

Asiakasohjelma toimii Linux-palvelimella taustasovelluksena, joka lukee lämpötila-antureilta arvoja tietyin väliajoin ja lähettää ne palvelimelle. Anturi toimii one-wire tekniikalla, jonka vuoksi ohjelma on toteutettu niin, että samassa tietokoneen portissa voi olla useampia antureita. Sovellus on myös suunniteltu toimimaan millä tahansa portilla, johon anturi voidaan laittaa, näin yhteen sovelluskoneeseen voi sijoittaa mittavan määrän antureita. Sovellus käynnistyy lukemalla asetukset, jos asetukset ovat virheelliset tai niitä ei ole, ohjelma ilmoittaa tästä ja sammuu.

```
class Client(Daemon):
 def run(self):
  self.readConfs()
  self.createLogger()

  self.logger.write("Client: Started up", 1)
  self.introduced = False

  self.probes = {}
  self.makeConnection()
  self.logger.write("Initializations ready", 2)
  for probe in self.confs['probes']:
   self.createProbe(probe)
  while 1:
   try:
    time.sleep(5)
    if not self.connection:
     self.logger.write("Client: Connection has failed,  restarting",
1)
     self.makeConnection()
```

Asetuksien lukemisen jälkeen sovellus avaa socketin palvelimelle ja jos yhteys saadaan avattua, lähettää se sinne tervehdysviestin. Palvelimen pitää vastata tervehdykseen oikein, jotta ohjelma jatkaa toimintaa. Tämän jälkeen ohjelma haarautuu ja muodostaa oman säikeen jokaiselle anturille ja nämä jäävät kyselemään tietyin väliajoin lämpötilaa.

```
def run(self):
  while 1:
   try:
    if self.parent.connection:
     self.updateTemp()
     if float(self.temp) != 85.0:
      self.parent.sender.updateData(self.name, self.temp)
      self.logger.write("Probe: Got temp:%(temp)f for
probe:%(name)s"
            % {'temp': self.temp, 'name': self.name}
     else:
      self.logger.write("Probe: Got temp 85.0, digitemp error code",
1)
    time.sleep(int(self.pollrate))
   except ValueError:
    self.logger.write("Probe: Something weird happened, "
            +str(self.temp)+", "+ str(sys.exc_info()), 1)
   except:
    self.logger.write("Probe: Error, " + str(sys.exc_info()[0])
            + ":"+str(sys.exc_info()[1]), 1)
    break
```

4.1 Digitemp

Anturin lämpötilojen lukemiseen käytetään valmista Digitemp sovellusta. Sovelluksen käyttö on yksinkertaista ja tarvitsee ajaa vain kerran asetusten luomista varten. Digitemp luo asetukset tiedostoon erikseen jokaiselle portille. Näihin asetuksiin pystyy määrittelemään mm. halutun tulostusformaatin. Koska digitemp:n käyttö haluttiin mahdollisimman yksinkertaiseksi ja vähän asetuksia vaativaksi, jätettiin kaikki asetukset oletusarvoihinsa ja näitä arvoja parsitaan Python-osassa.

```
TTY /dev/ttyS0
READ_TIME 1000
LOG_TYPE 1
LOG_FORMAT "%b %d %H:%M:%S Sensor %s C: %.2C F: %.2F"
CNT_FORMAT "%b %d %H:%M:%S Sensor %s #%n %C"
HUM_FORMAT "%b %d %H:%M:%S Sensor %s C: %.2C F: %.2F H: %h%%"
SENSORS 1
ROM 0 0x10 0x07 0x9E 0x28 0x00 0x08 0x00 0x71
```

Itse lämpötilan lukeminen tapahtuu yhtä yksinkertaisesti.

```
[root@localhost ~]# digitemp -a -c /etc/digitemp/S0.conf
DigiTemp v3.5.0 Copyright 1996-2007 by Brian C. Lane
```

```
GNU Public License v2.0 - http://www.digitemp.com
May 09 14:58:20 Sensor 0 C: 23.81 F: 74.86
```

4.2 Konfigurointi

Asetusten suunnitellussa, kuten muussakin sovelluksen suunnittelussa, oli tarkoitus tehdä asiat yksinkertaisesti ja siten, että käyttäjälle tulee mahdollisimman vähän vaivaa. Asiaskoneen asetukset ovat pieni poikkeus tähän, koska digitemp tunnistaa anturit erityisen heksa-muotoisen tunnisteen mukaan. Asetusten suunnittelussa tultiin siihen tulokseen, että anturin heksa-tunniste täytyy sisällyttää sovelluksen asetuksiin, jotta anturi voidaan tunnistaa ja liittää haluttuun nimeen. Tämän lisäksi asetuksiin tulevat normaalit asiat, palvelimen IP-osoite, käytettävä portti, lokituksen taso, antureiden käyttämä portti sekä antureiden kyselyväli.

```
/etc/tempcontrol/tempcontrol.conf
[probe1]
serial = S0
pollrate = 30
id = 0x10 0x07 0x9E 0x28 0x00 0x08 0x00 0x71

[Tempcontrol]
loglevel = 5
logfile = /var/log/tempcontrol.log
port = 3434
probes = probe1, probe2
server = ip.address.to.server

[probe2]
serial = S2
pollrate = 30
id = 0x10 0x51 0x32 0x40 0x00 0x08 0x00 0x73
```

4.3 Asennus

Asiakassovelluksen asennus vaatii ainoastaan asetustiedostojen luonnin. Asetustiedostot pitää luoda oikeisiin hakemistoihin ja tämän jälkeen sovellus on valmis käynnistettäväksi. Asennukseen on luotu skripta, joka luo tarvittavat hakemistot sekä tulostaa muut tarvittavat asennuksen tehtävät.

