LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Tietotekniikan osasto

Diplomityö

Sovellus meteorologisen tiedon tuotantojärjestelmän seurantaan

Diplomityön aihe on hyväksytty Tietotekniikan osaston osastoneuvostossa 12.11.2003.
Tarkastajat: Prof. Heikki Kälviäinen ja FM Anu Lahdensuu
Ohjaaja: Pirkko Kilpeläinen

Vantaalla 20. tammikuuta 2004

Pekka Keränen
Väritehtaankatu 2 C 65
01300 Vantaa
041-5443495
pekka.keranen@lut.fi

Tiivistelmä

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto
Pekka Keränen

Sovellus meteorologisen tiedon tuotantojärjestelmän seurantaan

Diplomityö
2004

76 sivua, 18 kuvaa, 2 taulukkoa ja 3 liitettä
Tarkastajat: Professori Heikki Kälviäinen ja FM Anu Lahdensuu

Hakusanat: Tuotantojärjestelmä, Java, servlet, tietokanta, hajautettu sovellus
Keywords: Production system, Java, servlet, database, distributed application

Työssä käsitellään selainkäyttöliittymää käyttävien oliopohjaisten tietokantasovellusten toteuttamista. Erityisesti keskitytään olio- ja relaatiomallien yhteensovittamiseen ja oliopohjaisten selainkäyttöliittymien toteutukseen Java-servlettien ja JSP-sivujen avulla. Myös hajautetut sovellusarkkitehtuurit käydään läpi ja niiden toteuttamista arvioidaan servlet-sovellusten näkökulmasta.

Työssä on toteutettu selainkäyttöliittymän avulla hallittava kaksitasoarkkitehtuuria käyttävä oliopohjainen sovellus Ilmatieteen laitoksen tuotantojärjestelmän seurantaan. Sovellus mahdollistaa mm. tuotantoajojen ja laajempien tuotantoketjujen suoritusaikojen tilastollisen seurannan.

Työn tuloksena todettiin Java-servlettien ja JSP-sivujen olevan suorituskykyinen ja monipuolinen ratkaisu selainkäyttöliittymien toteuttamiseen. Olio- ja relaatiomallien väliset erot sekä käyttöliittymän eriyttäminen toimintalogiikasta osoittautuivat ongelmakohdiksi.

Abstract

Lappeenranta University of Technology
Department of Information Technology
Pekka Keränen

An Application for the Monitoring of Meteorological Production System

Master's thesis
2004

76 pages, 18 figures, 2 tables and 3 appendices
Supervisors: Professor Heikki Kälviäinen and FM Anu Lahdensuu

Keywords: Production system, Java, servlet, database, distributed application

This Master's thesis describes the development of web-based object oriented database applications. Special attention is paid to the relationship between relational and object-oriented models and to the development of object oriented web-based applications using Java Servlet- and JSP-technologies. Also distributed applications have been discussed in the perspective of servlet-applications.

In the practical part of this work a web-based object-oriented application, that uses 2-tier architecture has been developed to monitor the production system of Finnish Meteorological Institute. The application makes possible to statistically monitor production processes.

The study concludes that the performance and programming interface of Java Servlet- and JSP-technologies is excellent. Differences between object-oriented and relational models and the distribution between presentation and operational layers were problematic issues.

Alkusanat

Tämä diplomityö on tehty Ilmatieteen laitoksen tietohallinnolle kesäkuun 2003 ja tammikuun 2004 välisenä aikana. Haluan kiittää työantajaani mahdollisuudesta diplomityön tekemiseen ja miellyttävästä työilmapiiristä. Erityisesti haluan kiittää työn ohjaamiseen osallistuneita Pirkko Kilpeläistä, Anu Kalliota, Timo Kuoremäkeä ja Jari Winbergiä. Kiitokset myös Veikko Nyforsille työn alkuperäisestä ideasta. Professori Heikki Kälviäistä haluan kiittää sujuvasta yhteistyöstä diplomityön ohjauksessa.

Opiskeluaika on ollut monella tavalla antoisaa Skinnarilan vapaavaltion teekkariyhteisössä ja siten myös opiskelukaverit ansaitsevat tulla muistetuksi. Kiitokset myös vanhemmilleni ja muille elämäni tärkeille ihmisille. Erään punk-levyn kansitekstiä lainatakseni ”kiitämme kaikkia”.

Vantaalla, 20. tammikuuta 2004

Pekka Keränen

Sisällysluettelo

1 Johdanto....5
2 Relaatio- ja oliomallit....7
2.1 Relaatiomalli ja relaatiotietokannat....7
2.1.1 Relaatiomalli....7
2.1.2 Relaatioiden luominen ja käsittely....9
2.1.3 SQL-kyselykieli....10
2.1.4 Relaatiotietokantojen tiedonhallintajärjestelmät....11
2.2 Olioparadigma....12
2.2.1 Oliot ja luokat....12
2.2.2 Olioparadigman edut....14
2.2.3 Oliotietokannat....15
2.3 Olioiden kartoittaminen relaatiotietokantaan....16
2.3.1 Ristiriidat mallien välillä....16
2.3.2 Luokkien kartoittaminen erillisiin tauluihin....17
2.3.3 Luokkien kartoitus yhteen tauluun....18
2.3.4 Aliluokkien jako erillisiin tauluihin....19
2.3.5 Muita menetelmiä olio- ja relaatiomallien yhteensovittamiseen....21
3 Hajautetut sovellukset....23
3.1 Tarve sovellusten hajauttamiselle....23
3.2 Sovelluksen kerrokset....23
3.3 Sovellustasot....24
3.3.1 Yksitasoiset sovellukset....24
3.3.2 Kaksitasoiset sovellukset....25
3.3.3 Kolmitasoiset sovellukset....25
3.3.4 N-tasoiset sovellukset ja hybridiarkkitehtuurit....27
3.4 Sovellusarkkitehtuurin valinta....27
3.5 MVC-paradigma....28
4 Selainpohjaisen sovelluksen toteuttaminen palvelimella....30
4.1 HTTP-protokolla ja selainpohjaiset sovellukset....30
4.2 CGI-ohjelmat....32
4.3 Server Side Includes....33

4.4 PHP ja ASP....33
4.5 Java-servletit....34
4.5.1 Java ohjelmointikielenä....34
4.5.2 Java-servletit....36
4.5.3 Ohjelmointirajapinta....38
4.5.4 Java Server Pages....39
4.5.5 Sessionhallinta....40
4.5.6 Tietokantayhteydet....40
4.5.7 Servlettien paketointi sovelluksiksi....42
4.5.8 Servlet-teknologian edut ja heikkoudet....44
5 Ilmatieteen laitoksen tuotantojärjestelmä....47
5.1 Ilmatieteen laitos....47
5.2 Ilmatieteen laitoksen tuotantojärjestelmä....48
5.2.1 Yleistä tuotantojärjestelmästä....48
5.2.2 Palvelinympäristö....50
5.2.3 SMS-valvontaohjelma....51
5.2.4 Oliohakurajapinta....53
6 Sovellus tuotantojärjestelmän seurantaan....55
6.1 Tarve tuotantojärjestelmän seurantaan....55
6.2 Sovellukselle asetetut vaatimukset....56
6.3 Sovelluksen kuvaus ja tekniset ratkaisut....56
6.3.1 Sovelluksen osat....56
6.3.2 Tekniset ratkaisut....58
6.3.3 Tietokannan kuvaus....60
6.3.4 Oliomalli....61
6.3.5 Käyttöliittymä....64
6.3.6 Käyttäjäroolit....66
6.3.7 Raportit....66
6.4 Toteutus ja käyttöönotto....68
6.5 Jatkokehitys....69
7 Yhteenveto....71
Lähteet....73
LIITTEET

Lyhenneluettelo

API	Application Programming Interface
ASP	Active Server Page
CDP	Command Display Program
CGI	Common Gateway Interface
CORBA	Common Object Request Broker
CSS	Cascading Style Sheets
DBMS	Database Management System
DDL	Data Definition Language
DHTML	Dynamic Hypertext Markup Language
DML	Data Manipulation Language
FTP	File Transfer Protocol
GTS	Global Telecommunications System
HIRLAM	High Resolution Limited Area Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
J2ME	Java 2 Micro Edition
JAR	Java Archive
JIT	Just-In-Time
JDBC	Java Database Connectivity
JSP	Java Server Pages
MVC	Model-View-Controller
NTP	Network Time Protocol
ORB	Object Request Broker
PNG	Portable Network Graphics
SCP	Secure Copy Protocol
SMS	Supervisor Monitor Scheduler
SNTP	Simple Network Time Protocol
SSI	Server Side Includes
SQL	Structured Query Language

STDIN	Standard Input
STDOUT	Standard Output
TCP	Transmission Control Protocol
UML	Unified Modeling Language
UDP	User Datagram Protocol
URL	Universal Resourse Locator
UTC	Universal Time Coordinated
WAR	Web Archive
WWW	World Wide Web
XCDP	X Command Display Program
XML	Extensible Markup Language

1 Johdanto

Säällä on suuri merkitys ihmisten jokapäiväisessä elämässä niin työssä kuin harrastuksissa. Äärimmäiset sääilmiöt voivat vaikuttaa jopa ihmisten turvallisuuteen. Tämän takia luotettavien sääennusteiden merkitys on kiistaton. Säähavaintojen jalostaminen lopputuotteeksi, kuten television uutislähetyksen sääkartoiksi, on pitkä ja monimutkainen prosessi.

Meteorologiassa on aina hyödynnetty tietotekniikan mahdollisuuksia ensimmäisten joukossa. Nykyaikana ympäri maapalloa kerättyjen säähavaintojen käsittely, ennustemallien laskeminen ja säätuotteiden tuotanto ei ole mahdollista ilman tietotekniikkaa. Säätuotteiden tuotanto on pitkälti automatisoitu, vaikka myös meteorologeilla on inhimillinen roolinsa supertietokoneiden laskemien ennustemallien tulkinnassa. Tuhansien erilaisten säätuotteiden tuottaminen ei ole mahdollista ilman nykyaikaista tietotekniikkaa. Lisäksi tuotantojärjestelmien on toimittava luotettavasti 24 tuntia vuorokaudessa ympäri vuoden.

Tässä diplomityössä on toteutettu sovellus Ilmatieteen laitoksen tuotantojärjestelmän seurantaan. Sovellus mahdollistaa yksittäisten tuotantoajojen ja laajempien tuotantoketjujen läpimenoaikojen kirjaamisen relaatiotietokantaan. Tietokannan perusteella on mahdollista luoda erilaisia raportteja ja löytää tuotantojärjestelmän ongelmakohdat. Sovellukseen on liitetty myös muita ominaisuuksia, kuten tuotantoajojen vastuuhenkilöiden ja asiakkaiden hallinta. Toteutettu sovellus on oliopohjainen ja alustariippumaton kaksitasoarkkitehtuuria käyttävä sovellus, jota loppukäyttäjä hallitsee selainpohjaisen käyttöliittymän avulla. Sovelluksen toteuttamisessa on otettu huomioon meteorologisen tuotantojärjestelmän erityispiirteet, mutta sovellusta voidaan käyttää myös muihin tarkoituksiin.

Diplomityön toteutuksessa on hyödynnetty useita ohjelmistokehityksen menetelmiä, jotka käydään läpi työn teoriaosuudessa. Teoriaosuudessa käydään läpi olio- ja relaatiomallit, hajautetut sovellusarkkitehtuurit ja selainpohjaisten sovellusten toteuttaminen. Erityisesti keskitytään olio- ja relaatiomallien yhteensovittamiseen sekä Java-servlettien toimintaan. Diplomityön soveltavassa osuudessa käydään läpi Ilmatieteen laitoksen tuotantojärjestelmän toiminta sekä toteutettu sovellus tuotantojärjestelmän seurantaan.

2 Relaatio- ja oliomallit

2.1 Relaatiomalli ja relaatiotietokannat

2.1.1 Relaatiomalli

E.F Codd esitteli relaatiomallin tiedon hallintaan vuonna 1970 julkaisussa ”A Relational Model of Data for Large Shared Data Banks”. [1] Codd pyrki luomaan mallin tietojen hallintaan, joka olisi riippumaton tiedon sisäisestä esitysmuodosta tiedonhallintajärjestelmissä ja säilyttäisi tiedon eheyden. Relaatiomallin taustalla on vankka matematiikan joukko-oppiin pohjautuva teoria, mutta mallin ymmärtäminen ei vaadi matemaattista taustaa. Relaatiomalli on nykyisin yleisin tietokannoissa käytetty tietomalli.

Relaatiomallissa tietokanta esitetään joukkona relaatioita. Käytännössä relaatio vastaa kaksiulotteista taulua, johon data jäsennetään. Relaatiomallin yhteydessä käytettävässä terminologiassa taulun rivejä kutsutaan monikoiksi (tupla), sarakkeita attribuuteiksi ja taulua relaatioksi. [2] Selkeyden vuoksi jatkossa käytetään nimityksiä sarake, rivi ja taulu.

Relaatiomallissa taulun sarakkeet vastaavat yksittäisiä datakappaleita kuten esimerkiksi numeroita, merkkijonoja ja päivämääriä. Taulun rivit edustavat yhtä dataentiteettiä. Entiteetillä tarkoitetaan relaatiomallin yhteydessä tietokantaan tallennettua asiaa. Esimerkiksi tilaus tai henkilö voi olla entiteetti. Jokaisessa taulussa on yksi tai useampi sarake, jotka yksilöivät yksiselitteisesti taulun rivit. Näitä sarakkeita kutsutaan perusavaimiksi. Taulun eheys vaatii, että kahdella tai useammalla rivillä ei ole samaa perusavainta ja että perusavaimen arvo ei ole tyhjä (null). [3] [4]

Asioilla on erilaisia yhteyksiä toisiinsa. Esimerkiksi jokaisella opiskelijalla on vain yksi biologinen äiti, opiskelija voi käydä useammalla kurssilla ja kurssilla voi olla useita opiskelijoita. Relaatiomallissa tällaisia riippuvuuksia ja yhteyksiä kuvataan taulujen välisillä yhteyksillä, jotka muodostetaan viiteavainten avulla. Viiteavainyhteydessä

taulun sarake viittaa toisen taulun perusavaimeen. Taulujen väliset yhteydet voidaan jakaa yhden suhde yhteen (1:1), yhden suhde moneen (1:n) ja monen suhde moneen (n:n) yhteyksiin riippuen, kuinka moneen toisen taulun riviin taulun rivit viittaavat. 1:n yhteys voidaan muuttaa 1:1 yhteydeksi laittamalla viiteavainsarakkeeseen yksilöivä indeksi. Monen suhde moneen yhteys ei ole suoraan mahdollinen relaatiomallin eheyden säilyttämiseksi, joten n:n yhteydet puretaan luomalla linkkitaulu kahden tai useamman taulun välille. Relaatiomalli mahdollistaa viiteavaineheyden määrittelyn taulujen välille. Viiteavaineheys määritellään siten, että taulun A rivin, joka viittaa taulun B riviin, täytyy viitata taulun B olemassa olevaan riviin. [3] [4]

Kuvassa 1 on esitetty yksinkertainen esimerkki relaatiotauluista ja taulujen välisistä yhteyksistä. Taulujen perusavaimia ovat opiskelijanro, kurssinro ja opettajanro. Osallistuja-taulussa taulun rivien yksilöintiin tarvitaan kaksi perusavainta, jotka ovat kurssinro ja opiskelijanro. Opiskelijoiden ja kurssien välillä on n:n yhteys. Opettajien ja kurssien välillä on 1:n yhteys. Osallistujat-taulu on linkkitaulu, jonka avulla on purettu n:n yhteys opiskelijoiden ja kurssien välillä.

Opiskelijat		
opiskelijanro	nimi	vuosikurssi
1	Matti Virtanen	2
2	Maija Kunnas	1
3	Liisa Korpi	4

Opettajat		
opettajanro	nimi	huone
1	Risto Kangas	1562
2	Auli Seppänen	1478
3	Kalle Koski	2211

Kurssit		
kurssinro	nimi	opettajanro
1	Matematiikka	2
2	Äidinkieli	3
3	Historia	3

Osallistujat	
kurssinro	opiskelijanro
1	1
1	2
2	1
2	3

Kuva 1. Yksinkertainen esimerkki relaatiotauluista ja niiden välisistä yhteyksistä.

2.1.2 Relaatioiden luominen ja käsittely

Relaatiotaulujen suunnittelu on tehtävä huolellisesti monimutkaisia tietokantoja luotaessa. Suunnitteluun kuuluu mm. tauluihin liittyvien sarakkeiden valinta, taulujen ja sarakkeiden nimeäminen, sarakkeiden tietotyyppien ja arvoalueiden määrittely sekä perus- ja viiteavainten määrittely. Relaatiomallin suunnitteluun on kehitetty useita menetelmiä. Taulut luodaan käyttäen tiedonhallintajärjestelmän tukemaa määrittelykieltä (Data Definition Language, DDL). [3]

Relaatiotauluille suoritettavat operaatiot voidaan jakaa tieto hakeviin ja tietoa muokkaaviin operaatioihin. Kolme perusoperaatiota tiedon muokkaamiseen ovat lisäys, poisto ja päivitys. Kaikki kolme operaatiota voivat rikkoa relaatiotietokannan eheyden, mikä pitää huomioida operaatioita suoritettaessa. [3]

Relaatioalgebralla tarkoitetaan operaatioiden joukkoa, jotka mahdollistavat kokonaisten relaatiotaulujen käsittelyn. Relaatioalgebraa käytetään tietojen hakemiseen ja uusien taulujen muodostamiseen olemassa olevista tauluista. Relaatioalgebran operaatiot voidaan jakaa kahteen joukkoon. Ensimmäisen joukon muodostavat puhtaasti matemaattisesti operaatiot kuten yhdiste (union), leikkaus (intersection) ja erotus (difference). Toinen joukko taas koostuu puhtaasti relaatiotietokannoille muodostetuista operaatioista, kuten valinta (select), projektio (project) ja liitos (join). [3] Taulukossa 1 on esitetty lyhyesti kuusi yleisintä relaatioalgebran operaatiota ja joukko-opin vastineet operaatioille.

Taulukko 1. Relaatioalgebran operaatioita.

Operaatio	***Tarkoitus***	***Vastine***
valinta	Valitsee osajoukon taulun riveistä, jotka sopivat valintaehtoon.	ei ole
projektio	Valitsee määritellyt sarakkeet taulusta.	ei ole
liitos	Yhdistää toisiinsa liittyvät rivit kahdesta taulusta yksittäiseksi riviksi.	ei ole
yhdiste	Muodostaa taulun, joka sisältää rivit, jotka ovat joko tauluissa R tai S tai molemmissa.	$R \cup S$
leikkaus	Muodostaa taulun, joka sisältää rivit, jotka ovat sekä taulussa R että S.	$R \cap S$
erotus	Muodostaa taulun, joka sisältää rivit, jotka ovat taulussa R, mutta eivät taulussa S.	$R - S$

2.1.3 SQL-kyselykieli

Hyvin harvat relaatiotietokantojen hallintaan suunnitellut kielet pohjautuvat suoraan relaatioalgebraan. Relaatioalgebran suurin rajoite on, että operaatiot on suoritettava tietyssä järjestyksessä. Korkeamman tason tietokantakielissä kyselyssä määritellään kyselyn haluttu lopputulos. Kyselyn lopullinen toteutus ja optimointi jää tällöin tiedonhallintajärjestelmän tehtäväksi. [3]

SQL (Structured Query Language) on yleisin relaatiotietokannoille suunnitelluista kielistä. SQL on IBM:n kehittämä kieli, joka on myöhemmin tullut standardin asemaan. SQL-kielestä on laadittu useita standardeja, joista uusin on ANSI-organisaation määrittelemä SQL-99. SQL määrittelee tietokantaoperaatiot tiedon määrittelyyn, kyselyihin ja päivittämiseen. [2]

SQL sisältää runsaasti tietotyyppejä tiedon määrittelyyn. Taulukossa 2 on listattu SQL:n yleiset tietotyypit. Tiedonhallintajärjestelmissä voi olla lisäyksiä tietotyyppeihin ja tietotyyppien toteutus vaihtelee tiedonhallintajärjestelmissä.

Taulukko 2. SQL:n yleiset tietotyypit. [4]

Tietotyypit	***Kuvaus***
CHARACTER, VARCHAR	Kiinteä- ja vaihtuvamittaiset merkkijonot
NUMERIC, DECIMAL, FLOAT	Liukuluvut
SMALLINT, INTEGER	Kokonaisluvut
DATE	Päivämäärä, joka sisältää vuoden, kuukauden ja päivämäärän
TIME	Kellonaika, joka sisältää tunnit, minuutit ja sekunnit
TIMESTAMP	Päivämäärän ja kellonajan yhdistelmä
BLOB	Binääriobjekti, joka ei sisällä merkkipohjaista tietoa (esim. kuvatiedosto)
CLOB	Merkkipohjaista tietoa sisältävä objekti

SQL-komentojen syntaksi on hyvin lähellä luonnollista kieltä. Esimerkiksi kuvan 1 tauluista voisi olla tarkoituksenmukaista hakea kaikkien opiskelijoiden nimet, joiden

vuosikurssi on 4. Tämä onnistuu SQL-lauseella ”SELECT nimi FROM Opiskelijat WHERE vuosikurssi=4”. Myös muiden SQL-komentojen syntaksi on hyvin samankaltainen. Yleisimpiä SQL-komentoja ovat taulujen luonti (CREATE TABLE), taulujen poistaminen (DROP TABLE), tiedon lisäys (INSERT INTO), tiedon päivitys (UPDATE) ja tiedon poisto (DELETE FROM). Näiden komentojen lisäksi tiedonhallintajärjestelmissä on SQL-komentoja esimerkiksi käyttäjätunnuksien ja taulujen oikeuksien hallintaan.

SQL-kielestä on annettu tässä kappaleessa vain lyhyt yleiskäsitys, mutta kyseessä on kuitenkin erittäin monipuolinen ja tehokas kieli relaatiotietokantojen hallintaan. Lisäksi tiedonhallintajärjestelmissä on valmistajakohtaisia lisäyksiä SQL-standardiin ja jopa standardien toteutus vaihtelee. Tämä hankaloittaa SQL-kieltä käyttävien tietokantasovellusten kehittämistä käytettävästä tiedonhallintajärjestelmästä riippumattomaksi.

2.1.4 Relaatiotietokantojen tiedonhallintajärjestelmät

Tiedonhallintajärjestelmä tai tietokannan hallintajärjestelmä (Database Management System, DBMS) on ohjelmisto, joka mahdollistaa käyttäjien määritellä, luoda ja ylläpitää tietokantoja ja tarjoaa kontrolloidun pääsyn tietokantoihin. [5]

DBMS tarjoaa käyttäjille tiedon määrittelykielen (Data Definition Language, DDL), jonka avulla määritellään tietokannan rakenne. Tiedon kyselyyn ja muokkaukseen DBMS tarjoaa tiedon muokkaamiseen tarkoitetun kielen (Data Manipulation Language, DML). Esimerkiksi SQL on sekä DDL- että DML-kieli. Kontrolloitu pääsy tietokantaan vaatii tiedonhallintajärjestelmältä mm. seuraavien piirteiden toteutusta:

- Käyttäjienhallinta, joka sallii vain määriteltyjen käyttäjien pääsyn tietokantaan.
- Tiedon eheyden hallinta.
- Yhtäaikaisuuden hallinta, joka sallii jaetun pääsyn tallennettuun tietoon.
- Tiedon palauttamisen mahdollisuus laitteisto- tai ohjelmistovian jälkeen.
- Käyttäjille tietokannan sisältöä kuvaavan luettelon ylläpitoa.
- Transaktioiden hallinta.

Lisäksi tiedonhallintajärjestelmät tarjoavat yleensä erilaisia näkymiä käyttäjille tietokannan dataan sekä työkaluja tietokannan ylläpitoon ja tarkkailuun. [5]

Entä milloin tiedonhallintajärjestelmä on relaatiotietokantojen tiedonhallintajärjestelmä? E.F Codd on määritellyt 12 sääntöä, jotka tiedonhallintajärjestelmän on toteutettava ollakseen relaatiotietokantojen tiedonhallintajärjestelmä. Säännöt määrittelevät mm., että kaikki tietokannan tieto on esitettävä tauluissa, tiedonhallintajärjestelmän on taattava tiedon eheys ja tarjottava käyttäjälle tietokannan hallintaan kieli, joka tarjoaa sekä DDL- että DML-ominaisuuksia. Näiden sääntöjen toteuttaminen ei ollut itsestään selvyys ensimmäisten relaatiotietokantojen tiedonhallintajärjestelmien tapauksessa. Nykyisin suurempi ongelma on rajanveto relaatiotietokantojen ja oliotietokantojen tiedonhallintajärjestelmien välillä. [5]

Nykyaikaiset tiedonhallintajärjestelmät ovat äärettömän monimutkaisia järjestelmiä, joilta vaaditaan lähes 100 %:n luotettavuutta. Suosittuja relaatiotietokantojen tiedonhallintajärjestelmiä ovat mm. Oracle, IBM:n DB2 ja Microsoftin SQL Server. Viime vuosina suosiotaan ovat myös kasvattaneet myös MySQL:n ja PostgreSql:n kaltaiset avoimeen lähdekoodiin pohjautuvat tietokantaohjelmistot.

2.2 Olioparadigma

2.2.1 Oliot ja luokat

Olioparadigma tai olioperustaisuus on yksi monista ohjelmointiparadigmoista eli tapa kuvata erilaisten järjestelmien rakennetta ja toimintaa. Perinteisesti suosituin ohjelmointiparadigma on ollut proseduraalinen ohjelmointi. Proseduraalisessa ohjelmoinnissa ohjelman toiminta jaetaan joukoksi aliohjelmia, jotka suorittavat oman pienen tehtävänsä. Proseduraalisessa ohjelmoinnissa tieto ja tiedon käsittelyyn tarvittavat aliohjelmat ovat erillään toisistaan. Sama periaate voidaan nähdä myös relaatiomalliin pohjautuvissa tietokannoissa. Taulujen sisältämä tieto ja tauluille suoritettavat SQL-lauseet ovat erillään toisistaan. Olioparadigmassa tieto ja operaatiot yhdistetään kokonaisuudeksi, jota kutsutaan olioksi.

Olio pystyy pyydettäessä suorittamaan oliolle ominaiset toiminnot, joita kutsutaan operaatioiksi. Jokaisella operaatiolla on nimi ja toiminnan määrittelevä runko. Operaatioilla voi olla myös parametreja ja operaatio voi palauttaa arvon. Olio tallentaa tietoa olion attribuutteihin, jotka ovat nimettyjä tietokenttiä. Attribuuttien arvojen yhdistelmät määrittelevät olion tilat. Olion piirteiksi kutsutaan attribuuttien ja operaatioiden yhdistelmää. Jokaisella oliolla on yksilöivä tunniste eli viite. Olio on suojattu kokonaisuus, jota voidaan käyttää vain tietyillä tavoilla. Kaikkia olion piirteitä ei voida käyttää olion ulkopuolelta. [6]

Oliokielissä olioiden piirteet määritellään tavallisesti olion luokassa ja jokaisella oliolla on yksikäsitteinen luokka, jonka mukaan olio on luotu. Oliota kutsutaan luokan ilmentymäksi. Luokka määrittelee olioiden attribuutit ja operaatiot. Kaikilla luokasta muodostetuilla olioilla on samat attribuutit ja operaatiot, mutta attribuuttien arvot vaihtelevat. Luokka on staattinen ja käännösaikainen käsite. Sen sijaan olio on dynaaminen ja ajonaikaisesti luotava rakenne. Luokat on kiinnitetty jo ennen järjestelmän käynnistämistä, kun taas oliot syntyvät ja kuolevat järjestelmän toiminnan aikana. Luokka ei ole välttämätön käsite olio-ohjelmoinnissa, koska uusia olioita voidaan luoda kloonaamalla olemassa olevista olioista. Suurin osa oliokielistä käyttää kuitenkin luokkia. [6]

Olion kannalta on tärkeää, että olio pystyy suojaamaan sisäisen tietonsa ulkopuolisilta käyttäjiltä. Olioparadigmassa on siten keskeistä, että olio tarjoaa ulkopuolisille selkeät rajapinnat olion tarjoamiin palveluihin, joiden sisäinen toteutus on piilotettu. Tämä suojaus mahdollistaa olion keskinäisten riippuvaisuuksien rajoittamisen rajapintoihin, mikä mahdollistaa olioiden paremman ylläpidettävyyden ja uudelleenkäytön. Olion suojaukset asetetaan yleensä luokan määrittelyissä. Yleinen periaate on, että olion attribuutit ovat suojattuja ulkopuoliselta käytöltä. Attribuuttien arvoja muutetaan operaatioilla, joita voidaan käyttää ulkopuolelta. Myös osa operaatioista eli lähinnä paikalliset apurutiinit voidaan suojata ulkopuoliselta käytöltä. [6]

Yksi olioparadigman keskeisimmistä käsitteistä on periytyminen. Periytyminen tarkoittaa luokan piirteiden siirtymistä toiselle luokalle. Luokka, josta piirteet periytyvät, on yliluokka, ja luokka, jolle piirteet siirtyvät on aliluokka. Periytyminen

voidaan jakaa yksittäisperiytymiseen ja moniperiytymiseen. Yksittäisperiytymisessä aliluokka voi periä piirteitä vain yhdeltä yliluokalta, kun moniperiytymisessä yksittäisellä aliluokalla voi olla useampia yliluokkia. Moniperiytyminen voi helpottaa joissakin tilanteissa merkittävästi luokkahierarkian muodostamista, mutta toisaalta moniperiytyminen altistaa helposti virheille. Siksi moniperiytymisen merkityksestä on kiistelty paljon ja kaikki oliopohjaiset ohjelmointikielet (esimerkiksi Java) eivät tue moniperiytymistä. [6]

Periytyminen mahdollistaa koodin uudelleenkäytön, koska aliluokkien ei tarvitse toistaa perittyjä piirteitä. Samoja yleiskäyttöisiä perusluokkia voidaan käyttää useissa sovelluksissa. Lisäksi periytyminen helpottaa sovellukseen liittyvän käsitehierarkian mallintamista ohjelmistokomponentteihin eli luokkiin. [6]

2.2.2 Olioparadigman edut

Ensimmäinen oliokieli oli 1960-luvun lopulla Norjan valtion laskentakeskuksessa simulointeja varten kehitetty Simula-kieli. Olioperustaisen ohjelmoinnin läpimurto tapahtui kuitenkin vasta 1980-luvun puolessa välin. 1990-luvulla suurin osa kaupallisista ohjelmistoista kehitettiin oliopohjaisesti. Oliopohjaisten ohjelmointikielien, kääntäjien ja suunnittelumenetelmien kehittyminen kypsälle tasolle sekä olioparadigman sisäistäminen ohjelmistokehittäjien keskuudessa vei oman aikansa.

Suurin syy olioperustaisen ohjelmistokehityksen suosiolle on uudelleenkäytettävyys, jonka oliopohjaisuus mahdollistaa muita menetelmiä paremmin. Ohjelmistokomponenttien uudelleenkäytettävyys johtaa nopeampaan ohjelmistokehitykseen ja laadukkaampiin ohjelmiin, koska voidaan käyttää valmiiksi testattuja komponentteja. Tämä on tärkeää, koska ohjelmistojen kehitys- ja ylläpitokustannukset ovat kasvaneet jatkuvasti ohjelmistojen monimutkaistumisen myötä. Olioperustaisten menetelmien käyttö ei välttämättä johda uudelleen käytettäviin komponentteihin, vaan ohjelmisto on edelleen suunniteltava tietoisesti uudelleenkäytettäväksi. [6] [7]

Toinen oliopohjaisuuden keskeinen etu on kyky mallintaa sovellusalueen käsitteitä. Tämän seurauksena järjestelmän toiminnalliset muutokset eivät yleensä aiheuta suuria muutoksia olioperustaiseen ohjelmistorakenteeseen, koska toiminnalliset muutokset eivät muuta käsitemaailmaa. Tämä johtaa helpommin ylläpidettäviin sovelluksiin. [6]

2.2.3 Oliotietokannat

Relaatiomallilla on selkeät puutteensa, vaikka malli on yleisesti hyväksytty ja suosittu tietokannoissa. Relaatiomallin puutteita ovat mm. [5]:

- Todellisen maailman käsitteiden hankala mallinnettavuus relaatiotauluihin.
- Semanttinen ylikuormitus, jolla tarkoitetaan puutteellista käsitteiden yhteyksien mallintamista.
- Puutteellinen datan eheyden hallinta, joka joudutaan usein toteuttamaan sovellusohjelmissa.
- Taulujen rakenne ei sovellu dynaamiselle tiedolle, koska taulujen rakenne on horisontaalisesti ja vertikaalisesti yhtenäinen.
- Rajoitettu määrä operaatioita.
- Olemassa olevan taulurakenteen muuttaminen on hankalaa.
- Ohjelmointikielien tietorakenteiden ja tietotyyppien yhteensovittaminen relaatiomalliin on työlästä.

Myös sovellusohjelmien kehittyminen on aiheuttanut haasteita relaatiotietokannoille. Esimerkiksi multimediaohjelmilla on usein tarve tallentaa ääni- ja videotiedostoja tietokantaan. Relaatiotietokannat eivät sovellu tähän hyvin, vaikka useimmat nykyaikaiset relaatiotietokantojen tiedonhallintajärjestelmät mahdollistavat binääritiedostojen tallentamisen. [5]

Näitä ongelmia on pyritty ratkaisemaan laajentamalla relaatiomallia ja kehittämällä SQL-standardiin uusia ominaisuuksia. Monet tutkijat ja tietokanta-asiantuntijat uskovat kuitenkin, että relaatiomalli ei pysty vastaamaan uusien sovellusten vaatimuksiin. Oliotietokannat pyrkivät yhdistämään olioparadigman edut tietokantoihin.

Oliotietokannoille on olemassa useita erilaisia toteutuksia, mutta yleisesti ottaen oliotietokannat yhdistävät oliopohjaisuuden ja tietokantojen ominaisuudet. Oliotietokannat mahdollistavat olioiden tallentamisen tietokantaan ja tukevat tiedonhallintajärjestelmien yleisiä ominaisuuksia, kuten kyselyitä, eheyden hallintaa ja virhetilanteista elpymistä. Relaatiotietokantoihin verrattuna oliotietokannoissa on tiedon perusyksikkönä tietueen sijaan olio. [5] [8]

Oliotietokannat tarjoavat relaatiotietokantoihin verrattuna enemmän ominaisuuksia, mutta silti oliotietokannat ovat harvinaisia. Oliotietokantojen yleistymistä on haitannut mm. eri valmistajien oliotietokantojen yhteensopimattomat toteutustavat, standardoidun kyselykielen puute, tietokantasuunnittelijoiden kokemuksen puute oliotietokannoista ja oliotietokantojen monimutkaisuus. Myös oliotietokantojen nopeudesta verrattuna relaatiotietokantoihin on ristiriitaisia kokemuksia. Oliotietokantojen harvinaisuudesta johtuen olio-pohjaiset sovellukset joutuvat yleensä tallentamaan tietoa relaatiotietokantoihin ja suorittamaan tiedon muuntamista oliomuodosta relaatiomallin mukaiseen esitystapaan. Päinvastaisia muunnoksia tarvitaan oliopohjaisen sovelluksen suorittaessa tiedon hakuja relaatiotietokannoista. [4] [5]

2.3 Olioiden kartoittaminen relaatiotietokantaan

2.3.1 Ristiriidat mallien välillä

Oliopohjaisessa sovelluskehityksessä aiheuttaa usein ongelmia relaatio- ja oliomallien yhteensovittaminen. Relaatiotietokannat ovat yleisesti käytössä, kun taas oliotietokannat ovat harvinaisia. Sovelluksissa oliomuodossa olevan tiedon tallentaminen relaatiotietokantoihin ei ole kuitenkaan ongelmatonta. Lisäksi oliopohjaiset suunnittelumenetelmät, kuten UML (Unified Modeling Language), eivät tarjoa menetelmiä olioiden kartoittamiseen relaatiomallin mukaiseen esitysmuotoon.

Relaatiotietokannoissa voidaan käyttää kolmea erilaista yhteystyyppiä. Taulujen välinen yhteystyyppi voi olla yhden suhde yhteen (1:1), yhden suhde moneen (1:n) tai monen suhde moneen (n:n), joista n:n yhteystyyppi toteutetaan linkkitaulun avulla.

Oliomallissa on käytössä enemmän vaihtoehtoja yhteystyyppien luontiin. Käytössä on relaatiomallin yhteystyyppien lisäksi useita tietorakenteita, joiden avulla olioiden väliset yhteydet voidaan luoda. Yhteydet voidaan luoda käyttäen esimerkiksi pinoja, listoja, jonoja, puita ja karttoja. Näiden tietotyyppien esittäminen relaatiotietokannassa on ongelmallista. [4]

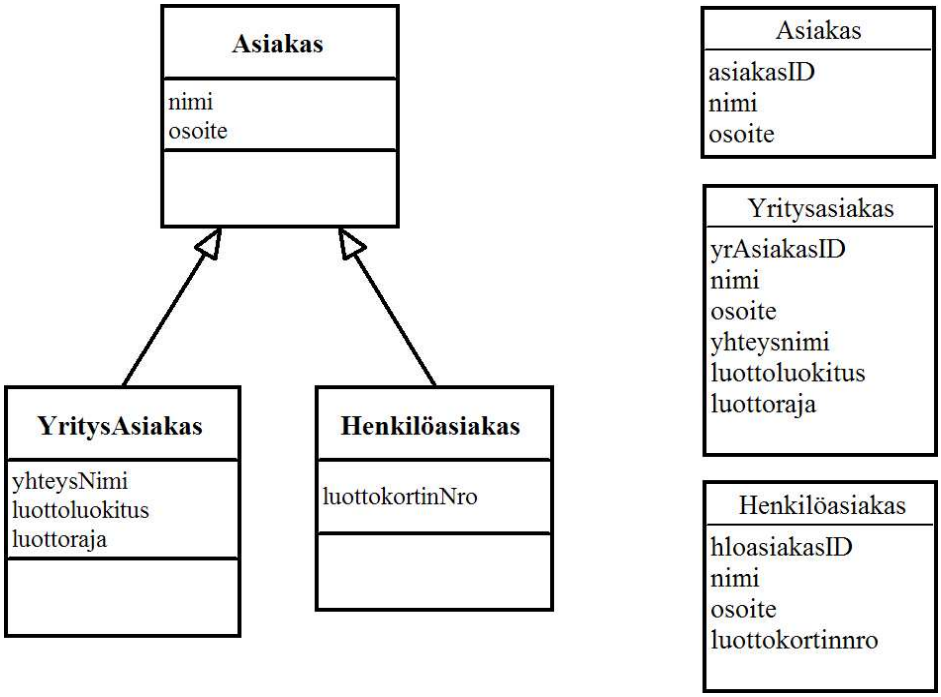
Relaatiotietokannat eivät tue myöskään periytymistä, mikä hankaloittaa oliohierarkioiden mallintamista relaatiotietokantaan. Osa relaatiotietokantojen tiedonhallintajärjestelmistä tukee taulutason periytymistä, mutta yleisesti ottaen relaatiomalli ja relaatiotietokannat eivät tue periytymistä. Luokkien hierarkkisuus voidaan kartoittaa relaatiotietokantaan kolmella eri tavalla: [4]

1. Kartoitetaan kaikki luokat erillisiin tauluihin.
2. Kartoitetaan kaikki luokat yhteen tauluun.
3. Kartoitetaan aliluokkien yksilölliset ominaisuudet erillisiin tauluihin.

Nämä vaihtoehdot käydään läpi seuraavissa kappaleissa tarkemmin.

2.3.2 Luokkien kartoittaminen erillisiin tauluihin

Intuitiivinen vaihtoehto ratkaista luokkien kartoittaminen relaatiotietokantaan on luoda jokaiselle luokkahierarkian luokalle erillinen taulu. Tällöin yhden luokan tiedot saadaan helposti yhdellä kyselyllä yhdestä taulusta. Kuitenkin luokkien tietyntyyppisten yhteyksien toteuttaminen on mahdotonta. On mahdotonta luoda kolmannesta taulusta yhteystyyppi ali- ja yliluokan tauluihin, koska ne ovat erillisiä tauluja. Myös esimerkiksi haut, joissa halutaan hakea yhteen luokkahierarkiaan kuuluvat tiedot ovat hankalia, koska joudutaan käyttämään useita tauluja. Kuvassa 2 luokkahierarkian jokaiselle luokalle on luotu oma relaatiotaulu. [4]



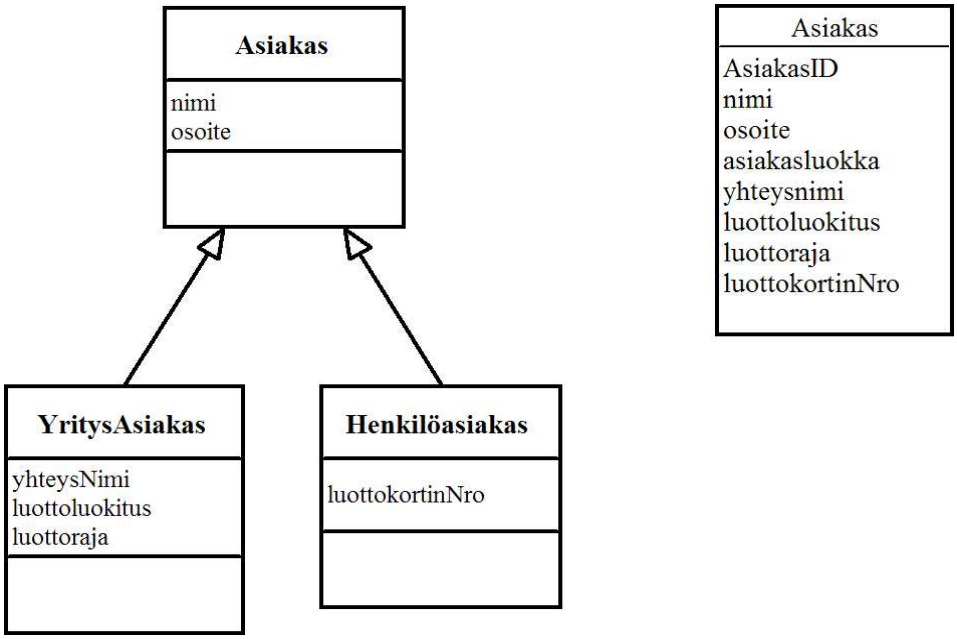
Kuva 2. Luokkahierarkian luokat on kartoitettu erillisiin relaatiotauluihin. [2]

2.3.3 Luokkien kartoitus yhteen tauluun

Toisen ääripään vaihtoehto on kartoittaa kaikki luokkahierarkian luokat yhteen tauluun. Tämä on suositeltava ratkaisu erityisesti silloin, kun aliluokilla on vähän yliluokasta poikkeavia ominaisuuksia. Jos aliluokilla on paljon omia ominaisuuksia, niin tietokannassa joudutaan varaamaan tilaa sarakkeille, joita käytetään harvoin. Tietokannan fyysisestä toteutuksesta riippuen tämä ei kuitenkaan ole yleensä suuri ongelma. [2]

Ratkaisu ei kuitenkaan ole optimaalinen tiedon eheyden suhteen, koska sarakkeille ei voida käyttää ”not null”-määritelmää eli sarakkeille joudutaan sallimaan tyhjät arvot. Tämä johtuu siitä, että yhteen tauluun tallennetaan erityyppisiä olioita. Osalle olioista tietty ominaisuus on pakollinen, mutta toisilla olioilla ominaisuutta ei välttämättä edes ole. Myös yhteystyyppien toteuttaminen on hankalaa, koska yhteystyyppien on pädettävä kaikkiin taulussa oleviin olioihin. Tietokanta on selvillä vain viiteavainsarakkeista ja tietokannalle ei voida määritellä, että viiteavain kuuluu vain taulun tietyn oliotyypin riveihin. [4]

Suorituskyvyn kannalta tämä vaihtoehto on yleensä paras ja siten ratkaisu on useimpiin tapauksiin suositeltava. Suorituskyky on hyvä, koska liitosten määrä tietokantahauissa pysyy vähäisenä. Valitettavasti suorituskyvyn takia joudutaan tekemään kompromisseja tallennustilan ja tietokannan eheyden suhteen. Kuvassa 3 on esitetty luokkahierarkian luokkien kartoittaminen yksittäiseen relaatiotauluun.