5 PALVELINSOVELLUS

Palvelinsovelluksen tehtävänä on toimia tietokannan ja asiakaskoneiden välillä sekä lähettää hälytys-tekstiviestit ja -sähköpostit. Kun sovelluksen käynnistää, se lukee sekä tarkastaa asetustiedostot ja avaa yhteyden tietokantaan. Kun tietokantaan saadaan yhteys, alkaa sovellus tarkastamaan antureiden asetuksia tietokannasta mahdollisten web-sivuilta tehtyjen muutosten varalta. Saatuaan edelliset asiat tehtyä sovellus muodostaa socketin ja jää kuuntelemaan yhteyksiä. Sovellus muodostaa säikeen jokaiselle yhteydelle. Jokainen säie hoitaa yhdestä asiakaskoneesta tulevan yhteyden ja kaikki sen anturit.

```
class Server(Daemon):
 def run(self):
  self.createConfigs()
  self.createLogger()
  self.createDatabaseLoop()

  self.connection = False
  self.connections = {}
  self.probes = {}
  try:
   while 1:
    if not self.dbl.mysqlConnection:
     self.connections = {}
     self.probes = {}
     self.connection = False
     self.logger.write("Server: Databaseconnection has failed, "
            +"trying to re-establish", 2)
     self.dbl.createDatabaseConnection()
     if not self.dbl.mysqlConnection:
      time.sleep(10)
    elif not self.connection:
     try:
      self.makeConnection()
      self.logger.write("Server: Socket created, listening", 2)
     except (socket.error):
      self.logger.write("Server: Socket error, Socket already in
use", 2)
      time.sleep(10)
    else:
     conn, addr = self.s.accept()
     sc = ServerConnection(conn,addr,self)
     sc.setDaemon(True)
     sc.start()
     self.connections[addr[0]] = {'sc': sc, 'probes': []}
```

Säie myös hallitsee lämpötilojen tarkkailun sekä hälytysten tekemisen. Saatua lämpötilaa verrataan tietokannassa oleviin varoitus- ja hälytys-arvoihin. Jos arvo ylittää hälytysrajan, ajetaan metodit, joilla lähetetään kaikille asianomaisille tekstiviesti sekä sähköposti. Pohjat näille luetaan tietokannasta. Sovellus ei myöskään saa tehdä useampaa hälytystä yhdestä lämpötilan ylityksestä, jolloin tietokantaan päivitetään do-

nealarm lippu. Jos lämpötila on alle hälytysrajan ja donealarm lippu on ylhäällä, otetaan se pois päältä ja palataan normaalitilaan. Jos lämpötila on sallituissa rajoissa ja virhelippu on ylhäällä sekä vastaanotetaan mikä tahansa lämpötila, lasketaan virhelippu alas.

```
   if temp > float(self.parent.probes[name]['ALERT']):
    self.alert(name, temp)
   elif int(self.parent.probes[name]['DONEALARM']):
    self.logger.write("Serverconnection: Temperatures ok for probe
%(probe)s,"
            + "resetting donealarm" % {'probe': name}, 2)
    sql = "UPDATE PROBES SET DONEALARM=FALSE WHERE
PROBE='%(probe)s';" % {'probe': name}
    self.mysql.query(sql)
   elif temp > float(self.parent.probes[name]['WARNING']):
    self.warning(name, temp)
   if int(self.parent.probes[name]['ERROR']):
    self.resetError(name)
```

Sovelluksen tietokantaluokan tehtävänä on tarkkailla tietokantayhteyttä, websovelluksella tehtäviä ja tietokannan kautta tulevia muutoksia sekä anturien virhetilaa. Luokka lukee anturit tietokannasta 5 sekunnin välein, tarkastaa onko päivityslippu, NEEDUPDATE, ylhäällä sekä ajan, jolloin anturi on viimeksi saanut päivityksen. Jos anturin päivityslippu on ylhäällä, käsketään kyseisen anturin säiettä lähettää päivitykset asiakkaalle. Virhetilalippu nostetaan ylös, jos anturi ei ole saanut lämpötilapäivitystä 30 minuuttiin. Tästä lähetetään myös sähköposti eteenpäin.