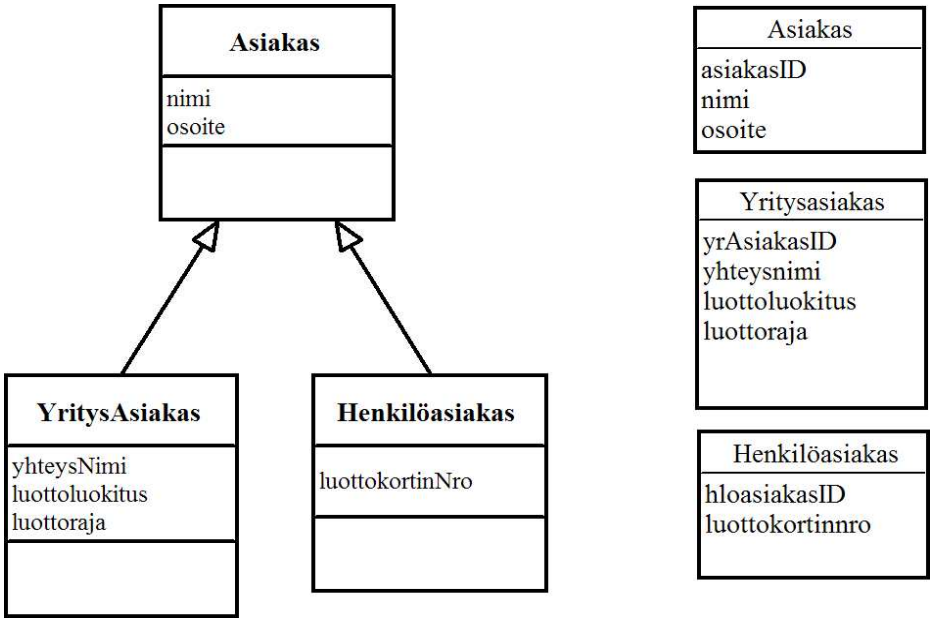
Kuva 3. Luokkahierarkian luokat on kartoitettu yksittäiseen relaatiotauluun. [2]

2.3.4 Aliluokkien jako erillisiin tauluihin

Kolmas vaihtoehto on mallintaa luokkien yhteiset ominaisuudet yhteen tauluun ja aliluokkien yksilölliset ominaisuudet erillisiin tauluihin. Aliluokista muodostetuissa tauluissa on viiteavaimet yliluokasta muodostettuun tauluun. Ratkaisu on lähellä oliosuunnittelun mallintamistapaa. [2] [4]

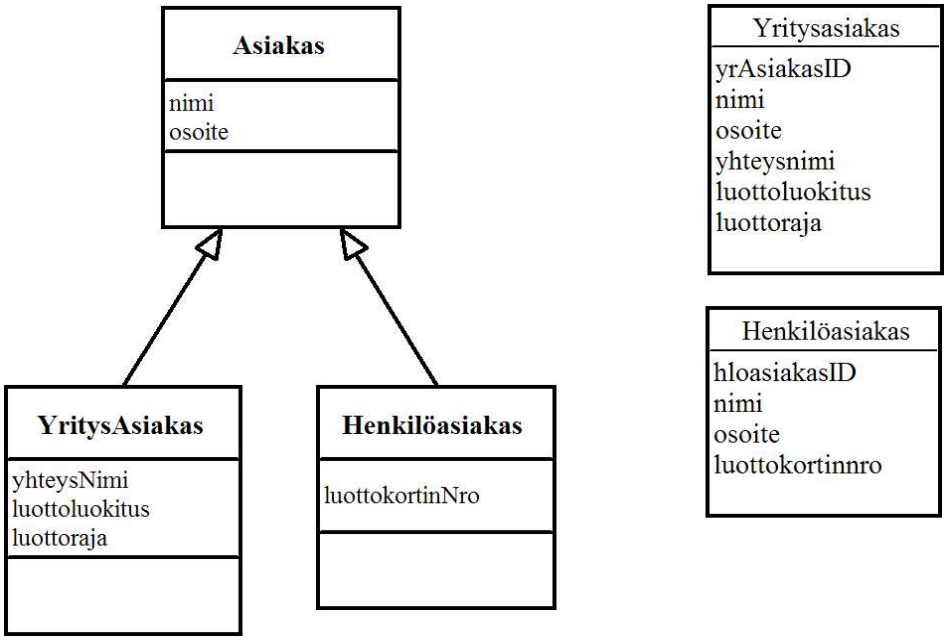
Tämä ratkaisu on toimivin silloin, kun luokilla on paljon toisistaan poikkeavia attribuutteja. Taulujen suuri määrä aiheuttaa kuitenkin hakujen hidastumisen, koska aikaa vieviä liitosoperaatioita joudutaan suorittamaan paljon. Ratkaisun etuna on tallennustilan optimaalinen käyttö, koska tilaa ei tuhlata ominaisuuksiin, joita olioilla ei ole. Aliluokkiin voidaan luoda yhteystyyppejä määrittelemällä viiteavain

aliluokkatietoon. Tämä mahdollistaa, että tauluihin ei voida määritellä virheellisiä yhteyksiä kuten esimerkiksi edellisessä kappaleessa esitetyssä ratkaisussa. Ratkaisu sopiikin parhaiten sovelluksiin, joissa tallennustilan optimointi ja tiedon eheyden varmistaminen ovat tärkeämpiä tekijöitä kuin suorituskyky. Myös tyhjät arvot voidaan kieltää ”not null”-määritelmällä, koska jokaisella luokalla on oma taulu. Esimerkkitapaus tästä menetelmästä on esitetty kuvassa 4. [2] [4]



Kuva 4. Esimerkki aliluokkatiedon jakamisesta erillisiin relaatiotauluihin. [2]

Menetelmän variaatio on kopioida yliluokan ominaisuudet aliluokkiin ja poistaa yliluokkaan liittyvä taulu. Tällöin taulujen määrä vähenee ja samalla tietokantahaut nopeutuvat. Tämä on suositeltavaa erityisesti silloin, kun yliluokalla on vähän omia ominaisuuksia verrattuna aliluokkiin. Kuvassa 5 on luokkahierarkian yliluokan ominaisuudet yhdistetty aliluokista luotuihin relaatiotauluihin. [2]



Kuva 5. Yliluokan ominaisuudet on yhdistetty aliluokkien relaatiotauluihin. [2]

2.3.5 Muita menetelmiä olio- ja relaatiomallien yhteensovittamiseen

Oliopohjaisissa sovelluksissa olioiden ja relaatioiden käsittely helpottuu, kun sovelluksessa ei käsitellä relaatiotietokantaa suoraan. Sovelluksen ja relaatiotietokannan välille voidaan luoda välikerros, joka piilottaa relaatiotietokannan taulurakenteet sovellukselta. Tällöin sovellus ei käsittele relaatiotietokantaa suoraan, vaan sovelluksen ja relaatiotietokannan välinen tiedon haku ja tallennus tapahtuu välikerroksen kautta. [2]

Oliopohjaiset sovellukset ja relaatiotietokannat käsittelevät tietoa eri tavalla. Kun oliomaailmassa halutaan käsitellä suurta joukkoa tietoa, niin yleensä oliot tallennetaan johonkin säiliöluokkaan. Säiliöluokassa olioita käsitellään yksi kerrallaan iteratiivisesti. Relaatiomaailmassa taas käsitellään suuria riviryhmiä ja ryhmien välisiä yhteyksiä yhdellä kerralla. Molemmat tavat ovat tehokkaita, mutta ongelmia aiheuttaa, kun oliopohjaisessa sovelluksessa joudutaan iteratiivisesti hakemaan tietoa relaatiotietokannasta. Tällöin joudutaan tekemään lukemattomia yksittäisiä tietokantahakuja, jotta saataisiin selville relaatiomaailman yhtä SQL-lausetta vastaavat tiedot. Tällöin syntyy helposti suorituskykyongelmia, joita voidaan helpottaa puskuroimalla kyselyiden osoiteoliot välimuistiin. Tällöin ajan myötä kyselyt nopeutuvat, kun tarvittava tieto on välimuistissa. [4]

Yleisimmille oliopohjaisille ohjelmointikielille on olemassa valmiita kirjastoja, jotka suorittavat muunnoksia olioiden ja relaatioiden välillä. Esimerkiksi Oraclen OracleAS Toplink [9] on joukko Javalle tehtyjä kirjastoja ja työkaluja, jotka mahdollistavat mm. attribuuttien ja yhteystyyppien kartoittamisen, luokkahierarkioiden mallintamisen relaatioiksi ja puskuroidun välimuistin käytön. Myös menetelmiä, joilla voidaan muodostaa esimerkiksi UML-pohjaisista luokkakaavioista tietokantojen suunnittelussa käytettäviä ER-kaavioita, on kehitetty.

3 Hajautetut sovellukset

3.1 Tarve sovellusten hajauttamiselle

Tietokoneiden alkuaikoina sovelluksia suoritettiin suurissa keskuskoneissa. 70-luvulla tietokannat yleistyivät ja samalla ensimmäiset hajautetut sovellukset syntyivät. Tällöin sovellukset saattoivat siirtää vastuun tiedon tallettamisesta ja hallinnasta tiedonhallintajärjestelmälle. 80-luvulla paikallisverkkojen suosio ja 90-luvulla Internetin yleistyminen lisäsivät hajautettujen sovellusten lukumäärää. 2000-luvun alussa uutta hajautettujen sovellusten sukupolvea edustaa mm. Grid-teknologia, joka mahdollistaa fyysisesti eri paikoissa sijaitsevan laskenta- ja tallennuskapasiteetin hyödyntämisen.

Hajauttamalla sovellus useampaan komponenttiin yhden ison kokonaisuuden sijaan pyritään saavuttamaan etuja mm. ylläpidettävyydessä, luotettavuudessa, käytettävyydessä ja laajennettavuudessa. Hajautetut sovellukset skaalautuvat usein paremmin suurille käyttäjämäärille ja ovat helpommin hallittavia. Esimerkiksi sovelluksen käyttövarmuus eli käytettävyys paranee, kun sovelluksen osia voidaan suorittaa eri palvelimilla. Tällöin yksittäisen palvelimen vikatilanteet eivät välttämättä vaikuta loppukäyttäjään. Sovelluksen osia voidaan myös muokata riippumatta muista osista, mikä helpottaa sovelluksen ylläpitoa. [4]

3.2 Sovelluksen kerrokset

Sovellukset koostuvat yleensä kolmesta kerroksesta. Nämä kerrokset ovat data-, toimintalogiikka- ja esityskerros. Datakerros hallitsee sovelluksen käsittelemää dataa. Toimintalogiikkakerros sisältää toimintasäännöt ja operaatiot, jotka määrittelevät, kuinka dataa käsitellään. Esityskerros toteuttaa käyttöliittymän ja on vuorovaikutuksessa sovelluksen käyttäjän kanssa. [4]

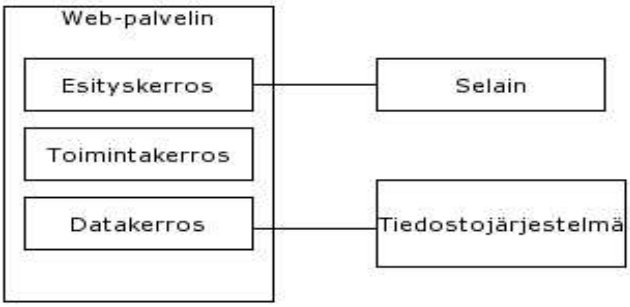
Sovellusarkkitehtuureissa käytetään käsitettä taso ryhmittämään sovelluksen kolme kerrosta yhdeksi komponenteiksi. Sovellukset voidaan luokitella tasojen määrään perusteella yksi-, kaksi-, kolmi- tai n-tasoiksi sovelluksiksi. Tasojen ei tarvitse

välttämättä sijaita erillisissä koneissa tai edes erillisissä ohjelmissa. [4] Seuraavissa kappaleissa esitellään nämä sovellustasot selainpohjaisten sovellusten näkökulmasta ja pohditaan sovellusarkkitehtuurin valintaan vaikuttavia tekijöitä.

3.3 Sovellustasot

3.3.1 Yksitasoiset sovellukset

Yksitasoisen sovelluksen data-, toiminta- ja esityskerros ovat yhdessä komponentissa. Yksitasoisen sovelluksen arkkitehtuuri on havainnollistettu kuvassa 6 selainpohjaisen sovelluksen näkökulmasta.

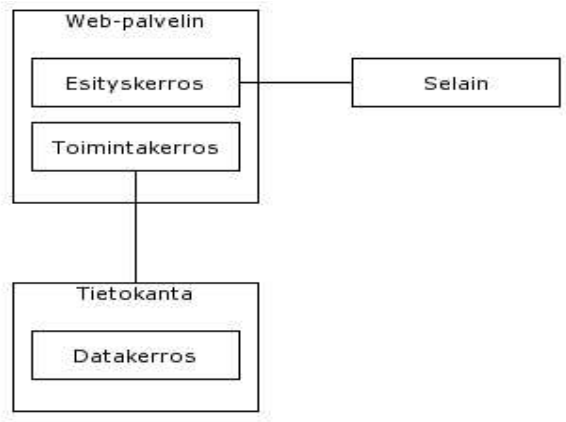


Kuva 6. Yksitasoisen sovelluksen arkkitehtuuri. [4]

Usein yksitasoisissa sovelluksissa on suhteellisen monimutkainen esityskerros. Sen sijaan toiminta- ja datakerrokset ovat yksinkertaisia. Yksitasoiset sovellukset tallentavat tietonsa yleensä tekstitiedostoihin tietokantojen sijaan. Esimerkiksi verkossa suoritettavat mielipidekyselyt tai adressit toteutetaan usein yksitasoisina sovelluksina. [4]

3.3.2 Kaksitasoiset sovellukset

Kaksitasoisissa sovelluksissa datakerros on monimutkainen ja datakerroksen toteuttaminen ohjelmassa vaatisi runsaasti ohjelmakoodia. Tällöin vastuu datakerroksesta siirretään tietokannalle. Kaksitasoisen sovelluksen arkkitehtuuri on havainnollistettu kuvassa 7.

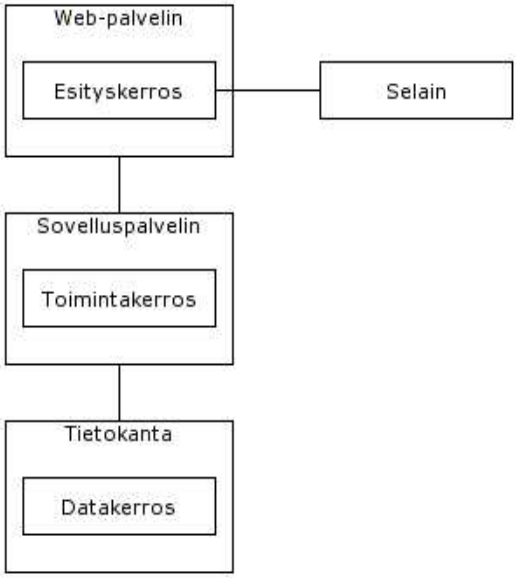


Kuva 7. Kaksitasoisen sovelluksen arkkitehtuuri. [4]

Kaksitasoisen sovelluksen etu seuraavassa kappaleessa esiteltävään kolmitasoiseen malliin verrattuna on, että datakerroksen suunnitteluun on olemassa huomattavasti enemmän työkaluja kuin toimintalogiikan toteuttamiseen. Tämän takia kaksitasoinen malli on varteenotettava vaihtoehto erityisesti projekteissa, joissa sovelluksen toimintalogiikka ei ole erityisen monimutkainen ja projektin aikataulu on tiukka. [4]

3.3.3 Kolmitasoiset sovellukset

Kolmitasoisissa sovelluksissa data-, toimintalogiikka- ja esityskerrokset on hajautettu omiin komponentteihin. Toimintakerros suoritetaan sovelluspalvelimella. Kolmitasoisen sovelluksen arkkitehtuuri on esitetty kuvassa 8.



Kuva 8. Kolmitasoisen sovelluksen arkkitehtuuri. [4]

Kolmitasoisen mallin etuna on joustavuus, sillä erillisiin komponentteihin voidaan tehdä muutoksia vaikuttamatta sovelluksen muiden osien toimintaan. Kuitenkin mallin toteuttaminen käytännössä on monimutkaista. Esimerkiksi komponenttien väliset rajapinnat on suunniteltava huolellisesti. Selainpohjaisissa verkkosovelluksissa esitys- ja toimintalogiikkakerroksien välinen rajapinta on haastava, koska selainpohjaisen sovelluksen vuorovaikutus toimintalogiikkakerroksen kanssa on hitaampaa kuin tavallisissa graafisissa käyttöliittymissä. [4]

Kolmitasoiset sovellukset käyttävät sovelluskomponenttien väliseen kommunikointiin yleensä oliovälitysohjelmistoa (middleware). Tällainen oliovälitysohjelmisto on esimerkiksi Object Management Group -yhteenliittymän CORBA (Common Object Request Broker Architecture). CORBA on oliopyyntöjen välitysarkkitehtuuri, jonka tärkein osa on ORB (Object Request Broker). ORB-ohjelma välittää asiakasohjelman kutsut palveluja tarjoavalle oliolle. ORB-ohjelma mahdollistaa verkkoläpinäkyvyyden, eli asiakkaan ei tarvitse tietää, missä verkon osassa palvelua tarjoava olio sijaitsee. [4] [10]

3.3.4 N-tasoiset sovellukset ja hybridiarkkitehtuurit

N-tasoiset sovellukset ovat samankaltaisia kuin kolmitasoiset sovellukset, mutta vievät hajautuksen pidemmälle. Esimerkiksi tietokantakerros voi koostua useista erilaisista tietokannoista ja käyttöliittymäkerros eri laitteille toteutetusta käyttöliittymäkomponenteista. N-tasoisella sovelluksella on useita hajautettuja olioita eri koneilla. Oliot käyttävät kommunikointiin esimerkiksi CORBA:a. [4] [10]

Koko sovelluksen ei tarvitse käyttää samaa arkkitehtuuria. Pääsovellus voi olla 3-tasoinen, mutta sovelluksen ylläpitotyökalu voi noudattaa kaksitasoarkkitehtuuria. Useat käytännön sovellukset ovatkin sekoituksia eri malleista. Tällaisten sovellusten arkkitehtuurista käytetään usein nimitystä hybridiarkkitehtuuri. [4]

3.4 Sovellusarkkitehtuurin valinta

Sovellusarkkitehtuurin valinta ohjelmistoprojektille ei ole yksinkertaista. Sovelluksen monimutkaisuus vaikuttaa käytettävien tasojen valintaan. Monimutkainen sovellus on syytä jakaa useampiin ohjelmistokomponentteihin kuin yksinkertainen sovellus. Tällöin eri komponenttien toteuttamiseen voidaan keskittyä yksitellen ja komponenttien kehitys jakaa sovelluskehittäjien välille. [4]

Osa sovelluksista tehdään nopeasti ratkaisemaan määritelty ongelma. Jos sovellus täytyy saada nopeasti valmiiksi ja on todennäköistä, että sovellusta ei jatkossa kehitetä, niin usein paras ratkaisu on käyttää mahdollisimman yksinkertaista sovellusarkkitehtuuria. Sen sijaan säännöllisesti ylläpidettävää ja kehitettävää sovellusta luotaessa monitasoarkkitehtuuri helpottaa ylläpitoa ja jatkokehitystä. Tällöin kuitenkin suunnitteluun käytettävä aika lisääntyy. [4]

Myös sovelluskehittäjien taito ja kokemus vaikuttaa sovellusarkkitehtuurin valintaan. Luotettavan kolmitasoisen arkkitehtuurin suunnittelu voi olla liian vaativaa, jos suunnittelijat ovat kokemattomia. Monitasoarkkitehtuuria käyttävän sovelluksen suunnittelijoilla täytyy olla kokemusta käytettävistä teknologioista ja hajautettujen sovelluskomponenttien suunnittelusta. Kokemattomammat sovelluskehittäjät voivat

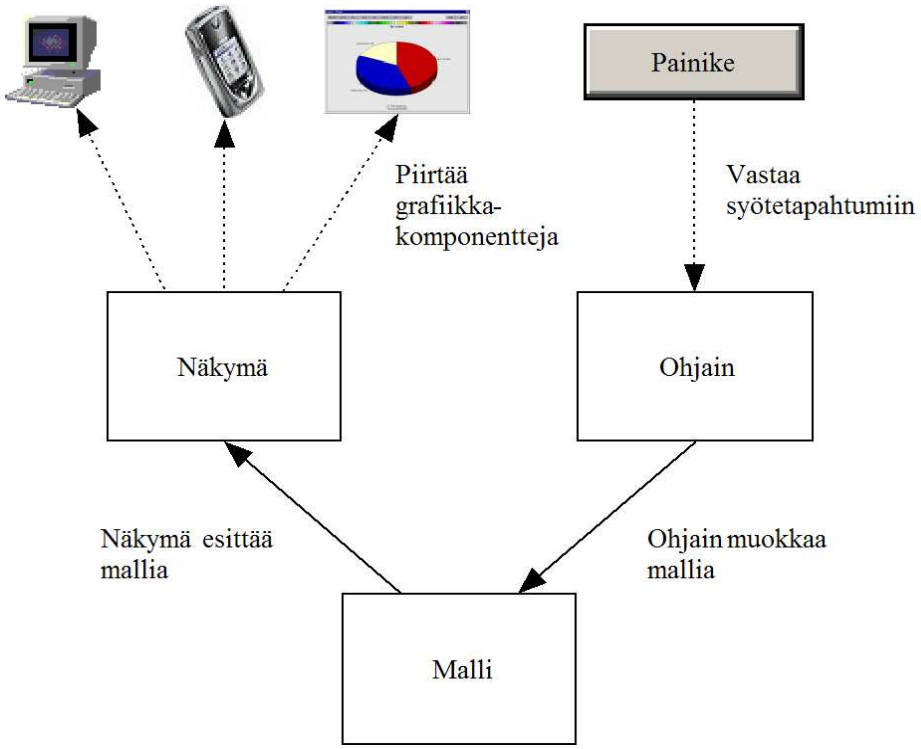
tehdä varsinaisen toteutuksen. On tärkeää, että myös toteuttajat pääsevät osallistumaan yhteisiin suunnittelukokouksiin, jolloin heidän osaamisensa arkkitehtuurisuunnittelusta paranee. Jos kokemattomat suunnittelijat kehittävät yksin monitasoarkkitehtuuria, niin yritysten ja erehdysten kautta päädytään helposti arkkitehtuuriin, jossa kerrosten väliset rajat ovat epäselviä. Lisäksi oliosuunnittelun puutteellinen ymmärtäminen johtaa koodin tuottamiseen kopioimalla valmista koodia, vaikka voitaisiin hyödyntää periytymistä. [4] [11]

Yhteenvetona voidaan sanoa, että monitasoarkkitehtuuria puoltavia tekijöitä ovat sovelluksen monimutkaisuus ja ylläpidettävyys. Aikataulun kireys ja sovelluskehittäjien taidot voivat kuitenkin johtaa yksinkertaisemman yksi- tai kaksitasoisen sovelluksen toteuttamiseen. Lisäksi ohjelmistoarkkitehtuurin valintaan vaikuttavat useat muut tekijät, kuten olemassa olevien sovellusten arkkitehtuuri.

3.5 MVC-paradigma

Myös MVC-paradigma (Model-View-Controller) on eräs tapa hajauttaa sovellus. MVC-paradigman lähtökohtana on malli (model), joka edustaa käsiteltävää dataa. Oliopohjaisissa sovelluksissa malli koostuu dataa käsittelevistä luokista. Mallin datalle luodaan näkymiä (view), jotka esittävät datan käyttäjälle näytöllä. Ohjain (controller) hyväksyy käyttäjän syötteet ja päivittää niiden mukaan mallia tai näkymää. [4]

MVC-mallissa datan esitys ja käsittely ovat erillään datasta. Tämä helpottaa sovelluksen ylläpitoa ja sovittamista erilaisille päätelaitteille. MVC-mallia käyttävä sovellus voidaan sovittaa erikokoisille päätelaitteille muokkaamalla vain sovelluksen näkymästä vastaavaa osaa. MVC-malli on havainnollistettu kuvassa 9.



Kuva 9. MVC-paradigman mukaisen sovelluksen toiminta. [4]

4 Selainpohjaisen sovelluksen toteuttaminen palvelimella

4.1 HTTP-protokolla ja selainpohjaiset sovellukset

Internet tuli suuren yleisön tietoisuuteen 90-luvun puolivälissä. Aluksi www-sivut olivat staattisia, mutta pian yleistyivät myös www-selainohjelmien avulla käytettävät sovellukset. Selainpohjaisten verkkosovellusten suosio on ollut jatkuvassa kasvussa. Selainpohjaisia sovelluksia käytetään sekä julkisessa Internetissä että yritysten Intraneteissä. Suosittuja sovellusalueita ovat esimerkiksi verkkokaupat, varausjärjestelmät ja pankkipalvelut. Selainpohjaisten sovellusten etuja ovat mm. käyttöjärjestelmäriippumattomuus ja mahdollisuus käyttää sovellusta miltä tahansa koneelta, johon on asennettu selainohjelma.

Internet ja yritysten Intranetit koostuvat palvelimista, jotka välittävät tietoa keskenään tarkoin määriteltyjen protokollien avulla. Selainpohjaisia sovelluksia kehitettäessä matalan tason protokollien, kuten tietoa kuljettavien ja reitittävien TCP (Transmission Control Protocol) ja IP (Internet Protocol) protokollien, tunteminen ei ole yleensä tarpeellista. Sovelluskehittäjän kannalta ensimmäinen merkittävä protokolla on HTTP (Hypertext Transfer Protocol), jonka avulla www-palvelin ja asiakasohjelmat keskustelevat.

HTTP-protokolla määrittelee joukon komentoja ja komentojen vastauskoodeja, joista käytetään nimitystä tilakoodi. Jokainen HTTP-tapahtuma koostuu neljästä osasta: [12]

1. TCP-yhteyden muodostus.
2. Asiakkaan lähettämä palvelupyyntö.
3. Vastaus palvelimelta.
4. Yhteyden sulkeminen palvelimeen.

HTTP-protokollassa selainohjelman ja palvelimen välinen yhteys muodostetaan käyttäen TCP-protokollaa. Käytännössä yhteyden muodostaminen alkaa, kun käyttäjä on syöttänyt URL-osoitteen (Uniform Resource Locator) selainohjelmaan tai valinnut linkin www-sivulta. Yhteyden muodostamisen jälkeen selainohjelma pyytää

haluamaansa tietoa palvelimelta, joko GET-, POST- tai HEAD-metodilla. GET- ja POST-metodien suurin ero on, että GET-metodi koodaa välitettävän tiedon URL-osoitteeseen. POST-metodissa tiedot koodataan välitettävään viestiin. GET-metodia voidaan käyttää tiedon hakemiseen ja lähettämiseen, mutta sitä suositellaan käytettäväksi vain tiedon hakuihin. POST-metodia voidaan käyttää molempiin tarkoituksiin. HEAD-metodin avulla voidaan pyytää otsikkotietoja, jotka välittyvät myös muissa HTTP-pyynnöissä. Esimerkiksi selainohjelman välittämät otsikkokentät sisältävät yleensä tietoa mm. käytettävästä käyttöjärjestelmästä, selainohjelman versiosta ja selainohjelman hyväksymistä tiedostomuodoista. [12]

HTTP-pyynnön vastaanottamisen jälkeen www-palvelin siirtää selainohjelman pyytämän objektin. Objekti on yleensä tekstimuodossa oleva HTML-tiedosto, mutta voi olla myös muuta digitaalisessa muodossa esitettyä dataa. Asiakas ja palvelin keskustelevat keskenään merkkimuotoisilla komennoilla ja tilakoodeilla eli vasteilla. Tilakoodit näkyvät käyttäjille lähinnä virhetilanteissa. Esimerkiksi tilakoodi 404 tarkoittaa, että haluttua objektia ei löydy palvelimelta. TCP-yhteys suljetaan, kun kaikki tieto palvelimelta asiakasohjelmalle on siirretty. HTTP-protokolla on tilaton eli yhteyden sulkemisen jälkeen yhteydestä ei tallenneta mitään tietoja seuraavaa yhteyttä varten. [12]

Selainohjelma vastaa palvelimelta saadun datan esittämisestä käyttäjälle. Internet-sivujen esittämiseen käytetään yleensä HTML-sivunkuvauskieltä, jonka standardoinnista vastaa World Wide Web Consortium (W3C). HTML-sivut voivat sisältää myös ohjelmakoodia. Useimmat selaimet tukevat JavaScript-ohjelmointikieltä. JavaScript on ohjelmointikielenä varsin rajoittunut ja soveltuu lähinnä yksinkertaisen toiminnallisuuden, kuten syötteiden tarkistamisen, toteuttamiseen. Selainpohjaisten sovellusten toimintalogiikka toteutetaankin yleensä www-palvelimella. Seuraavissa kappaleissa käsitellään, kuinka toimintalogiikka voidaan toteuttaa www-palvelimella.

4.2 CGI-ohjelmat

CGI (Common Gateway Interface) on palvelimen ja palvelimella suoritettavien ohjelmien välille määritelty standardoitu rajapinta. CGI mahdollistaa dynaamisten palveluiden luomisen. CGI ohjelmat voivat käsitellä www-sivujen lomakkeiden tietoja ja muodostaa yhteyksiä tietokantoihin. CGI on yksinkertainen rajapinta, joka on toteutettu käytännössä kaikkiin www-palvelinohjelmistoihin. [12]

CGI-ohjelma on tavallinen ohjelma tai prosessi, joka kommunikoi selaimen ja palvelimen kanssa STDIN-vakiosyötteen (Standard input) ja STDOUT-vakiotulosteen (Standard Output) avulla. Selainohjelma muodostaa yhteyden palvelinohjelmaan, joka välittää pyynnön CGI-ohjelmalle käyttäen STDIN-vakiosyötettä. Tämän jälkeen CGI-ohjelma käsittelee pyynnön ja välittää vastauksen palvelinohjelmiston kautta selainohjelmalle käyttäen STDOUT-vakiotulostetta. Selain ja palvelinohjelmisto kommunikoivat HTTP-protokollan avulla. Palvelinohjelmisto ja pyynnön käsittelevä ohjelma käyttävät kommunikointiin CGI-rajapintaa. [12] [13]

CGI-ohjelmia voidaan kirjoittaa ohjelmointikielillä, jotka tukevat STDIN-vakiosyötettä ja STDOUT-vakiotulostetta. Käytännössä kaikki ohjelmointikielet soveltuvat CGI-ohjelmien kirjoittamiseen. Perl on suosituin CGI-ohjelmien kirjoittamiseen käytetty ohjelmointikieli yksinkertaisuutensa, hyvien tekstinkäsittelyominaisuuksien ja laajan käyttöjärjestelmätuen ansiosta. CGI-ohjelmat käyttävät hyväkseen palvelinohjelmiston ja palvelimen käyttöjärjestelmän asettamia ympäristömuuttujia. CGI-ohjelmilla on yleensä varsin laajat suoritusoikeudet, vaikka palvelinohjelmisto ja käyttöjärjestelmä voivat asettaa rajoituksia CGI-ohjelmille. [12] [13]

CGI-ohjelmat olivat erittäin suosittuja 90-luvulla, mutta nykyisin laajat verkkosovellukset toteutetaan yleensä PHP:n ja Java-servlettien kaltaisilla tekniikoilla. CGI:n vahvuus ja heikkous on, että se määrittelee vain rajapinnan. Tämä tekee CGI:stä ohjelmointikieliriippumattoman, mutta toisaalta CGI-ohjelmien käytössä ei ole standardeja kirjastoja yleisimpien tehtävien suorittamiseen. Tällaiset kirjastot on toteutettava kolmansien osapuolien toimesta. Tietylle käyttöjärjestelmälle käännetyt CGI-ohjelmat ja ohjelmien käyttämät kirjastot eivät myöskään toimi suoraan toisissa

käyttöjärjestelmissä. Tämä ongelma voidaan ratkaista käyttämällä tulkattavia ohjelmointikieliä, kuten Perl, mutta tällöin suorituskyky voi osoittautua ongelmaksi. Myös prosessien synkronisointi ja yhtäaikaisuuden hallinta aiheuttavat ongelmia. [13]

4.3 Server Side Includes

Server Side Includes eli SSI [14] mahdollistaa erityisten direktiivien sisällyttämisen HTML-koodiin. Direktiivi voi sisältää mm. tietyn ympäristömuuttujan tulostamisen tai kutsun palvelimella sijaitsevaan ohjelmaan. SSI-tekniikalla voidaan siten luoda dynaamista sisältöä CGI-ohjelmien tapaan. Esimerkiksi dokumentin muutospäivämäärä voidaan tulostaa direktiivillä <!--#echo var="LAST_MODIFIED">. SSI-tekniikan hyviä puolia ovat yksinkertaisuus ja toistuvan informaation, kuten sivujen tekijänoikeuksien ja muutospäivämäärien, helppo esittäminen. Toisaalta laajemman toiminnallisuuden toteuttamisessa tekniikan puutteet ovat hyvin samanlaisia kuin CGI-ohjelmien yhteydessä.

4.4 PHP ja ASP

PHP on palvelinpuolen tulkattava ohjelmointikieli, jonka ohjelmakoodi voidaan sijoittaa HTML-koodin yhteyteen tai erilliseen tiedostoon. PHP-ohjelmakoodia ei lähetetä suoraan asiakkaalle, vaan PHP-ohjelma tai -moduuli jäsentää ohjelmakoodin ensin palvelimella. Rasmus Lerdorf julkaisi ensimmäisen version PHP:sta vuonna 1994. Silloin PHP oli vain joukko makroja helpottamaan kotisivun tekemistä. Nykyisin PHP on täysiverinen ohjelmointikieli, jolle on toteutettu runsaasti erilaisia laajennuksia. PHP onkin erittäin suosittu juuri yksinkertaisuutensa ja monipuolisten laajennusten ansiosta. [15]

ASP (Active Server Page) on samankaltainen tekniikka kuin PHP. ASP on kuitenkin rajoittunut Microsoftin Windows käyttöjärjestelmiin ja Internet Information Server -palvelinohjelmistoon. PHP-ohjelmia voidaan suorittaa useissa erilaisissa käyttöjärjestelmissä ja palvelinohjelmistoissa.

4.5 Java-servletit

4.5.1 Java ohjelmointikielenä

Java on Sun Microsystems:in kehittämä oliopohjainen ohjelmointikieli, jonka kehittäminen alkoi vuonna 1991. Alkuperäinen tavoite oli luoda älykäs kulutuselektroniikkaa ohjaava oliopohjainen ohjelmointikieli, mutta lopulta Internet teki ohjelmointikielestä suositun. Aluksi Javaa käytettiin lähinnä HTML-sivuille upotetuissa sovelmissa, joiden tulkitsemiseen selainohjelmissa oli Java-virtuaalikone. Sun Microsystems on kuitenkin kehittänyt Javaa voimakkaasti. Java on nykyisin erittäin suosittu ohjelmointikieli, jota voidaan käyttää lähes kaikkiin mahdollisiin ohjelmointitarkoituksiin matkapuhelinten pienistä J2ME-sovelluksista (Java 2 Micro Edition) laajojen hajautettuja monitasoarkkitehtuureja käyttävien sovellusten toteuttamiseen.

Java on verkkoympäristöön kehitetty, laitteistoista ja käyttöjärjestelmistä riippumaton oliopohjainen ohjelmointikieli. Java on saanut vaikutteita useista olio-ohjelmointikielistä ja sen kielioppi pohjautuu C++-kieleen. Java on kuitenkin C++-kieltä vahvempi olio-ohjelmointikieli. Kaikki muuttujat ja metodit kuuluvat aina luokkaan ja kaikki koodi sijoitetaan rajapintaan tai luokkaan. Olioparadigman sisäistäminen on välttämätöntä Java-sovelluksia suunniteltaessa. Muita ominaispiirteitä ovat automaattinen muistinvapautus ja monisäikeisyys eli ohjelman sisäinen moniajo. [10]

Jokainen Java-ohjelma sekä käännetään että tulkataan. Käännettyä Java-koodia kutsutaan tavukoodiksi (bytecode). Tavukoodi on laitteistoriippumatonta koodia, jonka suorittamiseen tarvitaan Java-tulkki eli virtuaalikone. Virtuaalikone sovittaa tavukoodin jokaiselle prosessorille ja käyttöjärjestelmälle. Virtuaalikone vastaa myös luokkien dynaamisesta latauksesta ajon aikana. Java-ohjelmia on mahdollista suorittaa kaikissa käyttöjärjestelmissä, joissa on Java-virtuaalikone. [10]

Java-laitteistojärjestelmään kuuluu virtuaalikoneen lisäksi Java-sovellusrajapinta eli Java API (Application Programming Interface). Java API koostuu laajasta kokoelmasta valmiita komponentteja. Java API sisältää mm. seuraavat ominaisuudet:

- Perusominaisuudet, kuten oliot, säikeet, tietorakenteet sekä syöte ja tulostus.
- Graafisen käyttöliittymän peruskomponentit.
- Verkko-ominaisuudet.
- Internationalisointi, joka mahdollistaa eri kielillä toteutetut sovellukset.
- Matalan ja korkean tason turvallisuusominaisuudet, kuten elektroniset allekirjoitukset ja salausavainten hallinta.
- JavaBeans-sovelluskomponentit.
- JDBC-rajapinta, joka mahdollistaa tietokantojen käyttämisen. [10]

Javan laajennettavissa oleva API-rajapinta ja monipuoliset ominaisuudet, kuten automaattinen muistinvapautus, nopeuttavat merkittävästi sovelluskehitystä ja vähentävät ohjelmakoodin virheitä. Java on myös ohjelmointikielenä helpompi opetella ja käyttää kuin C++. [10]

Javan ongelmana on pidetty hitautta, joka aiheutuu tavukoodin tulkkaamisesta. Lisäksi numeerisesti vaativan laskennan kannalta Javan ongelmina on pidetty seuraavia tekijöitä:

- Java tarkistaa moniulotteisten taulukoiden dimensiot ajonaikaisesti.
- Java tarkistaa kaikki taulukkoviittaukset poikkeuskäsittelyn kautta.
- Java ei salli koodin uudelleen järjestämistä, mikä estää kääntäjää optimoimasta koodia.
- Perustietotyypit esitetään olioina, mikä estää niiden tallentamisen pinoon ja rekistereihin ohjelman suorittamisen nopeuttamiseksi. [16]

Tavukoodia suorittavat virtuaalikoneet ovat nykyisin erittäin kehittyneitä ja käyttävät mm. JIT-käännöstekniikkaa (Just-In-Time). JIT-käännöksessä Java-tavukoodi käännetään konekielelle ennen ohjelman suorittamista. Tällöin ohjelmien käynnistyminen hidastuu, mutta suoritusnopeus paranee, koska säästytään koodin tulkkaamiselta. HotSpot-tekniikka on toinen Java-ohjelmien suorittamista nopeuttava menetelmä. Siinä tavukoodia ei käännetä ennen ohjelman suorittamista konekieleksi,

vaan sovelluksen tulkkaamisen aikana pyritään optimoimaan sovelluksen osia, joita käytetään paljon. Taulukoiden käytön tehostamiseksi on kehitetty erityisiä ohjelmointikirjastoja, jotka käsittelevät taulukoita tehokkaammin kuin Javan omat taulukkokirjastot. Java-ohjelmointikielen ja -suoritusympäristöjen kehittymisen ansiosta Java-ohjelmat ovat nykyisin usein yhtä nopeita kuin C- ja C++-kielillä toteutetut ohjelmat. [16] [30]

Javan hitauden voi nykyisin huomata lähinnä graafista käyttöliittymää käyttävissä sovelluksissa, koska Javan käyttöliittymäkomponentit eivät ole yhtä optimoituja kuin käyttöjärjestelmien omat käyttöliittymäkomponentit.