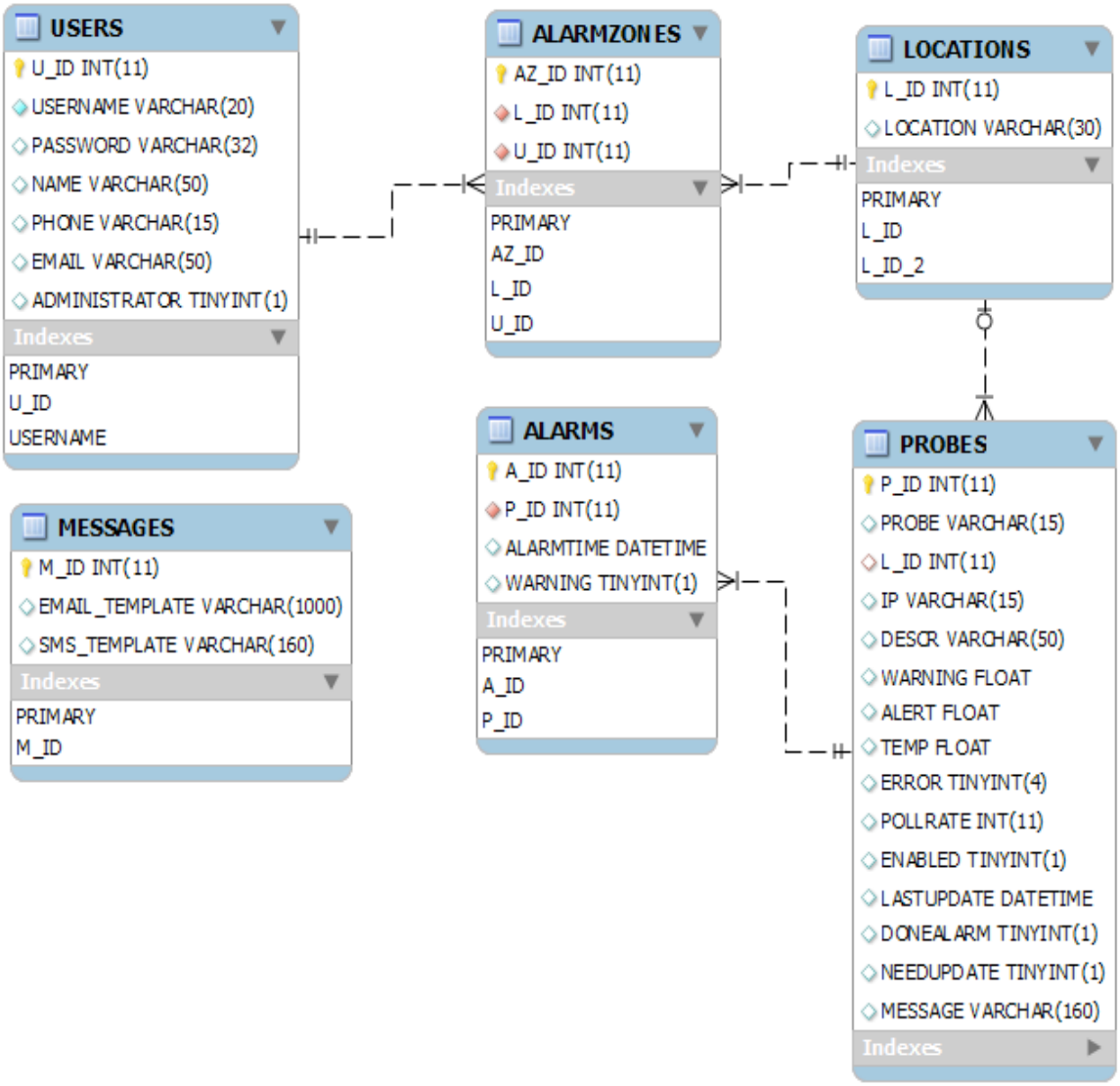
```
class DatabaseLoop(threading.Thread):
 def __init__(self, parent):
  self.parent = parent
  threading.Thread.__init__(self)
  self.logger = Logger()
  self.createDatabaseConnection();
 def run(self):
  while 1:
   if self.mysql.connection:
    self.mysqlConnection = True
    sql = "SELECT * FROM PROBES;"
    self.mysql.select(sql)
    while 1:
     row = self.mysql.next(how=1)
     if row == () or not self.mysql.connection:
      break
     self.parent.probes[row[0]['PROBE']] = row[0]
     if not int(row[0]['ERROR']):
      self.checkForError(row[0]['PROBE'], row[0]['LASTUPDATE'])
     if int(row[0]['NEEDUPDATE']):
      probe = row[0]['PROBE']
      self.logger.write("Databaseloop: Probe '%(probe)s' needs "
            +"to send updates" % {'probe': probe}, 2)
      for conn in self.parent.connections:
       c = self.parent.connections[conn]
       if c['probes'].count(probe):
```

```
        c['sc'].sendUpdates(row[0]['PROBE'])
   else:
    if self.mysqlConnection:
     self.mysqlConnection = False
     self.logger.write("Databaseloop: Connection to database has
failed", 1)
   time.sleep(5)
```