4.5.2 Java-servletit

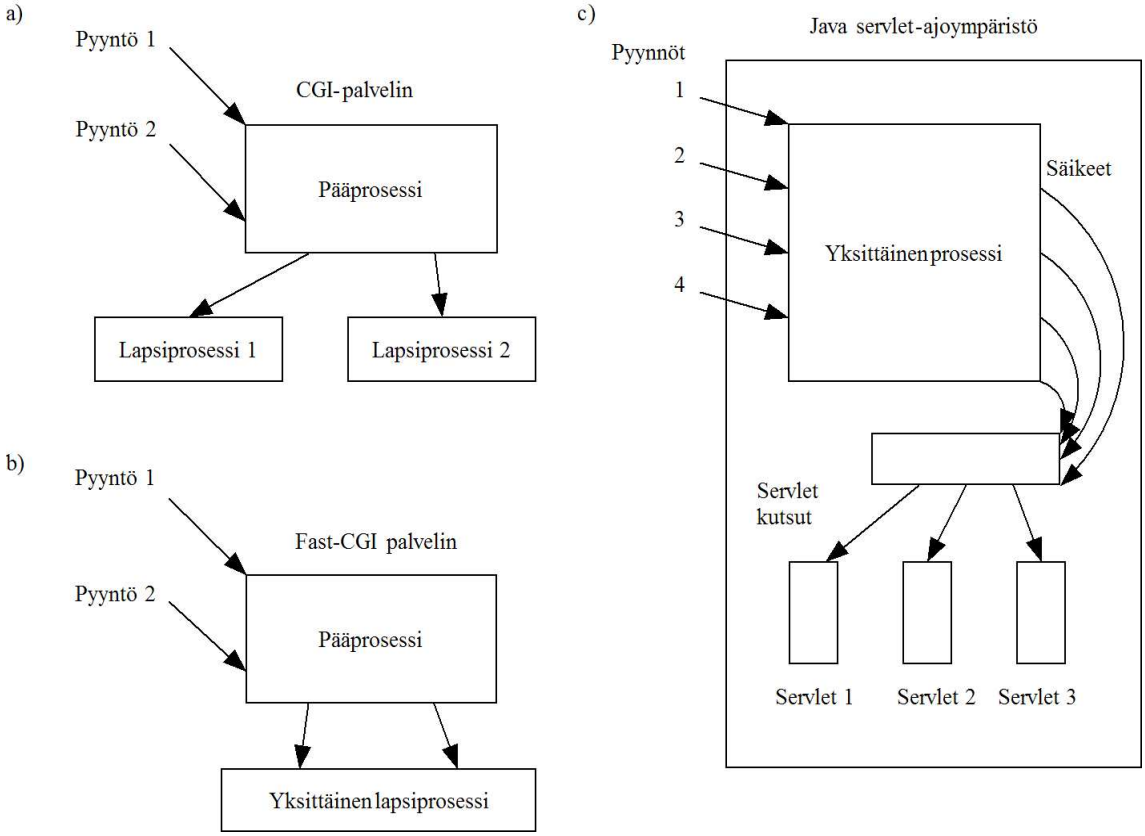
Servletit ovat Java-kielellä kirjoitettuja ja Java-tavukoodiksi käännettyjä moduuleja. Servlettien avulla voidaan laajentaa www-palvelinten toimintaa CGI-ohjelmien tapaan. Servlettejä kutsutaan myös palvelinsovelmiksi. Servletti käsittelee selainohjelman lähettämän pyynnön ja tekee tarvittavat toimenpiteet. Tämän jälkeen servlet tulostaa vastauksen, joka ohjataan pyynnön suorittaneelle selainohjelmalle. Servletit ovat palvelinkoneella sijaitsevia palveluja, joita käytetään selainohjelmien esittämien pyyntöjen käynnistäminä. [10]

Useat www-palvelinohjelmistot tukevat suoraan Java servlet -ohjelmointirajapintaa. Jos palvelinohjelmisto ei tue servlettejä suoraan, niin voidaan käyttää soveltuvaa ajoympäristöä servlettien suorittamiseen. Esimerkiksi maailman suosituimman www-palvelinohjelmiston Apachen kanssa voidaan käyttää Apache Tomcat servlet -ajoympäristöä. Tomcat toimii tarvittaessa myös itsenäisesti. Apachen tavoin Tomcat perustuu avoimeen lähdekoodiin ja on Java-servlettien virallinen referenssitoteutus. Servlet-ajoympäristöistä käytetään usein myös nimitystä sovelluspalvelin. [10] [17]

Servletit voivat käyttää Javan ohjelmointirajapinnan kaikkia ominaisuuksia ja komponentteja. Lisäksi servlettejä varten on määritetty oma Java Servlet API, joka sisältää verkkosovellusten ohjelmoinnissa tarvittavat ohjelmointirajapinnat. Laajojen ja standardoitujen ohjelmointirajapintojen lisäksi servletit ovat erittäin kilpailukykyisiä

suoritusnopeudessa verrattuna CGI-ohjelmiin. Servletit ladataan ajoympäristön muistiin, mikä nopeuttaa servlettien suorittamista. Pyynnön saapuessa suoritetaan servletin säie. Lisäksi tietokantayhteyksiä ei tarvitse varata jokaisella suorituskerralla, vaan tietokantayhteydet varataan ainoastaan ensimmäisellä kerralla ja niitä voidaan käyttää useiden pyyntöjen palvelemiseen. [10]

Kuvassa 10 esitetään kuinka servlettien suorittaminen eroaa CGI-ohjelmien suorittamisesta. Servletit suoritetaan säikeissä, kun taas CGI-ohjelmat suoritetaan omissa prosesseissa. FAST-CGI-tyyppisissä palvelimissa jokaiselle pyynnölle ei luoda omaa prosessia, vaan yksittäinen prosessi käsittelee kaikki pyynnöt. Käyttöjärjestelmän kannalta prosessien käynnistäminen ja prosessien vaihto ovat raskaita toimenpiteitä. Säikeiden luominen ja vuorottelu ovat selvästi nopeampia. Lisäksi säikeistys helpottaa kuorman jakamista useammalle prosessorille ja siten servlettejä voidaan tietyissä moniprosessoriympäristöissä suorittaa erittäin tehokkaasti. [23]



Kuva 10. Servlettien (c) ja CGI-ohjelmien suorittaminen (a, b). [23]

4.5.3 Ohjelmointirajapinta

Servlet-ohjelmointirajapinta määritellään paketissa javax.servlet ja rajapinnan määrittelemät metodit on esitetty kuvassa 11. Nämä metodit tarvitaan, jotta Java-luokasta tulisi servlet-luokka.

```
package javax.servlet;

public interface Servlet{

  public void destroy();
  public ServletConfig getServletConfig();
  public String getServletInfo();
  public void init(ServletConfig config) throws ServletException;
  public void service(ServletRequest request, ServletResponse
           response) throws ServletException, java.io.IOException;
}
```

Kuva 11. Servlet-rajapinnan määrittelemät metodit. [4]

Servlettien keskeinen osa on service-metodi, jota servlet-ajoympäristö kutsuu käsiteltäessä selaimelta vastaanotettuja pyyntöjä. Metodi ottaa parametreina selaimen pyynnöstä tietoa sisältävän request-olion ja selaimelle lähetettävän vastauksen sisältävän response-olion. Service-metodi on rajapinnan ainoa metodi, joka on pakko toteuttaa. Servlet alustetaan init-metodilla. Esimerkiksi tietokantayhteys voidaan luoda init-metodissa, jolloin se on käytettävissä koko servletin elinkaaren ajan. Kun servlet-ajoympäristö päättää poistaa servletin tai servlet lopetetaan hallintatyökaluilla, niin kutsutaan destroy-metodia. Metodissa voidaan suorittaa keskeneräisten prosessien poistaminen, kuten avoimien tiedostojen ja tietokantayhteyksien sulkeminen. Yleensä getServletInfo- ja getServletConfig-metodeja ei tarvitse ylikirjoittaa. Näistä ensimmäinen metodi palauttaa tietoa servletistä tekstimuodossa ja jälkimmäinen ServletConfig-olion, joka sisältää tietoa servletin konfiguraatiosta. [4]

Yleensä omat servlet-luokat luodaan perimällä joko HttpServlet- tai GenericServlet-luokka, jotka toteuttavat servlet-rajapinnan. HttpServlet-luokalla on lisämetodeja, jotka ovat erittäin hyödyllisiä ohjelmoijan kannalta. HttpServlet-luokka sisältää metodit erityyppisten HTTP-pyyntöjen käsittelyyn. Tavallisimmat GET- ja POST-pyynnöt voidaan käsitellä doGet- ja doPost-metodeilla. Metodit ovat samaa muotoa kuin service-metodi, jonka HttpServlet-luokan toteutus osaa ohjata selaimen pyynnön joko doGet- tai doPost-metodille. [4]

Servletin vastaus lähetetään selaimelle out.print- ja out.println-metodeilla. Ennen vastauksen lähettämistä servletin on lisäksi asetettava mediatyyppi, koska www-palvelinohjelmisto ei voi tietää servletin lähettämän vastauksen tyyppiä. Mediatyyppi kertoo selaimelle, kuinka lähetetttävä tiedosto on käsiteltävä. Esimerkiksi HTML-tiedoston mediatyyppi on text/html ja JPEG-kuvatiedoston image/jpeg. Mediatyyppi asetetaan ServletResponse-luokan setContentType-metodilla. Mediatyypin asettamisen jälkeen lähetetään vastaus. Merkkipohjainen data lähetetään response.getWriter-metodin palauttaman PrintWriter-olion avulla. Binääritiedoston, kuten esimerkiksi JPEG-kuvan, lähettämiseen käytetään response.getOutputStream-metodin palauttamaa ServletOutputStream-oliota. [4]

4.5.4 Java Server Pages

Servlettien ongelma on, että HTML-koodi joudutaan upottamaan Java-koodin sekaan. Java Server Pages (JSP) on laajennus servlet-teknologiaan. JSP-sivuilla Java-koodi sijoitetaan HTML-koodin sekaan. HTML-elementtejä käytetään normaalisti, mutta Java-koodi kirjoitetaan erityisten tagien (esimerkiksi <% ... %>) väliin. JSP-sivu käännetään ennen selaimelle esittämistä servletiksi. Näin ollen servletit ja JSP-sivut ovat sovelluspalvelimen kannalta sama asia. JSP-sivut suoritetaan servlettien tavoin säikeinä ja ohjelmoijan täytyy itse huolehtia säikeiden synkronisoinnista. [4] [10]

JSP-sivuina voidaan toteuttaa tavallisia HTML-sivuja, mutta JSP:n hyödyt tulevat esiin toteutettaessa verkkosovelluksia, joissa JSP-sivu toimii käyttöliittymänä ja sovelluslogiikka on toteutettu käyttöliittymästä erillisinä komponentteina. JSP-sivuja on usein helpompaa luoda ja ylläpitää kuin varsinaisia servlettejä. Myöskään Java-

ohjelmointikieltä ei tarvitse tuntea yhtä syvällisesti. JSP-sivuja luotaessa vältytään myös koodin kääntämiseltä, koska ajoympäristö huolehtii JSP-sivun kääntämisestä. Suurin JSP:n ongelma on, että usein HTML-koodin sekaan joudutaan kirjoittamaan paljon Java-koodia. Tällöin JSP-sivusta tulee helposti hankala ylläpidettävä. [4] [10]

4.5.5 Sessionhallinta

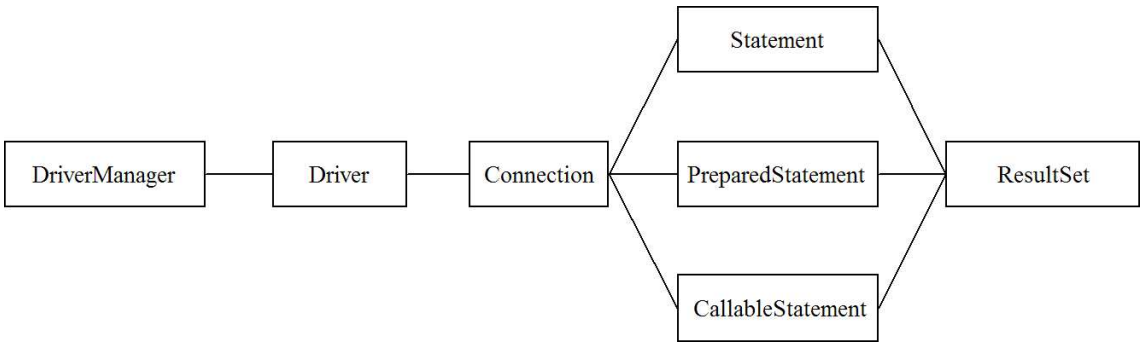
HTTP-protokolla on tilaton, minkä takia jokainen pyyntö on riippumaton muista pyynnöistä. Käytännössä on kuitenkin usein hyödyllistä säilyttää tietoa aikaisemmista pyynnöistä ja tätä kutsutaan sessionhallinnaksi. Esimerkiksi ei ole järkevää, että käyttäjä joutuu syöttämään käyttäjätunnuksen ja salasanan jokaiselle sivupyynnölle erikseen. Servletit ja JSP-sivut tukevat sessionhallintaa evästeiden (cookies), piilotettujen lomakekenttien tai URL-merkkijonoon lisätyn tunnisteen kautta. [10]

4.5.6 Tietokantayhteydet

Servletit ja JSP-sivut käyttävät tietokantayhteyksiin Javan normaalia JDBC-ohjelmointirajapintaa (Java Database Connectivity). JDBC on standardoitu ohjelmointirajapinta, jota lähes kaikki tiedonhallintajärjestelmät tukevat. JDBC:n käyttöä varten tarvitaan tietokantakohtainen ajuri. [4]

JDBC-ajurit jaetaan neljään tyyppiin eli Type1-, Type2-, Type3- ja Type4-ajureiksi. Tyypin 1 ja 2 ajurit sisältävät alustariippuvaista koodia eli ne eivät ole puhdasta Javaa. Tällaiset ajurit eivät yleensä toimi kaikissa laitteisto- ja käyttöjärjestelmäyhdistelmissä. Tyypin 1 ajurit käyttävät natiivia luokkakirjastoa ja ns. yleistä rajapintaa tietokantaan liittymiseksi. Tyypin 1 ajurit ovat yleensä hitaita, koska niissä tietokantaliitynnän toteuttamiseen tarvitaan monta välikerrosta. Tyypin 2 ajuri käsittelee tietokantakohtaista ajuria natiivin luokkakirjaston kautta. Natiivin luokkakirjaston ansiosta tyypin 2 ajurit ovat yleensä suhteellisen nopeita. Tyypin 3 ajurit ovat kokonaan Java-pohjaisia. Tämän tyypin ajurit kommunikoivat tietokannasta riippumattomalla protokollalla tietokantayhdystien (gateway) läpi. Tietokantayhdystie tekee tyypin 3 ajureista hitaita ja tehottomia. Tämän tyypin ajureita käytetäänkin lähinnä kiertämään Java-applettien

turvallisuusrajoitteita. Tyypin 4 ajurit ovat täysin Java-pohjaisia ja tämän tyypin ajurit kommunikoivat suoraan tietokannan kanssa. Nykyisten kehittyneiden JIT-kääntäjien ansiosta tyypin 4 ajurit ovat erittäin tehokkaita ja siten myös suosittuja. Ajurin tyypin valinta ei vaikuta ohjelmointirajapintaan. JDBC-ohjelmointirajapinnan ydinkomponentit on esitetty kuvassa 12. [4]



Kuva 12. JDBC-ohjelmointirajapinnan ydinkomponentit. [4]

Kuvan komponenteista ainoastaan DriverManager on konkreettinen Java-luokka ja muut ovat rajapintoja, joita ajuripaketit toteuttavat. DriverManager-luokka pitää lukua käytössä olevista JDBC-ajureista. Tietokantayhteys luodaan yleensä DriverManager-luokan kautta, mutta varsinaisesti tietokantayhteydestä on vastuussa Driver-rajapinta. Connection-rajapinta on JDBC-ohjelmointirajapinnan ydin. Rajapinnan metodit suorittavat tietokantatiedon hakuja, tietokantalauseiden luontia ja tietokantatransaktioiden käsittelyä. Statement-, PrepareStatement- ja CallableStatement-lauseilla suoritetaan tietokantakomentoja. Jokainen näistä toimii hiukan eri tavalla, mutta lopputuloksena on kaikilla tietokantakomento. ResultSet-rajapinnan avulla käsitellään SQL-kyselyn palauttamaa tietoa. Yleensä rajapintaa käytetään vain tiedon lukemiseen, mutta JDBC:n versiosta 2.0 lähtien kyselyn rivejä voidaan myös päivittää ja poistaa. [4]

Tietokantaohjelmoinnissa käytetään usein yhteysvarantoja. Yhteysvaranto on erityinen tietolähde, joka ylläpitää tietokantayhteyksien kokoelmaa. Usein sovelluksissa joudutaan käyttämään useita tietokantayhteyksiä yhtäaikaisesti. Tällöin voidaan laittaa tietokantayhteydet yhteysvarantoon, josta tietokantayhteys otetaan tarvittaessa käyttöön. JDBC-ohjelmointirajapinnassa ei ole valmista toteutusta yhteysvarannolle. Saatavissa

on kuitenkin useita yhteysvarantototeutuksia. Esimerkiksi dbConnectionBroker-luokkakirjasto toteuttaa yhteysvarannon käyttäen JDBC-ohjelmointirajapintaa. [4] [18]

JDBC-tietokantarajapinta helpottaa merkittävästi tietokantaohjelmointia ja eri tiedonhallintajärjestelmien yhteensovittamista. Eri tiedonhallintajärjestelmien ominaisuudet ovat kuitenkin erilaiset ja JDBC ei voi ratkaista kaikkia yhteensopivuusongelmia. Esimerkiksi tietotyyppien erilainen toteutustapa eri tiedonhallintajärjestelmissä johtaa yhteensopivuusongelmiin, jotka on huomioitava sovelluksen suunnittelussa.

4.5.7 Servlettien paketointi sovelluksiksi

Servlet-luokat, JSP-sivut ja muut sovelluksen tarvitsemat tiedostot, kuten kuvatiedostot, voidaan pakata helposti levitettäväksi sovellukseksi. Sovellus koostuu tavallisesti seuraavista tiedostotyypeistä:

- Servlet-luokat.
- JSP-sivut.
- Apuluokat.
- Staattiset dokumentit (HTML-tiedosto, kuva- ja äänitiedostot jne.).
- Asiakkaan appletit ja luokat.
- Metatieto, joka sitoo kaikki sovelluksen tiedostot yhteen. [18]

Sovelluksille on määritelty erityinen tiedostorakenne, jotta palvelinohjelmistot löytävät tiedostot aina samoista paikoista. Java Servlet Specification määrittelee erikoishakemiston WEB-INF, joka ei ole näy julkisessa hakemistopuussa. Servlet-luokat pystyvät kuitenkin käsittelemään hakemiston tietoja erityisten metodien kautta. Hakemisto WEB-INF sisältää määrittelyn mukaan seuraavat hakemistot ja tiedostot

- /WEB-INF/web.xml on XML-tiedosto, joka sisältää sovelluksen asetukset.
- /WEB-INF/classes sisältää servlet-luokat ja muut sovelluksen omat luokat.
- /WEB-INF/lib sisältää JAR-tiedostoiksi (Java Archive) paketoidut luokkakirjastot. [18]

Sovelluksen juurihakemistoon sijoitetaan HTML- ja JSP-sivut. Juurihakemistoon voidaan luoda hakemistoja erityyppisille tiedostoille. Sovellukset on mahdollista koota pakatuiksi WAR-tiedostoiksi (Web Archive), jotka useimmat servlet-sovelluspalvelimet osaavat purkaa ja ottaa käyttöön automaattisesti. WAR-tiedostot ovat samantyyppisiä kuin normaalit JAR-tiedostot (Java Archive), joita voidaan luoda Sunin ilmaisten sovelluskehitystyökalujen avulla.

Tiedosto web.xml on tärkeässä asemassa luotaessa Servlet-pohjaisia sovelluksia. Tiedosto sisältää mm. seuraavat määrittelyt:

- Sovelluksen asetukset, kuten kuinka kauan sovellus pidetään pyynnön jälkeen ajoympäristöön ladattuna.
- Servlettien ja JSP-sivujen määrittelyt.
- Sovelluksen oletussivu.
- Määriteltyjen servlettien ja JSP-sivujen vastaavuudet URL-osoitteissa.
- MIME-tyypit eli tiedostopäätteiden ja tiedostomuotojen vastaavuudet.
- Virhesivujen määrittelyt.
- Turvallisuusasetukset. [18]

Kuvassa 13 on esitetty yksinkertainen esimerkki web.xml-tiedoston määrittelystä.

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" ”http://java.sun.com/j2ee/dtds/web-
app_2_2.dtd">

<web-app>
    <display-name>Testisovellus</display-name>
    <servlet>
          <servlet-name>henkilolomake</servlet-name>
          <servlet-class>PersonForm</servlet-class>
    </servlet>
    <servlet-mapping>
          <servlet-name>henkilolomake</servlet-name>
          <url-pattern>/servlets/PersonForm</url-pattern>
    </servlet-mapping>
</web-app>
```

Kuva 13. Yksinkertaisen sovelluksen web.xml-tiedosto.

Määrittelytiedosto web.xml on XML-muodossa (Extensible Markup Language). XML on metakieli, joka kuvaa tiedon rakennetta ja sopii siten hyvin esimerkiksi asetusten esittämiseen. [20] Esimerkkikuvassa määritellään sovellus, jossa on käytössä yksi luokka (PersonForm).

4.5.8 Servlet-teknologian edut ja heikkoudet

Servletit ja JSP-sivut ovat CGI-ohjelmiin ja PHP-ohjelmiin verrattuna nopeampia kehittyneiden virtuaalikoneiden ja säikeistyksen ansiosta. Lisäksi servletit käsittelevät pyyntöjä erittäin nopeasti ensimmäisen pyynnön jälkeen, koska servletit ovat ajoympäristössä valmiiksi ladattuina. Tämä mahdollistaa myös tietokantayhteyksien säilyttämisen, mikä nopeuttaa tietokantasovelluksia. Tietokantayhteyden luominen vie paljon aikaa, jos tietokantayhteys joudutaan jokaista pyyntöä varten luomaan uudestaan. Servlettien suorittaminen ajoympäristössä parantaa myös tietoturvaa.

Servlettien ohjelmointirajapinta on monipuolinen ja lisäksi servletit voivat käyttää Javan muita monipuolisia luokkakirjastoja. Javan standardoidut ohjelmointirajapinnat ja luokkakirjastot ovat erittäin monipuoliset. Lisäksi on saatavilla runsaasti sovelluskehittäjien toteuttamia luokkakirjastoja ja ohjelmistokomponentteja. CGI-ohjelmien ongelma on standardoitujen ohjelmointirajapintojen puute. Tosin CGI-ohjelmoinnissa suositulle Perl-kielelle on olemassa kattavat kirjastot CGI-ohjelmointiin ja myös PHP tarjoaa monipuoliset rajapinnat www-sovellusten luontiin.

Servlettien monipuolisuus ja riippuvaisuus Java-teknologiasta on myös servlettien eräs heikkous. Servlet-ohjelmoinnin ja ajoympäristöjen opetteluun kuluu selvästi enemmän aikaa kuin yksinkertaisempien CGI-ohjelmointiin tarkoitettujen tulkattavien ohjelmointikielien tai PHP:n kaltaisten tekniikoiden opetteluun. Erityisesti tämä korostuu, jos Javasta ei ole ohjelmointikielenä aikaisempaa kokemusta. Myös olio-ohjelmoinnin perusteet on hallittava ennen servlet-ohjelmoinnin aloittamista. On mahdollista luoda servlet-sovelluksia ilman kunnollista oliomallia, mutta tällöin sovelluksista tulee vaikeasti ylläpidettäviä. Toisaalta kokeneille ohjelmoijille ja laajoja sovelluksia luotaessa servlettien riippuvaisuus Javan oliomallista on suuri etu, koska oliomallien ja monikerrosarkkitehtuurien avulla on mahdollista luoda helposti

ylläpidettäviä ja tehokkaita sovelluksia. Yksinkertaiset verkkosovellukset on kuitenkin yleensä helpompi toteuttaa esimerkiksi CGI-ohjelmien avulla varsinkin, jos ohjelmoijalla ei ole servlet-ohjelmoinnista aikaisempaa kokemusta.

Internet-tutkimusyritys Netcaftin [21] mukaan servlet- ja JSP-sivustoja on selvästi vähemmän verrattuna ASP- ja PHP-sivustojen lukumäärään. Toisaalta Netcraftin mukaan servlettien ja JSP-sivujen varaan rakennetut sivustot ovat yleensä laajempia ja monimutkaisempia muihin sivustoihin verrattuna. Kyseisten tekniikoiden käyttö on myös ollut voimakkaassa kasvussa.

JSP-sivut alentavat oppimiskynnystä esimerkiksi PHP-ohjelmoijille ja mahdollistavat lisäksi käyttöliittymän erottamisen servlettejä paremmin ohjelmakoodista ja sovelluslogiikasta. JSP-sivujen toteuttaminen on myös usein nopeampaa kuin servlettien. JSP-sivutkaan eivät mahdollista MVC-paradigman toteutusta ilman huolellista suunnittelua, koska Java-koodia joudutaan usein sijoittamaan HTML-koodin sekaan. Jos JSP-sivussa käytetään runsaasti Java-koodia, niin lopputulos on yleensä sekavampi kuin saman asian toteuttava servlet-luokka. Eräs mahdollisuus MVC-paradigman parempaan toteuttamiseen on avoimeen lähdekoodiin perustuva Struts-sovelluskehysalusta [22], joka käyttää JSP-teknologiaa. Struts-projekti on kuitenkin keskittynyt tarjoamaan työkaluja MVC-mallin mukaisten sovellusten toteuttamiseen.

MVC-paradigman toteuttamisen hankaluus servlettien ja JSP-sivujen avulla ei johdu ainoastaan kyseisistä teknologioista, vaan yleisesti ottaen selainpohjaiset käyttöliittymät eivät sovellu hyvin MVC-mallin mukaisten sovellusten toteuttamiseen. Käyttäjän syötteiden tarkistaminen ja toteuttaminen ei tapahdu reaaliaikaisesti kuten normaaleissa graafisissa käyttöliittymissä. Sovelluksen ja käyttäjän välisen vuorovaikutuksen toteuttaminen selainkäyttöliittymässä vaatii koko sivun päivittämistä. Sivun päivittäminen puolestaan vaatii usein tietojen päivittämistä tietokannasta. Verkon aiheuttamien viiveiden takia käyttäjän ja sovelluksen toimintalogiikan vuorovaikutus on olennaisesti hitaampaa ja ohjelmoinnillisesti hankalampaa toteuttaa kuin normaaleissa käyttöliittymissä. Toisaalta selainkäyttöliittymissä on monia hyviä puolia kuten alusta- ja paikkariippumattomuus. Selainkäyttöliittymien rajoitukset on hyvä tiedostaa sovelluksia suunniteltaessa. [31]

Ongelman ratkaisuksi on ehdotettu mm. selainpuolen ohjelmointikielien, Java applet-sovellusten ja DHTML-kuvauskielen käyttämistä selainkäyttöliittymissä. Jokaisessa ratkaisussa on kuitenkin omat ongelmat. Selainpuolen ohjelmointikielet, kuten JavaScript ovat rajoittuneita ja tietokantaliitynnät ovat hankalia toteuttaa. Applet-sovellukset rikkovat selainkäyttöliittymien yhtenäisyyden ja selaimissa on eritasoisia Java-virtuaalikoneita, mikä aiheuttaa yhteensopivuusongelmia. DHTML-kuvauskieli on osa HTML 4.0 -standardia, mutta varsinkin edistyneiden ominaisuuksien toteutus vaihtelee selainohjelmissa aiheuttaen yhteensopivuusongelmia. Toisistaan poikkeavien menetelmien käyttäminen rikkoo helposti sovelluksen yhtenäisyyden ja hankaloittaa ylläpitoa. [31]

5 Ilmatieteen laitoksen tuotantojärjestelmä

5.1 Ilmatieteen laitos

Ilmatieteen laitos on liikenne- ja viestintäministeriön alaisuudessa toimiva virasto. Ilmatieteen laitoksen tehtävä on tuottaa tietoa ilmakehän ominaisuuksista ja ilmiöistä yleisen turvallisuuden edistämiseksi sekä elinkeinoelämän ja yleisön tarpeita varten. Tätä varten suoritetaan havaintoja ilmakehän fysikaalisesta tilasta, kemiallisesta koostumuksesta ja sähkömagneettisista ilmiöistä. Ilmatieteen laitos on myös tutkimuslaitos, jonka tutkimusaloihin kuuluvat meteorologia, ilmasto, ilmanlaatu, avaruusfysiikka, kaukokartoitus ja geomagnetismi. Ilmatieteen laitos suorittaa lisäksi asiantuntijapalveluihin erikoistunutta liiketoimintaa Suomessa ja ulkomailla. Ilmatieteen laitos on perustettu vuonna 1838 ja nykyinen nimi on annettu vuonna 1969. [24]

Ilmatieteen laitos, kuten myös muut kansalliset ilmatieteen laitokset muissa maissa, tuottaa säädataa, -tuotteita ja -palveluita. Kansallisten ilmatieteen laitosten lisäksi useissa maissa toimii yksityisiä sääalan yrityksiä. Foreca on suomalainen yksityinen sääpalveluyritys, joka tuottaa sääpalveluita kotimaisille ja kansainvälisille markkinoille. Yksityiset sääpalveluyritykset tuottavat yleensä vain lopputuotteita. Ilmatieteen laitos tuottaa lisäksi säähavaintodataa, johon kuuluvat mm. pintasäähavainnot, sääluotaushavainnot, säätutkamittaukset, lentokoneluotaukset, satelliittihavainnot ja ilmanlaadun havainnot. Säädata muodostuu säähavaintodatasta ja tietokonemalleilla laskettavista ilmakehän meteorologisten tekijöiden analyyseista ja ennusteista. Säätuote on säädatasta tuotettu ilmakehän fysikaalista tilaa tai sen muutosta kuvaava teksti tai graafinen esitys. Sääennuste on ilmakehän fysikaalisen tilan muutoksen kuvaus, jonka tekemiseen käytetään yleensä sekä säädataa että säätuotteita. Sääpalvelulla tarkoitetaan asiakkaan tarpeisiin tuotettavaa lisäarvopalvelua, jossa hyödynnetään säädataa ja säätuotteita. [25]

Suurelle yleisölle näkyvin osa Ilmatieteen laitoksen toimintaa ovat sääennusteet, joita julkaistaan mm. Internetissä, sanomalehdissä, radiossa ja televisiossa. Ilmatieteen laitoksen asiakkaisiin kuuluvat mm. Yle ja Nelonen sekä lukuisat sanomalehdet, kuten

Helsingin Sanomat, Ilta-Sanomat, Etelä-Saimaa ja Karjalainen. Sähköisten medioiden kehittyminen vaikuttaa myös sääennusteiden tuotantoon, koska sääennusteet joudutaan sovittamaan erilaisille päätelaitteille kuten älykännyköille ja digitaaliselle televisiolle. Lisäksi Ilmatieteen laitos tuottaa mm. lentosääpalvelun Ilmailulaitokselle sekä ennusteita puolustusvoimille ja muihin viranomaistarkoituksiin.

Seuraavissa kappaleissa kuvataan tarkemmin meteorologisen tiedon jalostaminen ja tuotantojärjestelmän toiminta Ilmatieteen laitoksella.

5.2 Ilmatieteen laitoksen tuotantojärjestelmä

5.2.1 Yleistä tuotantojärjestelmästä

Säähavaintojen ja sääennusteiden historia on pitkä. Ensimmäinen kansainvälinen meteorologinen havaintoverkko perustettiin vuonna 1654. Havaintoverkkoon kuului seitsemän Pohjois-Italian asemaa ja neljä muuta asemaa Euroopasta. Ensimmäinen kansainvälinen meteorologian konferenssi järjestettiin vuonna 1853. Nykyinen WMO (World Meteorological Organization) perustettiin vuonna 1950 ja sitä ennen oli vuodesta 1873 saakka toiminut IMO (International Metorological Organization). Säähavaintojen nopean levittämisen mahdollisti Samuel Morsen vuonna 1832 keksimä lennätin, jota käytettiin ensimmäisen kerran säätietojen välittämiseen 1840-luvulla. Lewis Fry Richardson kehitti ensimmäisen matemaattisen mallin ja menetelmän numeeriselle sääennustukselle vuonna 1922. Tuolloin numeeristen sääennusteiden hyödyntäminen ei ollut käytännössä mahdollista, koska ennusteiden laskenta ihmisten avulla kesti monta kuukautta. Ensimmäiset tietokoneet syntyivät 1940-luvulla sotatarkoituksiin. Ensimmäiset ulkopuoliset sovellukset olivat meteorologisia ja vuonna 1950 laskettiin ensimmäinen sääennuste tietokoneella. 1960-luvulla sääsatelliitit alkoivat välittämään pilvikuvia. [24]

Meteorologisten tuotantojärjestelmien automatisointi on lisääntynyt jatkuvasti tietokoneiden laskenta-, tiedonsiirto- ja tallennuskapasiteetin kasvaessa. Entistä tarkempia sääennusteita voidaan tuottaa koko ajan nopeammin hyödyntämällä

supertietokoneiden laskentakapasiteettia. Automatisointi on lisääntynyt myös havaintojärjestelmissä. Esimerkiksi manuaalisia säähavaintoja on korvattu automaattihavainnoilla ja uutta tekniikkaa, kuten säätutkia, on otettu käyttöön lisääntyvässä määrin.

Liitteessä 1 on esitetty kuva meteorologisen tiedon jalostamisesta Ilmatieteen laitoksella. Kuva havainnollistaa ketjua, joka alkaa säähavaintojen teosta ja päättyy asiakastuotteisiin.

Säähavainnoista muodostettavia sääsanomia on useita erilaisia. Esimerkkeinä sääsanomista voidaan mainita SYNOP (maanpintasanoma), METAR (lentokenttähavaintosanoma) ja TEMP (radioluotaussanoma). Sääsanomat välitetään eteenpäin käyttäen WMO:n GTS-verkkoa (Global Telecommunication System). Perinteisten sääsanomien lisäksi tutkista ja satelliiteista saatava data on lähtökohtana myöhemmälle prosessoinnille. Seuraavassa vaiheessa datalle suoritetaan perusprosessointia, johon kuuluu mm. laadunvalvonta. Säähavainnosta muodostetaan erityyppisiä tiedostoja, jotka tallennetaan tietokantoihin ja muihin tietovarastoihin. Esimerkiksi Ilmatieteen laitoksen ilmastotietokanta sisältää kaikki Suomessa tehdyt säähavainnot vuodesta 1881 lähtien. Seuraavaksi suoritetaan ennustemallien tuottaminen sekä havaintojen ja numeeristen tuotteiden automaattinen perusjalostus. Meteorologit tulkitsevat ja korjaavat automaattisesti laskettuja säämalleja. Periaatteessa asiakastuotteet olisi mahdollista tuottaa nykyistä automaattisemmin vähentämällä meteorologien kontrollia säämalleihin. Säämallit eivät kuitenkaan ole vielä tarpeeksi kehittyneitä ja siksi täysin automaattisesti tuotettujen sääennusteiden laatu ei ole riittävän hyvä. Osa säätuotteista tuotetaan kuitenkin suoraan säämallien perusteella. Numeeriset sääennustusmallit, meteorologien tekemät sääennusteet ja muualta maailmasta saadut säähavainnot tallennetaan reaaliaikatietokantaan. Reaaliaikatietokanta toimii myös automaattisen tuotannon tietovarastona, eli reaaliaikatietokannan sisältämän tiedon pohjalta syntyvät automaattisesti tuotettavat asiakastuotteet.

Säätuotteita on automaattisessa tuotannossa tuhansia. Säätuotteet välitetään asiakkaille pääasiassa standardeja tiedonsiirtomenetelmiä, kuten HTTP-, FTP- (File Transfer Protocol) ja SCP-protokollia (Secury Copy Protocol), käyttäen.

5.2.2 Palvelinympäristö

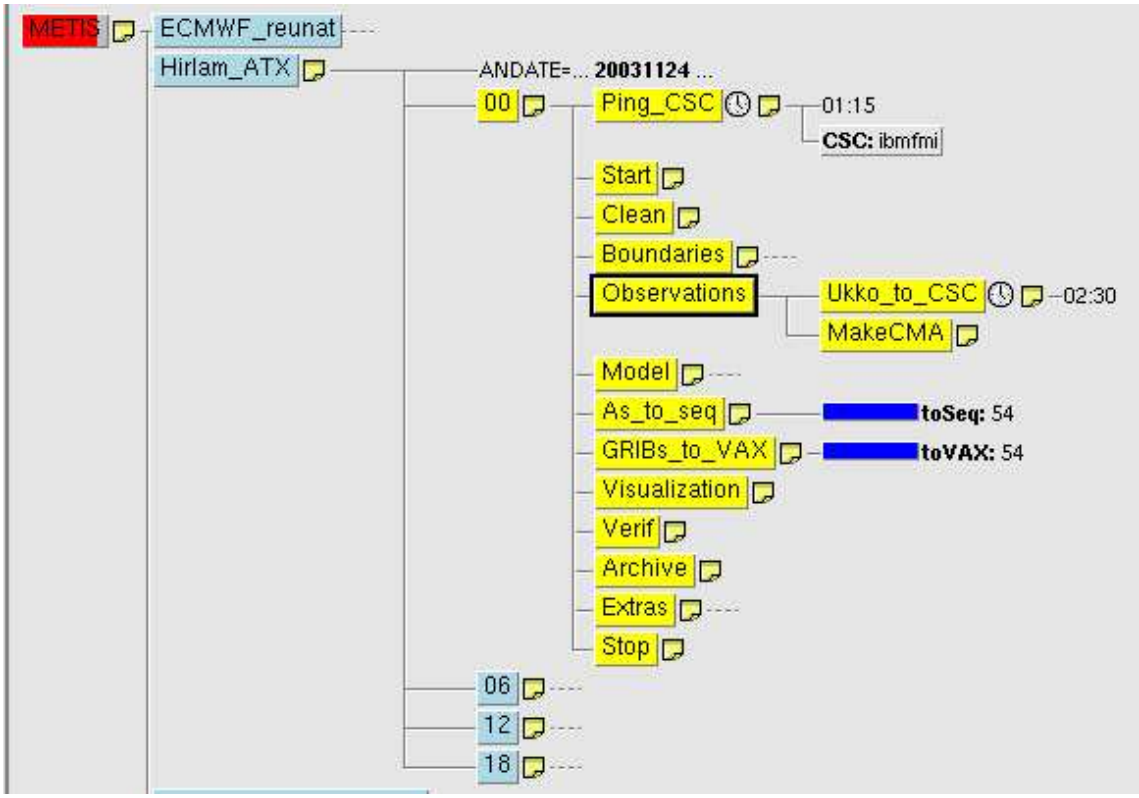
Ilmatieteen laitoksen palvelinympäristö koostuu useista eri laitealustoista ja käyttöjärjestelmistä. Käytössä on VAX/VMS-, Unix-, Linux- ja Windows-palvelimia. Unix-palvelimet ovat pääasiassa SGI Irix ja HP Tru64 Unix -pohjaisia. Omien palvelimien lisäksi Ilmatieteen laitos käyttää Tieteen tietotekniikan keskuksen eli CSC:n supertietokonekapasiteettia.

Tällä hetkellä paljon laskentatehoa vaativat säämallit lasketaan CSC:n tehokkaimmalla IBM eServer Cluster 1600 -supertietokoneella. IBM eServer Cluster 1600 koostuu 512:sta 1.1 GHz Power 4 -prosessorista, joiden teoreettinen laskentateho on 2,2 Tflop/s (Floating Point Operations per Second). CSC:n IBM eServer Cluster 1600 oli marraskuussa 2003 maailman 91. tehokkain supertietokone. [26] [27]

Tuotantoympäristön kannalta keskeisimmät palvelimet koostuvat SGI- ja Windows-palvelimista. SGI-palvelimilla ajetaan SMS-valvontaohjelmaa (Supervisor Monitor Scheduler) ja Windows-palvelimilla toimii tuotannon kannalta keskeinen oliohakurajapinta. Tru64 Unix -käyttöjärjestelmää käytetään lähinnä tietokanta- ja levypalvelimissa. Linuxin käyttö on lisääntynyt viime vuosina ja esimerkiksi satelliitti- ja tutkatuotantoa suoritetaan 8 prosessorin OpenMosix-tyyppisessä Linux-klusterissa. Raskas laskenta suoritetaan pääasiassa Linux- ja Unix-palvelimilla. Käytössä on yhä myös VAX/VMS-palvelimia, jotka ovat kuitenkin poistumassa lähivuosina.

5.2.3 SMS-valvontaohjelma