5.1 MySQL

Tietokantasovelluksena oli sekä asiakkaan pyynnöstä että omasta valinnasta MySQL. MySQL:ään on helppo yhdistää ja tehdä toteutus lähes kaikilla ohjelmointikielillä. Python ei ole tästä poikkeus ja sillä on erittäin helppoa tehdä MySQL-kyselyitä sekä -hakuja. Tietokannan suunnittelussa tärkeintä oli eheys ja selvyys. Kun tietokannan taulut on suunniteltu, pitää sovelluksen kehittäjän osata tästä taulusta löytää kaikki tarvitsemansa, tämän takia taulujen sekä solujen nimien pitää kuvata sisältävää tietoa hyvin. Tietokannassa käytettiin oletustietokantamoottorin, MyISAM:in, tilalla InnoDB-moottoria, joka mahdollistaa mm. rajoitteiden, constraints, käytön. Nämä rajoitteet ovat erittäin hyvä tapa pakottaa tietokanta pitämään itsensä eheänä. Jos taululle on asetettu rajoitteita, hylkää se kaikki muutokset, jotka rikkovat näitä sääntöjä. Näiden avulla onnistuu myös eheyden säilytys tiedon poistamistilanteessa. Jos riville on määritetty cascade ominaisuuksia ja sitä muutetaan tai se poistetaan, päivittyvät tiedot kaikissa tauluissa automaattisesti rajoitteiden mukaisesti /7/.

```
CREATE TABLE TEMPCONTROL.ALARMZONES (
      AZ_ID INT NOT NULL UNIQUE AUTO_INCREMENT,
      L_ID INT NOT NULL,
      U_ID INT NOT NULL,
      PRIMARY KEY(AZ_ID),
      CONSTRAINT FOREIGN KEY (L_ID)
                  REFERENCES TEMPCONTROL.LOCATIONS(L_ID),
      CONSTRAINT FOREIGN KEY (U_ID)
                  REFERENCES TEMPCONTROL.USERS(U_ID)
) ENGINE=InnoDB;
```



Kuva 5. Tietokantakaavio

Sovelluksessa tehtiin python –sovellukseen MySQL avustaja-luokka, jonka kautta tietokannan käskytys näyttää yksinkertaisemmalta ja on mielekkäämpi toteuttaa. Luokkaan toteutettiin neljä metodia. Init eli konstruktori saa parametreinaan tietokannan IP osoitteen, portin, käyttäjän, salasanan ja käytettävän tietokannan, joita käyttäen se luo yhteyden tietokantaan. Query suorittaa update- tai insert-lauseen annetulla parametrilla. Select suorittaa select-lauseen ja pistää saadut rivit talteen. Next iteroi läpi saatuja rivejä.

```
def __init__(self, _host, _port, _user, _password, _database):
def query(self, sql):
def select(self, sql):
def next(self, how=0):
```

5.2 Gammu

Sovelluksessa käytetään Gammua tekstiviestien lähettämiseen hälytystilanteissa. Sovellus käyttää hyväkseen SAMK:n järjestelmissä olevaa valmista Gammu-järjestelmää. Gammu:n käyttö toimii ulkopuolisen tietokannan kautta, Gammun tietokantayhteyden konfigurointi tehdään sovelluksen asetuksiin, joista sovellus ne lukee ja hälytyksen tullessa muodostaa tarpeelliset kentät sekä lähettää viestin eteenpäin Gammun käyttämään tietokantaan.

```
    def sendSms(self, address, msg):
        ghost = self.parent.confs['Gammu']['host'];
        gport = self.parent.confs['Gammu']['port'];
        guser = self.parent.confs['Gammu']['username'];
        gpw = self.parent.confs['Gammu']['password'];
        gdb = self.parent.confs['Gammu']['database'];
        try:
            gammu = MySQL(ghost, int(gport), guser, gpw, gdb);
            sql = "INSERT INTO outbox (SendingDateTime, Text, Text-
Decoded, DestinationNumber, Coding, CreatorID) \
            values (NOW(),
NULL,'"+msg+"','"+address+"','Default_No_Compression','Tempcontrol')
"
            gammu.query(sql)
        except:
            self.logger.write("Serverconnection: Error while writing
to gammu database", 1)
        return
```

5.3 Sendmail

Sähköpostin lähetys hälytystilanteissa on toteutettu linuxin avoimen lähdekoodin Sendmail-ohjelmalla. Sendmail on Internetin yleisin sähköpostin välitysohjelmisto. Sen yleisyys johtuu suureksi osaksi siitä, että se on standardi sähköpostin välitysohjelma monissa unix- ja unixin tyylisissä käyttöjärjestelmissä. Sendmailin lasketaan lähettävän 29.4% maailman sähköposteista /8/. Sovellus käyttää Sendmailia ajamalla sitä suoraan Pythonin os kirjaston kautta.

```
    def sendEmail(self, address, msg):
        SENDMAIL = "/usr/sbin/sendmail"
        p = os.popen("%s -t" % SENDMAIL, "w")
        p.write("To: "+ address + "\n")
        p.write("Subject: Tempcontrol: Alert\n")
        p.write("\n") # blank line separating headers from body
        p.write(msg + "\n")
        sts = p.close()
```

6 PROTOKOLLA

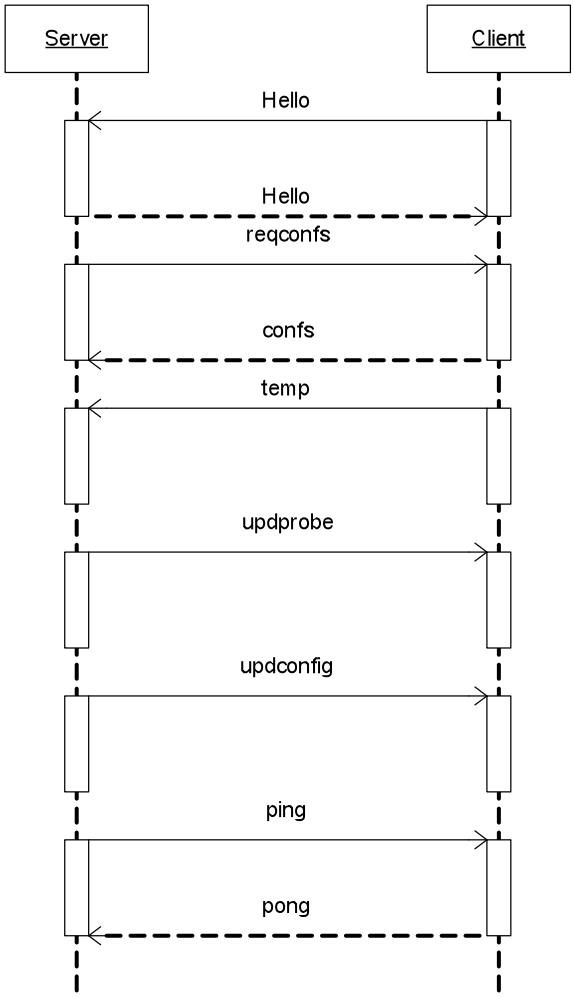
Protokolla on sarja sääntöjä, joita verkossa toimivat koneet käyttävät kommunikoimiseen. Sovellukset tarvitsevat protokollia, jotta ne osaavat toimia keskenään oikein /9/. Protokollan suunnittelussa on otettu huomioon käytön yksinkertaisuus, mutta kuitenkin mahdollisten toimintojen lisäämisen helppous. Ohjelman toteutusta suunniteltaessa todettiin, että ohjelman tarvitsee siirtää verkon yli lämpötiloja, asetuksia sekä näitä pitää voida komentaa web-sovelluksesta. Protokolla on vaatimukset huomioon ottaen toteutettu seuraavalla tavalla. Viestit sovellusten välillä liikkuvat 52 tavun jonoissa. Ensimmäiset 15 tavua ovat komento, seuraavat 15 merkkiä anturi, seuraavat 15 merkkiä muuttuja ja viimeiset tavut kuuluvat float muuttujaan. Tätä protokollaa käyttämällä on toteutettu kaikki toiminta.

komento	anturi	muuttuja	arvo
hello			
temp	probe1		23,5
updprobe	probe2	pollrate	30
reqconfs	probe2	pollrate	

Esimerkkiviestejä

Verkkoyhteys alkaa, kun asiakassovellus lähettää palvelimelle viestin Hello. Palvelimen pitää tähän viestiin vastata Hello, jonka jälkeen merkitään kättely suoritetuksi. Yhteys pidetään auki, kunnes se katkeaa, eli sitä ei katkaista itse ollenkaan. Tämä mahdollistaa käskytyksen palvelimelta asiakkaalle, vaikka asiakaskone olisi palomuurin tai NAT:n takana. Kun yhteys on päällä, alkaa asiakas lähettää lämpötila-arvoja palvelimelle. Ensimmäisellä lämpötila-arvon lähetyskerralla vastaa palvelin viestiin ’reqconfs’ viestillä. Tällä palvelin tarkistaa. onko anturin asetuksia muutettu ja jos on, niin tallentaa ne tietokantaan. Yhteyden tarkistusta pidetään yllä ’ping’ viesteillä.

```
    def send(self, command, probe, variable, value):
        try:
            self.logger.write("Serverconnection: Sending... ", 3)
            self.conn.send(tcPack(command, probe, variable, value))
            self.logger.write("Serverconnection: Sent!", 3)
        except socket.error:
            self.logger.write("Serverconnection: Connection error",
1)
            self.connection = False
```



Kuva 6. Protokolla

7 WEBSOVELLUS

Web-sovelluksen toteutusalustaksi valittiin asiakkaan pyynnöstä Java EE. Sovelluksen suunnittelussa tärkeimmät huomioonotettavat asiat olivat sovelluksen selkeys, toimivuus ja koodin muokattavuus. Sovelluksen käyttö vaatii kirjautumisen, joka toteutettiin yksinkertaisella lomakkeella. Lomakkeen tiedot tarkastetaan tietokantaa vastaan, jonka jälkeen päätetään JSF:lle tyypillisellä tavalla, mille sivulle seuraavaksi päädytään. Kirjautumisen yhteydessä myös katsotaan käyttäjän oikeudet. JSF:n tapana toteuttaa sivuilla liikkuminen on komentolinkeillä ja ohjaimen metodien palautusarvoilla.

```
<?xml version='1.0' encoding='UTF-8' ?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
 <h:head>
  <title>Tempcontrol</title>
 </h:head>
 <h:body>
  <h:form>
   <h:panelGrid columns="2" >
    <h:outputText value="Käyttäjätunnus"/>
    <h:inputText value="#{UsersController.user.username}" ti-
tle="Login" id="username" />
    <h:outputText value="Salasana"/>
    <h:inputSecret value="#{UsersController.user.password}" ti-
tle="Password" id="password" />
    <h:commandButton action="#{UsersController.login}" val-
ue="Kirjaudu" />
   </h:panelGrid>
  </h:form>
 </h:body>
</html>
```