Ilmatieteen laitoksella käytetään tuotantoajojen käynnistämiseen ja valvontaan Euroopan keskipitkien sääennusteiden keskuksen ECMWF:n (European Centre for Medium-Range Weather Forecasts) kehittämää SMS-valvontaohjelmaa. SMS-valvontaohjelmaa voidaan käyttää eri laitealustoilla suoritettavien ajojen käynnistämiseen, seuraamiseen ja ohjaamiseen. SMS:n hallintaan voidaan käyttää CDP (Command Display Program) ja XCDP (X Command Display Program) ohjelmia, joista CDP on tekstipohjainen komentoriviohjelma ja XCDP (kuva 14) graafinen X11-ikkunointia käyttävä ohjelma. SMS on palvelinohjelma, joka palvelee asiakasohjelmia. CDP- ja XCDP-ohjelmat sekä suoritettavat ajot ovat SMS:n kannalta asiakasohjelmia. SMS toimii lukuisissa eri Unix-versioissa ja vanhemmat versiot tukevat myös VMS-käyttöjärjestelmiä. [28] [29]



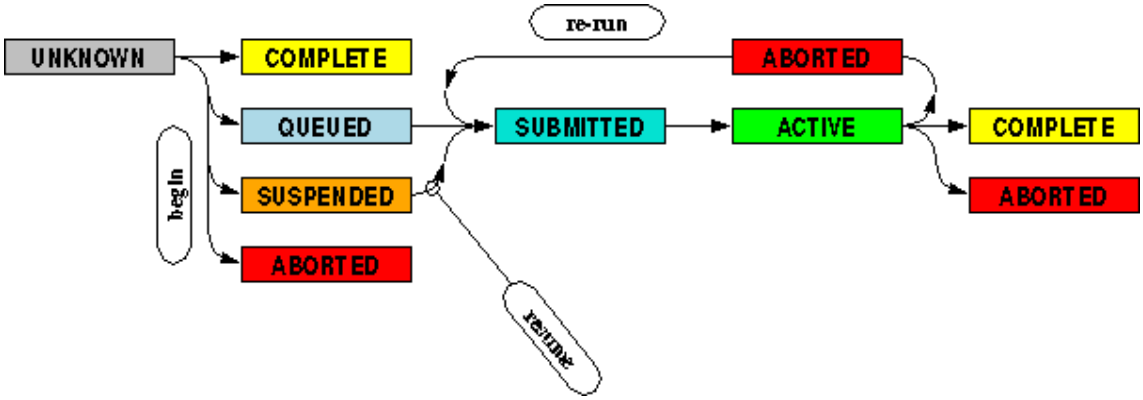
Kuva 14. Kuvaruutukaappaus osasta XCDP-ohjelman näyttöä.

SMS-ohjelma mahdollistaa ajojen suorittamisen hallitusti eri laitealustoilla. Jos ajo keskeytyy laite- tai ohjelmistovian seurauksena, niin ajo voidaan käynnistää uudestaan.

SMS käynnistää ajoja (tasks) ja vastaanottaa ajoilta viestejä, kun ajojen tila (status) vaihtuu. Ajon tilan vaihtuessa SMS osaa käynnistää tarvittaessa muita ajoja. Ajon käynnistyminen voi olla riippuvainen muista ajoista tai ajo voidaan määrätä käynnistymään esimerkiksi kellonajan perusteella. Esimerkiksi sääennustemallin käynnistyminen voi olla riippuvainen säähavaintojen vastaanottamisesta. [28]

SMS hallitsee sarjoja (suite), jotka koostuvat perheistä (family) ja perheet koostuvat ajoista (task). Yleistä termiä solmu (node) käytetään viittaamaan kaikkiin edellisiin. Solmut näkyvät XCDP:n näytöllä (kuva 14) puumaisina rakenteina. Määrittelyyn käytetään erillisiä tiedostoja, jotka voivat sisältää tavallisista ohjelmointikielistä tuttuja kontrollirakenteita ja muuttujia. [28]

Jokaisella SMS-ajolla on tila (status). Ajon tila voi olla tuntematon (unknown), jonossa (queued), valmis (completed), keskeytetty (aborted), pysäytetty (suspended), lähetetty (submitted) tai aktiivinen (active). Mahdolliset ajojen tilojen muutokset eri tiloista on esitetty kuvassa 15. XCDP:n näytöllä (kuva 14) ajojen tilat esitetään väreillä. Solmun tila määräytyy solmuun kuuluvien ajojen tilojen perusteella. [28]



Kuva 15. Mahdolliset SMS-ajojen tilojen muutokset. [28]

Muita SMS:n ja XCDP:n ominaisuuksia ovat mm. riippuvaisuuksien esittäminen graafisesti ja suoritettujen ajojen esittäminen graafisesti aikajanana. Ajojen riippuvaisuudet voivat muodostua erittäin monimutkaisiksi, jos ajojen lukumäärä on suuri. Ajojen riippuvuudet voivat olla peräkkäisiä tai rinnakkaisia. Rinnakkaisissa riippuvaisuuksissa ajoa ei voida suorittaa ennen kuin joukko ajoja on valmistunut. [29]

Ilmatieteen laitoksen tuotantojärjestelmässä ajojen käynnistämiseen käytetään SMS:n lisäksi jonkin verran myös muita tapoja, kuten Unix-käyttöjärjestelmissä perinteisesti ohjelmien ajastettuun suorittamiseen käytettyä Cron-ohjelmaa. SMS on ensisijainen tapa käynnistää tuotantoajoja. SMS soveltuu kuitenkin huonosti esimerkiksi tiheästi suoritettavien ohjelmien käynnistämiseen.

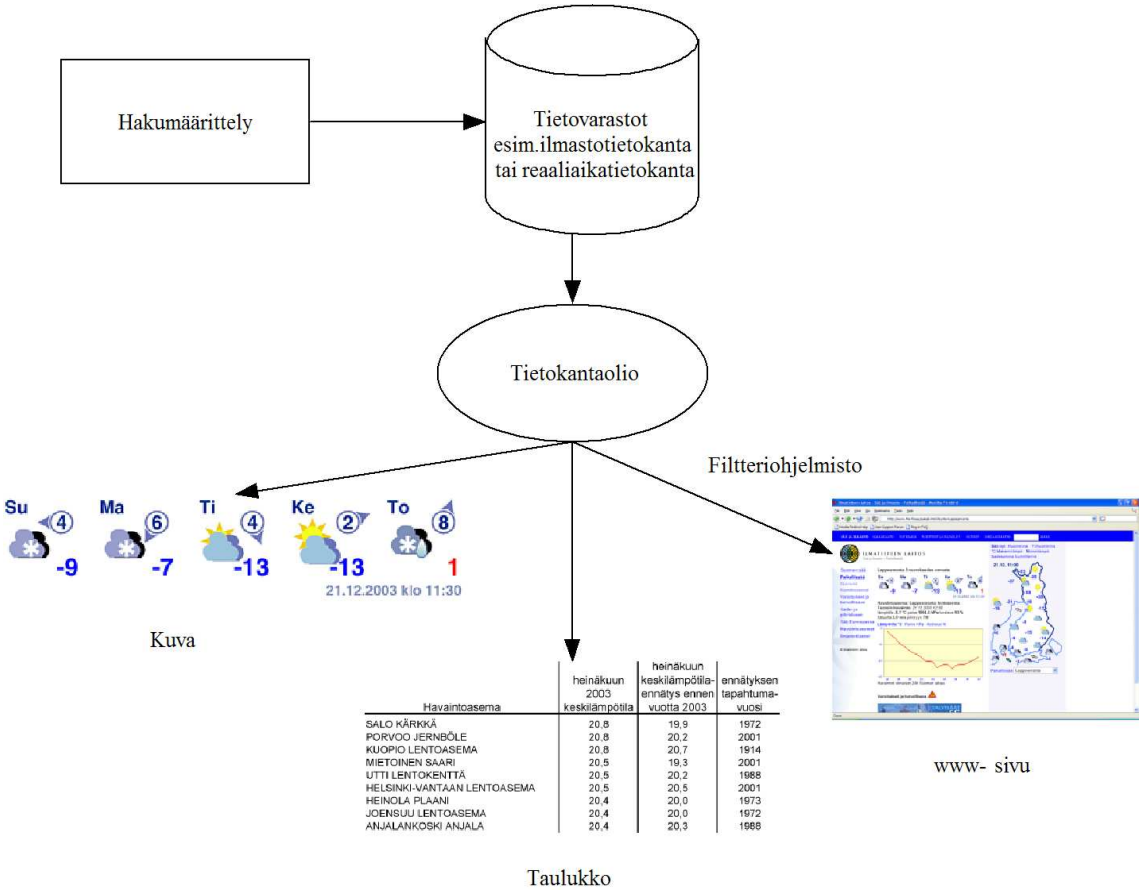
Ilmatieteen laitoksella SMS-palvelinohjelmaa suoritetaan SGI-palvelimissa. SMS-ohjelman avulla hallitaan reaaliaikaisesti eri laitealustoilla käynnistettävät tuotantoajot. Tuotantojärjestelmän täytyy toimia luotettavasti katkoitta, joten sitä valvotaan 24 tuntia vuorokaudessa. SMS-valvontaohjelma on välttämätön työkalu tuotantojärjestelmän valvonnan kannalta.

Ilmatieteen laitoksella SMS käyttää UTC-aikaa. UTC (Universal Time Coordinated) on virallinen standardiaika. UTC-aika on riippumaton aikavyöhykkeistä. UTC-aika lähes sama kuin aikaisemmin käytetty GMT-aika (Greenwich Mean Time). UTC on kuitenkin määritelty tarkemmin kuin GMT-aika, joka perustui aurinkoaikaan. UTC-aika on määritelty cesium-atomin värähdysten perusteella. Maapallon pyörimisliike hidastuu jatkuvasti ja sen myötä vuorokausi pitenee. UTC-ajan määrittelyssä tämä on huomioitu käyttämällä karkaussekunteja (leap seconds). Tämä tarkoittaa, että UTC-aikaan voidaan tarvittaessa lisätä tai vähentää sekunteja. UTC-ajan käyttäminen meteorologisessa tuotantoympäristössä on luonnollista, koska esimerkiksi sääsanomat tehdään kaikkialla maailmassa samaan UTC-aikaan. [32]

5.2.4 Oliohakurajapinta

Keskeinen osa Ilmatieteen laitoksen tuotantojärjestelmää on oliohakurajapinta. Oliohakurajapinta tarjoaa yhtenäisen sekä tiedon tallennuspaikasta ja esitysmuodosta riippumattoman tavan hakea tietoa lopputuotteiden tekoa varten. Hakumäärittelyjä varten on luotu erityinen par-kieli, jonka avulla voidaan määritellä filtteriohjelmisto, aikarajat, tiedon tuottaja, parametrit ja haun alue. Par-tiedoston perusteella haetaan tarvittava data tietokannoista ja muista tietovarastoista ja luodaan dataa kuvaava tietokantaolio. Tietokantaoliota käsitellään erityisellä filtteriohjelmistolla, joka luo halutun lopputuotteen. Lopputuote voi olla esimerkiksi kuva, taulukko, tekstitiedosto tai

www-sivu. Oliohakurajapinta on alun perin kehitetty Windows-ympäristöön, mutta nykyisin se toimii myös Linux-ympäristössä. SMS-ohjelmaa käytetään käynnistämään oliohakurajapinnan mukaiset haut. Oliohakurajapinnan toimintaperiaate on esitetty kuvassa 16.



Havaintoasema	heinäkuun 2003 keskilämpötila	heinäkuun keskilämpötila-ennätys ennen vuotta 2003	ennätyksen tapahtuma-vuosi
SALO KÄRKKÄ	20,8	19,9	1972
PORVOO JERNBÖLE	20,8	20,2	2001
KUOPIO LENTOASEMA	20,8	20,7	1914
MIETOINEN SAARI	20,5	19,3	2001
UTTI LENTOKENTTÄ	20,5	20,2	1988
HELSINKI-VANTAAN LENTOASEMA	20,5	20,5	2001
HEINOLA PLAANI	20,4	20,0	1973
JOENSUU LENTOASEMA	20,4	20,0	1972
ANJALANKOSKI ANJALA	20,4	20,3	1988

Kuva 16. Ilmatieteen laitoksen tuotantojärjestelmän oliohakurajapinnan toiminta.

6 Sovellus tuotantojärjestelmän seurantaan

6.1 Tarve tuotantojärjestelmän seurantaan

Tässä diplomityössä toteutettiin sovellus Ilmatieteen laitoksen tuotantojärjestelmän seurantaan. Tuotantojärjestelmän toimintaa on aikaisemmin seurattu pääasiassa SMS-valvontaohjelman avulla. SMS on reaaliaikainen järjestelmä ja sen tarjoamat mahdollisuudet tuotantoajojen tilastolliseen seurantaan ovat rajoittuneet. SMS tallentaa ajojen suoritusajoista ja tiloista tiedot tekstimuotoiseen lokitiedostoon. XCDP osaa esittää lokitiedoston perusteella aikajanan ajojen suoritusajoista. Lokitiedostoihin kertyy kuitenkin paljon tietoa ja muutaman päivän lokitiedosto on satojen megatavujen kokoinen. Siksi aikajanojen luominen lokitiedostojen perusteella on hidasta. Lokitiedostoja on myös mahdollista tutkia tekstityökaluilla, mutta se on myös hidas ja hankala tapa.

SMS ei myöskään käynnistä kaikkia tuotantoajoja. Näin ollen lokitiedostoon ei kerry tietoja kaikista tuotantoajoista. Tilastotietoja ei myöskään saada Ilmatieteen laitoksen ulkopuolelle menevistä tuotantoketjuista. Joissain tilanteissa olisi hyödyllistä saada tietoa milloin asiakas on vastaanottanut tietyn tuotteen. Aikaisemmin on kehitetty työkaluja yksittäisten tuotantoketjujen, kuten lentosääsanomien seuraamiseen. Erikoistuneilla työkaluilla voidaan saada yksityiskohtaista tietoa tietyistä tuotantoajoista, mutta ne eivät ole kattava ratkaisu kaikkien tuotantoajojen tilastolliseen seurantaan.

Tuotannon läpimenoaikojen seuraaminen on tärkeää, koska se mahdollistaa esimerkiksi tuotannon ongelmakohtien löytämisen ja tuotantoketjujen muutosten vaikutusten selvittämisen. Tuotannon läpimenoaikoja pyritään luonnollisesti jatkuvasti parantamaan. Esimerkiksi laitekannan uusimisen yhteydessä on hyödyllistä saada tietoa tuotantoaikojen parannuksista. Läpimenoaikojen tilastointi antaa myös mahdollisuuden sovelluskehittäjille seurata sovellustensa suoritusaikoja ja ohjelmakoodin muutosten vaikutusta suorituskykyyn.

6.2 Sovellukselle asetetut vaatimukset

Tuotantojärjestelmän seurantaan tarkoitetulle sovellukselle asetettiin seuraavat vaatimukset:

- Sovelluksen on toimittava Ilmatieteen laitoksen tuotantojärjestelmässä käytetyissä laitealustoissa ja käyttöjärjestelmissä.
- Sovellus mahdollistaa kaikkien tuotantoajojen suoritusaikojen kirjaamisen.
- Virhetilanteet on kirjattava tietokantaan.
- Sovellus hallitsee yksittäisten tuotantoajojen lisäksi laajemmat tuotantoketjut.
- Sovelluksen käyttöliittymä on selainpohjainen.
- Tuotantoajoihin- ja ketjuihin voidaan liittää vastuuhenkilöitä, asiakkaita ja kirjata muutoksia.
- Sovellus käyttää tuotantojärjestelmässä käytettyä Oracle-tietokantaa.
- Sovellus mahdollistaa suoritusaikojen hakemisen hakuehdoilla.
- Ajojen ja tuotantoketjujen suoritusajoista voidaan luoda erilaisia raportteja www-käyttöliittymän avulla.
- Sovellus soveltuu sekä tuotantojärjestelmän kehittäjien että tuotantojärjestelmää valvovien operaattoreiden työkaluksi.

Sovelluksen toimintaympäristö on tuotantojärjestelmä ja siksi oli selvää, että sovellus ei saa häiritä tuotantoajojen suorittamista ja sovelluksen on toimittava mahdollisimman automaattisesti ilman valvontaa.

6.3 Sovelluksen kuvaus ja tekniset ratkaisut

6.3.1 Sovelluksen osat

Alusta alkaen oli selvää, että ajojen, tuotantoketjujen, vastuuhenkilöiden, asiakkaiden ja muutosten määrittely kannattaa toteuttaa www-käyttöliittymän avulla. Samoin suoritusaikojen ja raporttien esittäminen olivat luonnollisesti osa selainkäyttöliittymää. Selvästi hankalampaa oli päättää, kuinka ajojen suoritusajat saadaan kirjatuksi tietokantaan.

Suurin osa tuotantoajoista käynnistetään SMS-valvontaohjelman avulla, mutta osa tuotantoajoista ei ole SMS-valvontaohjelman hallinnassa. Eräs tapa kirjata suoritusaikoja tietokantaan on käydä läpi sovellusten lokitiedostoja. Lokitiedostojen analysointi on yleinen menetelmä esimerkiksi www-palvelinten tilastojen keräämisessä. Tämän ratkaisun ongelma on, että sovellusten lokitiedostojen muoto vaihtelee, joten jokaiselle sovellukselle joudutaan kirjoittamaan oma ohjelma lokitiedoston läpikäymiseen. Ongelma voidaan osittain ratkaista muokkaamalla lokitiedostojen informaatiota yleisempään muotoon ja muuntamalla tiedot esimerkiksi XML-muotoon. Tällöin lokitiedoston tietojen käsittelyyn ja raporttien luomiseen voidaan käyttää yhtenäisiä sovelluksia. Esimerkiksi Lire-ohjelma [33] lokitiedostojen analysointiin ja raportointiin toimii tällä tavalla. Toinen lokitiedostojen käsittelyyn liittyvä ongelma on lokitiedostojen suuri koko, jolloin niiden parsiminen kestää kauan ja kuluttaa käyttöjärjestelmän resursseja.

Toinen mahdollisuus on lisätä ohjelmien lähdekoodeihin ja tulkattaviin ohjelmiin kutsut, jotka tallentavat tiedot suoritusajoista tietokantaan. Tämä ratkaisu on huomattavasti yleiskäyttöisempi ja kevyempi kuin lokitiedostoihin pohjautuva ratkaisu. Ongelmaksi muodostuu olemassa olevien ohjelmien lähdekoodien ja tulkattavien ohjelmien muokkaaminen. Tämä korostuu tuotantojärjestelmissä, joissa on runsaasti erilaisia järjestelmän toiminnan kannalta kriittisiä ohjelmia.

Tässä tapauksessa päätettiin toteuttaa molemmat vaihtoehdot eli ohjelma SMS-valvontaohjelman lokitiedoston parsimiseen ja ohjelma, joka mahdollistaa ajojen suoritusaikojen kirjaamisen tulkattavissa ohjelmissa. Molemmat ohjelmat voivat käyttää yhteisiä ohjelmointirajapintoja, jolloin niiden toteuttaminen on suhteellisen yksinkertaista. Yleiskäyttöistä ohjelmointirajapintaa voidaan käyttää myös myös muissa ohjelmissa suoritusaikojen kirjaamiseen.

Tietokoneiden kellopiirit eivät ole yleensä laadukkaita, minkä takia tietokoneiden kellojen pitäminen oikeassa ajassa edellyttää aikapalvelimien käyttämistä. NTP (Network Time Protocol) ja SNTP (Simple Network Time Protocol) ovat yleisesti käytettyjä protokollia aikapalvelimien kanssa kommunikoimiseen. NTP-protokolla käyttää sisäisesti nanosekunnin tarkkuutta ja lopullinen tarkkuus riippuu verkon

viiveistä. Yleensä virhe on vain muutamia millisekunteja. SNTP-protokolla on yksinkertaistettu versio NTP-protokollasta, jonka tarkkuus riittää kuitenkin hyvin asiakasohjelmissa käytettäviksi. [32]

Ilmatieteen laitoksen aikapalvelimet synkronisoidaan Mittatekniikan keskuksen Mikesin ylläpitämästä aikapalvelimesta, joka määrittelee Suomen virallisen ajan. Sovelluksen aikatiedot päätettiin hakea NTP- tai SNTP-protokollan avulla aikapalvelimilta, jotta suoritusajat olisivat mahdollisimman vertailukelpoisia. Aikatiedot tallennetaan tietokantaan UTC-ajan mukaan.

Sovellus koostuu siis viidestä osasta:

1. Relaatiotietokanta, johon sovelluksen tiedot tallennetaan.
2. Yhteisen ohjelmointirajapinnan toteuttavat kirjastot.
3. SMS-valvontaohjelman lokitiedostoa käsittelevä ohjelma.
4. Suoritusaikoja kirjaava ohjelma, joka voidaan liittää tuotantoajoja ohjaaviin tulkattaviin ohjelmiin.
5. Sovelluksen hallintaan käytettävä www-käyttöliittymä.

Sovelluksen nimeksi annettiin Tuomo, joka tulee sanoista tuotannon monitorointi. Sovelluksen tarkempi tekninen toteutus on kuvattu seuraavissa kappaleissa.

6.3.2 Tekniset ratkaisut

Sovellus päätettiin toteuttaa kokonaisuudessaan Java-ohjelmointikielellä. Sovelluksen www-käyttöliittymä toteutettiin Java servlettien ja JSP-sivujen avulla. Tähän ratkaisuun vaikuttivat Ilmatieteen laitoksen palvelinympäristön koostuminen useista laitealustoista ja käyttöjärjestelmistä. Javalla toteutettuja ohjelmia voidaan suorittaa virtuaalikoneen avulla lähes kaikissa palvelimissa. Ainoastaan lähinnä VAX/VMS-käyttöjärjestelmälle ei ole olemassa virtuaalikonetta. Sovelluksen kaikkien osien toteuttaminen samalla ohjelmointikielellä mahdollistaa myös yhtenäisten ohjelmointikirjastojen käyttämisen. Lisäksi Javan tietokantaominaisuudet ja ohjelmointikirjastot ovat erittäin monipuoliset. Ilmatieteen laitoksen tuotantojärjestelmää ollaan parhaillaan uudistamassa ja Java on valittu keskeiseksi tuotantojärjestelmän ohjelmointikieleksi. Myös tämän takia Java oli

luonnollinen valinta, koska organisaation sisällä on tärkeää käyttää yhtenäisiä ohjelmistokehitysmenetelmiä.