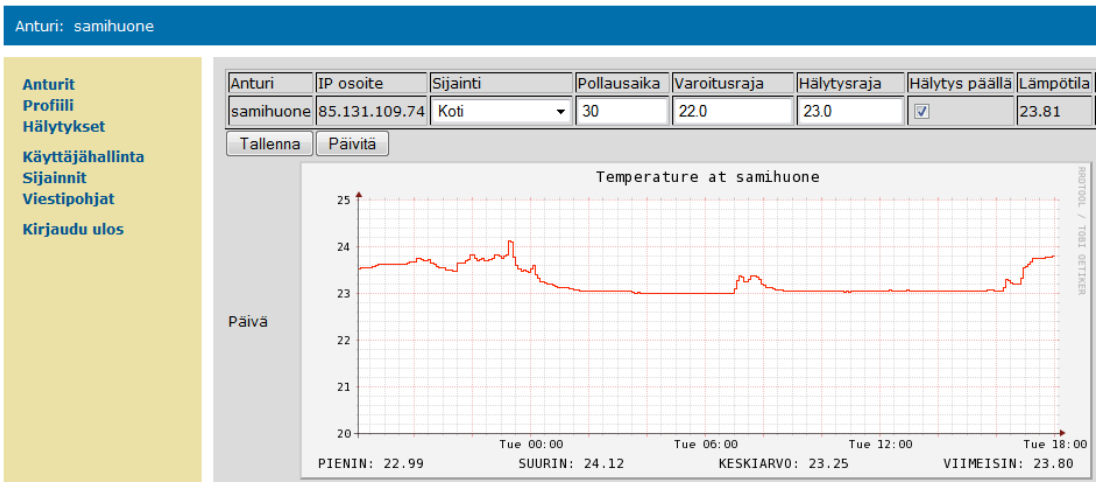
Kirjautumispainikkeen action-parametriksi on annettu expression language -kielellä (EL) ohjaimen metodi. Tämä metodi tarkastaa lomakkeen tiedot tietokantaa vastaan, ja palauttaa arvon ”probes” tai arvon ”index”. Jos sivujen välillä liikkuminen olisi suunniteltu JSF:n faces-config.xml -tiedostoon sijoitetuilla kuvauksilla, olisi näille voinut antaa mitkä tahansa arvot, kuten ”success” tai ”failure”, mutta koska sovellus on kohtuullisen pieni, ei tätä toimintaa nähty tarpeelliseksi. Ilman näitä määrittelyitä JSF käyttää komentolinkin action parametria, eli ohjelman tapauksessa kontrollerin metodien palautusarvoa oletusosoitteena, ns. index.jsf tai probes.jsf.

```
public String login() {
  Users tempUser = users.findByUsername(currentUser.getUsername());
  if (tempUser.getUsername() != null
    && !tempUser.getUsername().isEmpty()
    && tempU-
ser.getPassword().equals(encode(currentUser.getPassword()))) {
      currentUser = tempUser;
      admin = currentUser.getAdministrator();
      authed = true;
      return "probes";
  }
  else {
    return "index";
  }
}
```

Web-sovelluksen käyttöliittymä on muodostettu JSF:n composite templates mallio-minaisuudella. Sivuista on luotu yksi mallipohja, jota kaikki muut sivut käyttävät runkonaan. Mallipohjassa on määritelty sivuston tyyli sekä sijainnit sisällölle ja linkeille. Sivuston kirjautumisen tarkistus on tehty käyttäen tätä mallipohjaa, jos sivun käyttäjä ei ole kirjautunut, ohjataan hänet kirjautumissivulle, muuten näytetään oike-

at sisällöt. Sivustolla liikkuminen on toteutettu käyttäen yksinkertaista linkki-listaa sivun vasemmassa reunassa.

Kuva 7. Sivuston malli, antureiden listaus.

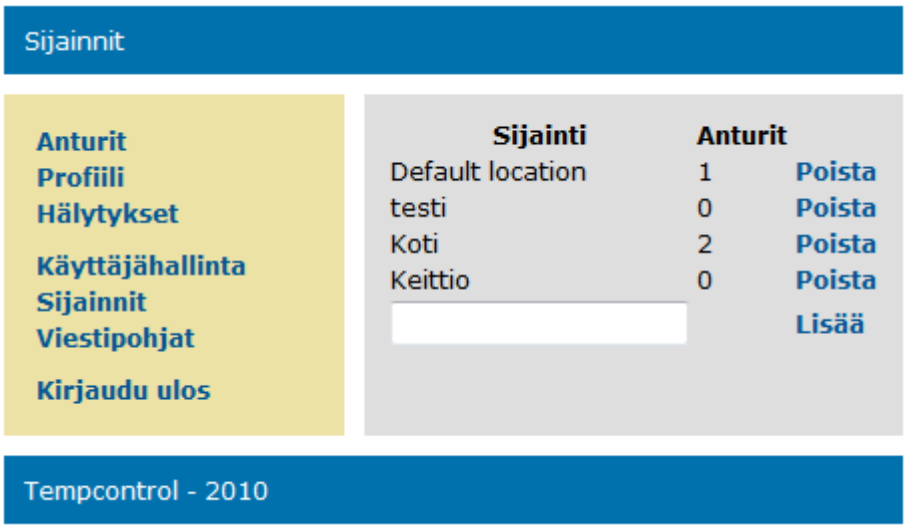


Kuva 8. Esimerkki anturin tiedoista

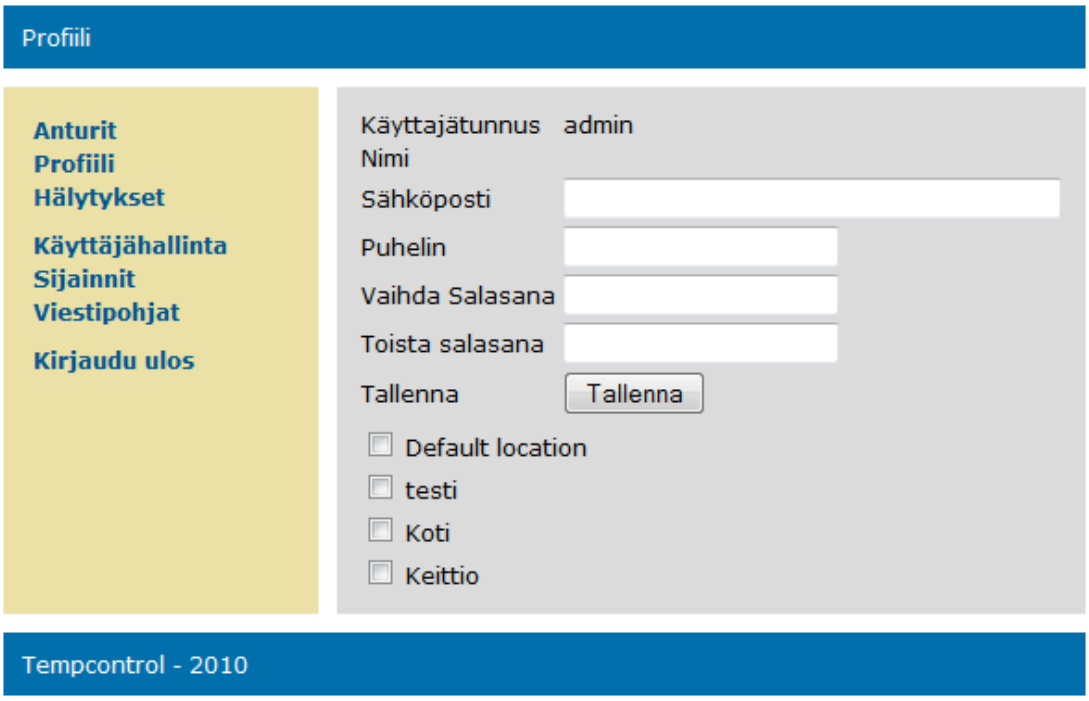
Sivustolla on mahdollista muokata tiettyjä anturin ominaisuuksia ja käyttäjä-listausta, katsoa tilastoja hälytyksistä sekä muodostaa hälytysviestien pohjat. Viestipohjiin on mahdollista liittää linkkejä anturin ominaisuuksiin, jotka korvataan oikeilla tiedoilla python-osassa viestiä lähetettäessä.

Kuva 9. Esimerkki viestimallien luomisesta.

Sivuilla pystyy luomaan ja poistaa hälytysalueita, joihin käyttäjät pystyvät liittymään. Jokainen anturi kuuluu vain yhteen hälytysalueeseen. Tätä käyttämällä on saman alueen hälytysten valvonta helppo sijoittaa tietyille henkilöille. Tämän perusteella Python-sovellus osaa valita kenelle anturien hälytykset lähtevät. Pääkäyttäjäoikeudet omaava käyttäjä voi myös liittää muihin käyttäjiin tiettyjä hälytysalueita, näin jokaisen henkilön ei tarvitse käyttää ohjelmaa, vaan he voivat vain saada hälytykset. Tavalliset käyttäjät eivät pysty tekemään muuta kuin tarkastellaantureiden ja hälytysten statistiikkaa sekä muuttaa omia tietojaan.



Kuva 10. Sijaintien luominen ja poisto



Kuva 11. Profiilissa valitaan sijainnit.

8 TIETOTURVA

Sovelluksen tietoturva suunniteltiin sovelluksen käyttöympäristö huomioon ottaen. Ainoastaan web-sovellukseen on mahdollista päästä käsiksi palomuurin ulkopuolelta. Tämän vuoksi asiakkaan ja palvelimen väliseen yhteyteen ei ole toteutettu monipuolista tietoturvaa. Tietoturva koostuu HELLO-viestin mukana lähetettävästä ns. salasanasta. Jos tämä on puuttellinen, hylkää palvelin yhteyspyynnön. Tämän vuoksi

ohjelma on haavoittuvainen mies välissä –hyökkäykselle, jossa hyökkääjä asettuu palvelimen sekä asiakkaan väliin, jossa tiedot näkyvät salaamattomana. Tietoturvaa olisi voinut parantaa lähettämällä tiedot salattuina, mutta tähän ei sovelluksen käyttöympäristö sekä sovelluksen ei-kriittinen toiminta huomioon ottaen ryhdytty.

Web sovelluksen tietoturva koostuu kirjautumisesta, jossa salasana tiivistetään SHA-1 tiivistealgoritmilla. SHA-1:ä ei nykyään pidetä niin luotettavana, että sitä voisi käyttää kriittisissä sovelluksissa, mutta kyseinen pakkaus todettiin tarpeeksi luotettavaksi sovelluksen salasanojen tallennukseen. SHA-1 muodostaa tietynmittaisen merkkijonon annetuista parametreista. Jokaisen merkin muutoksesta pitää tiivisteen muuttua monikertaisesti. /10/

```
Hei maailma:
WANY0wyD3YWrHSSL4rli5qt540o=

Hei maailmb:
s0DKY9mjow+skByI854Yc3QrV/Y=
```

Salasanojen tiivisteiden muodostamiseen on kirjoitettu funktio, joka muuttaa annetun merkkijonon SHA-1 algoritmin lävitse tiivistemerkkijonoksi.

```
public String encode(String raw) {
  try {
    MessageDigest digest = MessageDigest.getInstance("SHA1");
    digest.update(raw.getBytes("UTF-8"));
    byte[] hash = digest.digest();
    String pw = (new BASE64Encoder()).encode(hash);
    return pw;
  } catch (Exception e) {
    return "";
  }
}
```

9 TESTAUS

Testauksessa oli tarkoitus suunnitella ja toteuttaa yleisesti kattava ohjelmistotestaus. Testaussuunnitelmaan tulee määritellä mahdollisia testaustapahtumia, niiden halutut tulokset sekä testin tulokset. Sovelluksen testauksessa keskityttiin yleisesti kattamaan ongelmatilanteet, joissa ohjelmisto saattaa toimia väärin tai kaatua. Kuten palvelinsovellusten yleensä, on tämänkin sovelluksen tärkein ominaisuus luotettavuus, ohjelma ei saa kaatua käytössä. Nämä asiat huomioon ottaen on valmisteltu sekä toteutettu seuraava testaussuunitelma. Toimintaa myös testattiin kahden viikon ajan aset-

tamalla rajoiksi sellaiset arvot, jotka ylittyivät silloin tällöin. Ohjelma toimi toivotulla tavalla koko testin ajan.

#	Tapahtuma	Toivottu tulos	Saatu tulos
1	Kirjautuminen väärällä salasanalla	Paluu kirjautumissivulle	Toivottu
2	Kirjautuminen oikealla salasanalla	Siirtyminen anturilistaukseen	Toivottu
3	Antureiden asetusten muokkaus	Asetusten tallentuminen tietokantaan sekä lähetys asiakkaalle	Toivottu
4	Profiilin muokkaus	Muokkaus onnistuu	Toivottu
5	Profiilin muokkaus jos salasanat eivät täsmää	Muokkaus ei onnistu	Toivottu
6	Käyttäjän admin poisto	Poisto ei onnistu	Toivottu
7	Käyttäjän lisääminen	Tiedot tallentuvat tietokantaan	Toivottu
8	Käyttäjän lisääminen tyhjillä kentillä	Tallennus ei onnistu	Toivottu
9	Viestimallien muokkaus	Tiedot tallentuvat tietokantaan	Toivottu
10	Yritys avata sivu suoraan kirjautumatta	Ohjautuu takaisin etusivulle	Toivottu
11	Asiakassovelluksen käynnistys väärillä asetuksilla	Sovellus ei käynnisty	Toivottu
12	Asiakassovelluksen toiminta jos verkkoyhteys menetetään	Asiakassovellus yrittää yhdistää uudestaan	Toivottu
13	Asiakassovelluksen toiminta jos digitemp antaa epämääräisen arvon	Asiakassovellus hylkää arvon	Toivottu
15	Palvelinsovelluksen käynnistys puuttellisilla asetuksilla	Sovellus ei käynnisty	Toivottu
16	Palvelinsovelluksen toiminta jos tietokanta-yhteys katkeaa	Sovellus katkaisee socket-yhteydet ja palaa yrittämään tietokantayhteyttä	Toivottu
17	Palvelinsovelluksen toiminta jos socket vastaanottaa väärän tervehdyksen tai komennon	Sovellus katkaisee yhteyden	Toivottu

LÄHTEET

1. Distributed Multitiered Applications – The Java EE 6 Tutorial, Volume 1. [verkkodokumentti]. Saatavissa: http://java.sun.com/javaee/6/docs/tutorial/doc/bnaay.html

2. Goncalves, I. Beginning Java trade EE 6 Platform with GlassFish trade 3 From Novice to Professional. 1.painos. Berkeley, CA, Apress, 2005. 478 s.

3. Wikipedia. MySQL [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/MySQL

4. Wikipedia. Python [verkkodokumentti]. Saatavissa: http://fi.wikipedia.org/wiki/Python

5. Wikipedia. Gammu [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/Gammu_(software)

6. Wikipedia. GlassFish [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/GlassFish

7. Enforcing Foreign Keys Programmatically in MySQL [verkkodokumentti]. Saatavissa: http://dev.mysql.com/tech-resources/articles/mysql-enforcing-foreign-keys.html

8. Wikipedia. Sendmail [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/Sendmail

9. Wikipedia. Protocol [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/Protocol_%28computing%29

10. Wikipedia. SHA-1 [verkkodokumentti]. Saatavissa: http://en.wikipedia.org/wiki/SHA-1