Sovelluksen pääasiallisena tietokantaympäristönä päätettiin käyttää Oraclea. Oraclea käytetään Ilmatieteen laitoksen tuotantojärjestelmän tietokantaympäristönä. Javan JDBC-tietokantarajapinta mahdollistaa yhtenäisen tavan käyttää eri tiedonhallintajärjestelmiä. Sovelluskehityksen helpottamiseksi sovellus tehtiin Oraclen lisäksi yhteensopivaksi MySql-tiedonhallintajärjestelmän kanssa. MySql on avoimeen lähdekoodin pohjautuva relaatiotietokantojen tiedonhallintajärjestelmä, joka voidaan siten vapaasti asentaa sovelluskehittäjien työasemille. Tämä helpottaa sovelluskehitystä, koska sovelluskehittäjä voi tarkkailla helposti sovelluksen toimintaa tiedonhallintajärjestelmän kannalta. Myös mahdolliset virhetilanteet, kuten tietokantayhteyksien vuotaminen, eivät häiritse tuotantokäytössä olevaa tiedonhallintajärjestelmää. Kahden tiedonhallintajärjestelmän ratkaisu pakotti myös sovelluksen kehittämisen mahdollisimman tietokantariippumattomaksi ja siten helposti eri tietokantaympäristöihin siirrettäväksi. Oracle ja MySql tarjoavat käyttöjärjestelmäriippumattomat JDBC-ajurit.

Sovellus käyttää JDBC-ajurien lisäksi muutamia ulkopuolisia Java-ohjelmointikirjastoja. Sovelluksen tietokantayhteysvaranto on toteutettu dbConnectionBroker-kirjaston avulla. [18] Kyseinen kirjasto mahdollistaa tarvittavien tietokantayhteyksien varaamisen etukäteen ja yhteyksien käytön dynaamisesti tarpeen vaatiessa. SNTP-liityntä aikapalvelimeen toteutettiin käyttämällä Michel Van den Berghin SNTP-toteutusta Javalle. [34] Servlet-pohjaisen www-käyttöliittymän graafisten raporttien toteuttamiseen käytettiin monipuolista JFreeChart-luokkakirjastoa. [35] JFreeChart mahdollistaa useiden erilaisten graafisten kuvaajien luomisen.

Servlet-ajoympäristönä käytettiin Tomcatin 4.1-sarjan versioita, jotka toteuttavat Java servlet 2.3 määrityksen. Sovellus kehitettiin Linux-ympäristössä käyttäen JBuilder 9 Personal Edition ja NetBeans IDE sovelluskehitysympäristöjä (Integrated Development Environment). IDE-sovelluskehitysympäristöt tarjoavat useita työkaluja, kuten luokkien metodien automaattisen täydennyksen, sovelluskehityksen nopeuttamiseen ja mahdollistavat ohjelmakoodin dokumentaation luomisen automaattisesti. UML-

mallinnuskielen mukainen oliomalli luotiin Poseidon For UML Community Edition ohjelmalla. MySql:n ja Oraclen kanssa käytettiin MySql Control Center ja Tora ohjelmia. Molemmat ovat graafisia avoimeen lähdekoodiin pohjautuvia tietokantatyökaluja, jotka helpottavat mm. relaatiotaulujen luomista merkittävästi.

6.3.3 Tietokannan kuvaus

Sovelluksen käyttämät tietokantataulut on esitetty liitteessä 2, missä on esitetty MySql:n relaatiotaulut. Oraclen taulut ovat samanlaiset lukuun ottamatta pieniä eroja tietotyypeissä.

Task-tauluun tallennetaan ajoihin liittyvät tiedot, kuten ajon nimi, suorituspaikka ja SMS-ohjelmaan liittyvät määrittelyt. Ajoista muodostuu tuotantoketjuja, jotka määritellään Chain- ja Chain_tasks-taulujen avulla. Yksittäisiin ajoihin liittyy tapahtumia, jotka tallennetaan Event-tauluun. Tapahtumiin liittyy mm. aloitus- ja lopetusaika sekä tieto päättyikö tapahtuma normaalisti vai virheeseen. Persons- ja Customers-taulut ovat ajojen ja tuotantoketjujen vastuuhenkilöiden sekä asiakkaiden tallentamista varten. Task_modifications- ja Chain_modifications-taulut ovat ajoihin ja tuotantoketjuihin liittyvien muutosten kirjaamista varten. Muut relaatiotaulut ovat linkkitauluja, joiden avulla puretaan n:n yhteydet taulujen välillä. Sovellusalueen käsitteille on määritelty numeeriset tunnisteet, joita käytetään relaatiotaulujen pää- ja vierasavaimina.

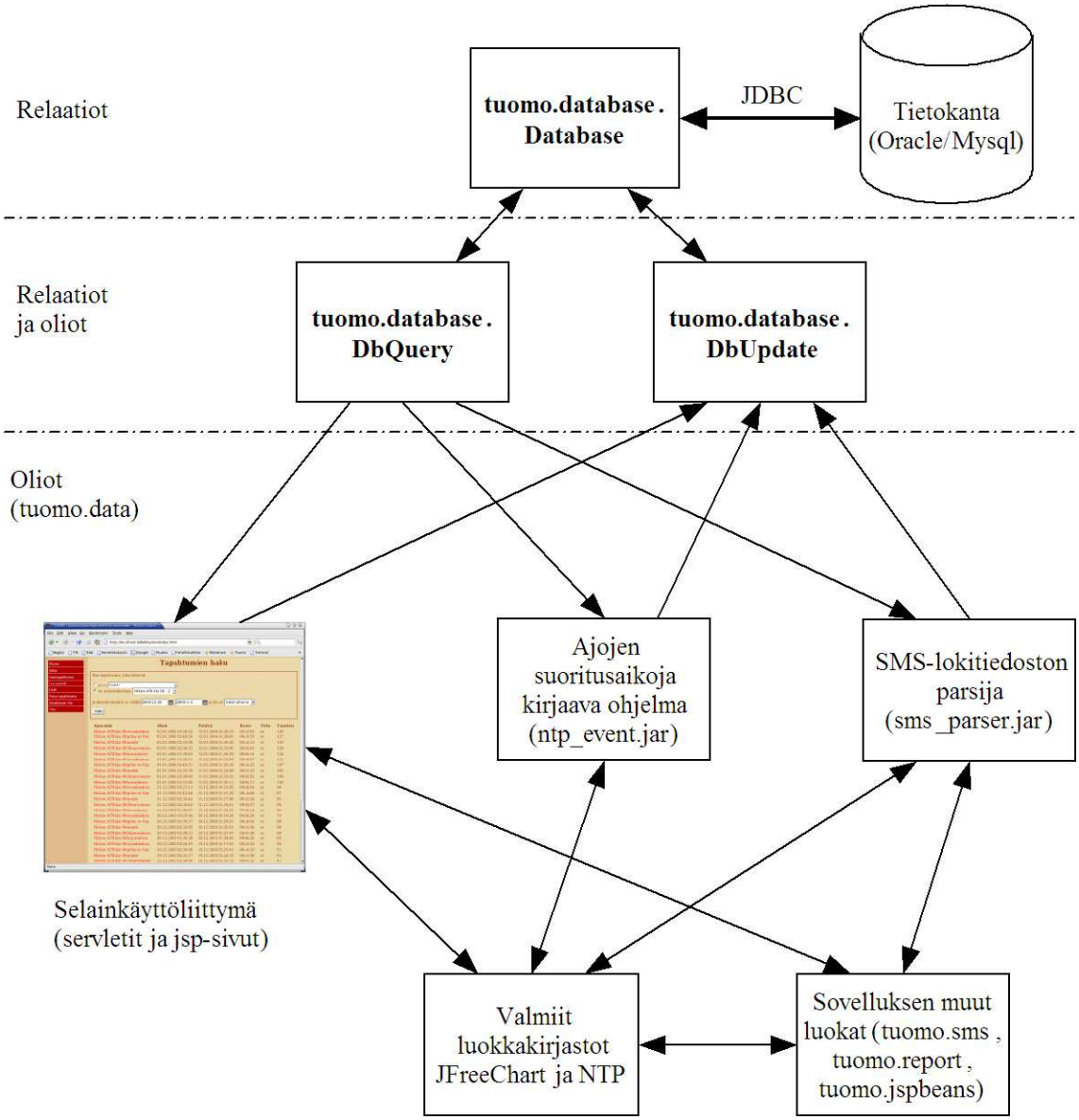
Relaatiotauluissa pyrittiin käyttämään SQL-standardin mukaisia tietotyyppejä, jotta yhteensopivuus sekä MySql:n että Oraclen kanssa säilyisi. Esimerkiksi MySql tukee totuusarvojen tallentamiseen boolean-tietotyyppiä, mutta Oraclessa ei ole vastaavaa tietotyyppiä. Siksi totuusarvot tallennettiin kokonaislukuina 0 (epätosi) tai 1 (tosi). MySql:n suuri puute on, että tietokannan eheyden varmistamiseksi ei voida määritellä vierasavaineheyttä. Oracle puolestaan tukee vierasavainten käyttämistä. MySql:n takia vierasavaineheyden ylläpitäminen jäi sovelluksen vastuulle. Vierasavaineheyden puuttumisen takia sovelluksen suorittama virheellinen SQL-lause voi johtaa tietokannan eheyden särkymiseen. Tietokannalle määriteltiin suunnitteluvaiheessa eheyssäännöt.

Oraclen ja MySql:n JDBC-ajurit eivät toteuta kaikkia JDBC-ohjelmointirajapinnan ominaisuuksia. Suurin puute havaittiin MySql:n JDBC-ajurissa, joka ei ota huomioon Javan GregorianCalendar-luokan päivämäärä ja aikatiedon sisältämää aikavyöhykeinformaatiota. Tämän takia aikatiedot jouduttiin JDBC-ohjelmointirajapinnan metodien sijaan päivittämään ja hakemaan SQL-lauseiden avulla. MySql ja Oraclen aikatiedon käsittely eroaa jonkun verran toisistaan, joten aikatietoa sisältävät SQL-lauseet jouduttiin toteuttamaan erikseen molemmille tiedonhallintajärjestelmille. Muissa tapauksissa JDBC-ohjelmointirajapinnan metodit olivat yhteensopivia molempien tiedonhallintajärjestelmien kanssa ja yhteensopivuuden säilyttäminen ei tuottanut ongelmia.

6.3.4 Oliomalli

Sovelluksen oliomalli mallinnettiin aluksi käyttäen UML-mallinnuskielen käyttötapauskaavioita ja luokkakaavioita. Luokkakaavioiden pohjana oli pitkälti relaatiomallin taulut, mikä johti datakeskeisen oliomallin luomiseen. Yleisesti ottaen luokkakaavioiden käyttäminen johtaa helposti datakeskeiseen oliomalliin. Tähän sovellukseen datakeskeinen oliomalli soveltui hyvin, koska sovellus ei ollut toimintalogiikan kannalta erityisen vaativa.

Sovelluksen luokat on jaettu Javan paketteihin (package), jotka ovat tuomo.data, tuomo.database, tuomo.report, tuomo.sms ja tuomo.jspbeans. Paketti tuomo.data sisältää luokat, jotka mallintavat sovelluksen käsitteitä, kuten ajoja, tuotantoketjuja ja tapahtumia. Paketti tuomo.database sisältää luokat, jotka vastaavat tietokantayhteyden ylläpidosta, tiedon päivittämisestä tietokantaan ja tietokantakyselyistä. Kyseinen paketti käsittelee myös muunnokset relaatioiden ja olioiden välillä. Paketti tuomo.report sisältää luokat, jotka toteuttavat raporttien luonnin. Paketit tuomo.sms ja tuomo.jspbeans sisältävät tällä hetkellä vain yksittäiset luokat, jotka hoitavat SMS-ohjelman lokitiedoston parsimisen ja JSP-sivujen tarvitsemien tietojen palauttamisen. Kuvassa 17 on esitetty pakettien ja sovelluksen osien väliset suhteet tarkemmin. Lisäksi kuva esittää kuinka relaatioiden ja olioiden käsittely on jakautunut sovelluksen osien välille. Sovelluksen loogiset kokonaisuudet on paketoitu JAR-paketeiksi.



Kuva 17. Sovelluksen osien ja luokkakirjastojen väliset suhteet.

Sovelluksen tuomo.database paketin DbQuery- ja DbUpdate-luokat hoitavat muunnokset relaatioiden ja olioiden välillä, jolloin muut sovellukset osat voivat käsitellä tietoa oliomuodossa. DbQuery-luokka käsittelee oliopyynnöt ja DbUpdate-luokka päivittää tietokannan tietoja. Välikerroksen käyttäminen osoittautui hyväksi ratkaisuksi, vaikka olioiden muuntaminen relaatioiksi ja päinvastoin vaatii suhteellisen paljon ohjelmakoodia. Kun tarvittavat metodit on toteutettu ja testattu, niin tietokannan käsittely on yksinkertaista. Sovellukset käsitteitä mallintavat luokat olivat suhteellisen yksinkertaisia, jolloin niiden muuntaminen relaatiotauluiksi ei ollut monimutkaista.

Olioiden ja relaatioiden välisten muunnosten toteuttaminen vei kuitenkin merkittävän osan sovelluskehitykseen kuluneesta ajasta. Erityisesti, jos luokilla olisi ollut monimutkaisempia perinnöllisyyssuhteita, niin valmiiden työkalujen käyttämistä muunnoksiin olisi kannattanut harkita.

Data-, toimintalogiikka- ja esityskerrokset eivät ole täysin selkeitä käytetyssä oliomallissa. Datakerros koostuu paketeista tuomo.data ja tuomo.database. Sen sijaan toimintalogiikka on osittain omissa paketeissa (tuomo.report ja tuomo.sms), mutta toimintalogiikkaa on myös selainkäyttöliittymän toteuttavissa servlet-luokissa. Ihanteellisessa tapauksessa esityskerros olisi kokonaan JSP-sivuissa. Tämä osoittautui kuitenkin hankalasti toteutettavaksi ja esityskerroksen eriyttäminen olisi vaatinut huomattavasti tarkempaa suunnittelua. Servlet-luokissa HTML-koodi sekoittuu Java-koodin sekaan ja JSP-sivuja on hankala toteuttaa ilman ohjelmakoodia eli pelkkien JSP-tagien avulla. Tämä hankaloittaa merkittävästi sovelluskerrosten eriyttämistä ja MVC-paradigman mukaisen käyttöliittymän toteuttamista. Parhaiten JSP-sivujen todettiin sopivan lomakkeita sisältäville sivuille, joissa lomakkeiden tieto voidaan hakea yksittäisten JSP-tagien avulla muokattavaksi. MVC-mallin toteuttaminen ei ole pelkästään Servlet- JSP-tekniikoiden ongelma. Kyseessä on selainkäyttöliittymien ominaispiirre, joka johtuu käyttäjän ja sovelluksen hitaasta vuorovaikutuksesta.

Sovellus voidaan luokitella kaksi- tai kolmitasoarkkitehtuurin mukaiseksi näkökulmasta riippuen. Ajojen suoritusaikoja kirjaava ohjelma ja SMS-lokitiedostoa käsittelevä ohjelma toteuttavat kolmitasoarkkitehtuurin puhtaasti eli pääohjelma huolehtii vain toimintalogiikan tarjoavien luokkien ohjaamisesta. Kuitenkin selainkäyttöliittymässä esityskerros ja toimintakerros ovat osittain päällekkäisiä. Selainkäyttöliittymä on suuri osa kokonaisuutta, joten tässä diplomityössä sovelluksen arkkitehtuuri luokitellaan kaksitasoiseksi. Kolmitasoarkkitehtuurin toteuttaminen olisi parantanut sovelluksen ylläpidettävyyttä, mutta toisaalta uusien menetelmien opetteluun ja suunnitteluun olisi kulunut huomattavasti enemmän aikaa. Toteutettu ratkaisu oli hyvä kompromissi ylläpidettävyyden ja toteutukseen kuluneen ajan suhteen.

6.3.5 Käyttöliittymä

Sovelluksen hallintaan tarkoitettu selainkäyttöliittymä on toteutettu Java servlettien ja JSP-sivujen avulla. Käyttöliittymä toimii useimmilla nykyaikaisilla graafisilla www-selaimilla.

Sovelluksen www-käyttöliittymän avulla voidaan tehdä seuraavat asiat:

- Lisätä, muokata ja poistaa ajoja, tuotantoketjuja, vastuuhenkilöitä ja asiakkaita.
- Selata ja tarkastella ajoja, tuotantoketjuja, vastuuhenkilöitä ja asiakkaita sekä näiden välisiä riippuvaisuuksia.
- Hakea ja poistaa ajojen ja tuotantoketjujen tapahtumia.
- Luoda teksti- ja graafisia raportteja ajojen ja tuotantoketjujen tapahtumista.
- Tarkastella sovelluksen tilaa, kuten tietokantatauluja ja tietokantayhteyksien lukumäärää.

Käyttöliittymän ulkoasu on määritelty CSS-tyylitiedostojen (Cascading Style Sheets) avulla. CSS-tyylitiedostot mahdollistavat yhtenäisen ulkoasun määrittelyn sovellukselle. Tässä tapauksessa käytettiin kahta tyylitiedostoa, joista toinen toteuttaa normaalin ulkoasun ja toinen pelkistetymmän mustavalkoisen ulkoasun sivujen tulostamista varten. Käyttäjä voi valita käyttöliittymän tyylin näistä kahdesta vaihtoehdosta. Liitteessä 3 on esitetty kuvaruutukaappaus sovelluksen käyttöliittymästä. Kuvaruutukaappaus on otettu sivusta, joka mahdollistaa tuotantoajojen tapahtumien hakemisen tietokannasta.

Myös SMS-lokia käsittelevä ohjelma ja ajojen suoritusaikoja kirjaava ohjelma ovat tapa käyttää ohjelmaa. Molemmat ohjelmat on paketoitu suoritettaviksi JAR-paketeiksi. Käyttäjä lisää ajon selainkäyttöliittymän avulla, jolloin ajolle annetaan yksilöllinen numeerinen tunniste. Tätä tunnistetta käytetään yksilöimään ajo kirjattaessa ajojen suoritusaikoja. Käyttäjä voi kirjata ajon käynnistymiset, päättymiset ja virhetilanteet tulkattavissa ohjelmissa seuraavilla komennoilla:

- Ajon käynnistyminen: java -jar ntp_event.jar tunniste start.
- Ajon päättyminen normaalisti: java -jar ntp_event.jar tunniste stop.
- Ajon päättyminen virhetilanteeseen: java -jar ntp_event.jar tunniste error.

Jokaisessa tapauksessa nykyinen aika käydään hakemassa aikapalvelimelta käyttäen SNTP-protokollaa. Ilmatieteen laitoksen aikapalvelimet palauttavan ajan Suomen aikana, joten aika muunnetaan ohjelmassa UTC-ajaksi.

SMS-lokia käsittelevä ohjelma kannattaa suorittaa säännöllisin väliajoin automaattisesti Cron-ohjelman avulla. Ohjelma tarvitsee parametreikseen SMS-palvelimen nimen ja lokitiedostojen nimet. Muut tarvittavat tiedot ohjelma hakee automaattisesti tietokannasta. SMS-loki sisältää jokaisella rivillä aikatiedon, suoritettavan ajon polun ja ajon tilan. Ajojen tilojen (kuva 14) perusteella tiedetään milloin ajo alkaa ja päättyy. SMS-lokia käsittelevä ohjelma kirjaa ajon alkaneeksi, kun ajon tila on aktiivinen ja ajo päättyy valmis- (complete) tai keskeytetty-tilaan (aborted). Keskeytetty-tilaan päättyvät tapahtumat kirjataan virheellisiksi. Ohjelma ohittaa lokirivit, jotka sisältävät muuta tilainformaatiota tai SMS-ohjelman muita viestejä.

SMS-lokitiedostoa lukeva ohjelma käsittelee suuria datamääriä, joten se on hyvä esimerkkitapaus sovelluksen ja Javan suorituskyvystä. Ohjelma käsittelee yhden 80 megatavun kokoisen lokitiedoston n. 20 sekunnissa Linux-työasemalla, jossa on 1 gigatavu keskusmuistia ja 2,4 GHz Pentium 4 prosessori. Tätä tulosta voidaan pitää erinomaisena. Kuormitetussa SGI-tuotantopalvelimessa saman lokitiedostoon käsittelyyn kuluu n. 5 minuuttia, mikä on riittävän hyvä tulos, jotta ohjelma voidaan suorittaa esimerkiksi tunnin väliajoin kuormittamatta palvelinta liikaa. Sovellusta voitaisiin optimoida suorituskyvyn suhteen. Lokitiedostojen käsittelyssä suurin pullonkaula on kuitenkin tietokoneen levynkäsittelyn nopeus, johon ei voida vaikuttaa ohjelmakoodia optimoimalla.

Kaikki sovelluksen osat käyttävät yhteistä asetustiedostoa, joka sijaitsee käyttäjän kotihakemistossa. Asetustiedosto sisältää mm. tietokannan ja NTP-aikapalvelimen määrittelyt.

6.3.6 Käyttäjäroolit

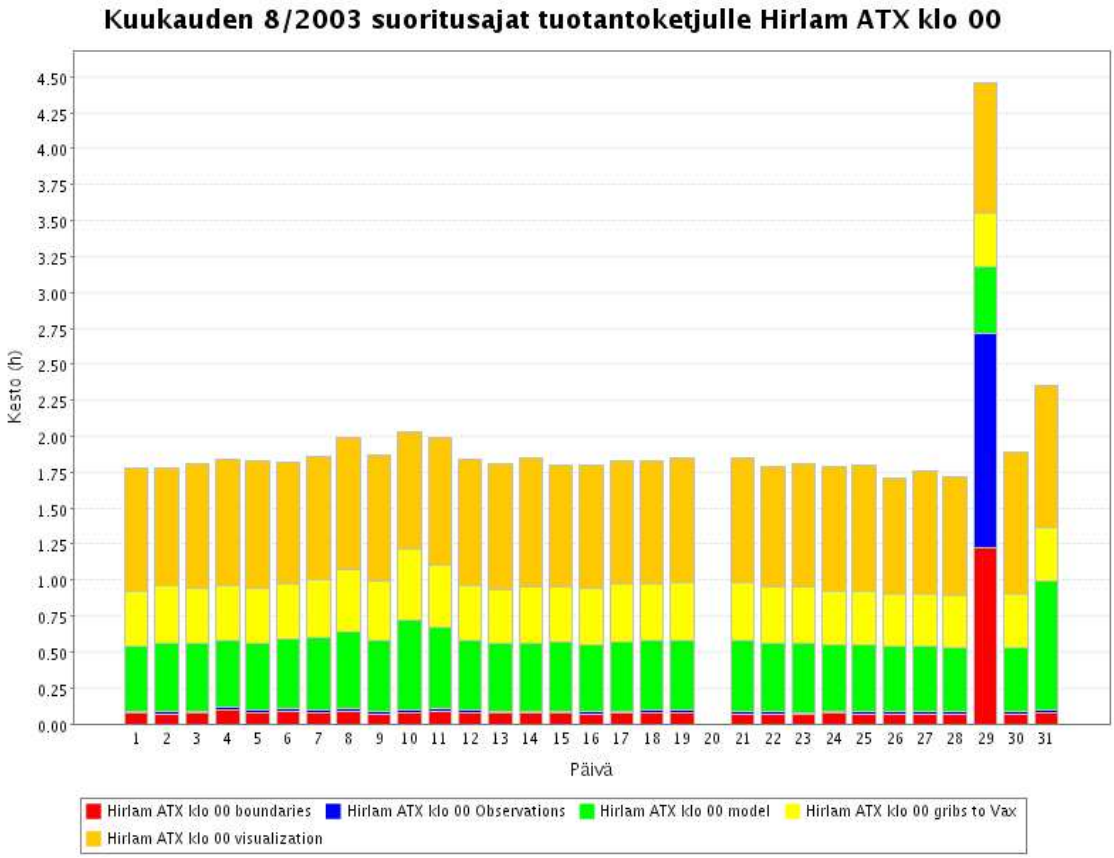
Sovelluksen selainkäyttöliittymään on määritelty käyttäjätunnukset ja salasanat käyttäen servlet-standardin turvallisuusominaisuuksia. Tomcat mahdollistaa käyttäjäroolien määrittelyn yksittäisen tiedoston (servlet-luokat ja JSP-sivut) tai hakemiston tarkkuudella. Käyttäjärooleja ja salasanoja voidaan muokata Tomcatin ja sovelluksen XML-muotoisten asetustiedostojen avulla. Tomcat mahdollistaa myös salattujen yhteyksien käyttämisen autentikoinnissa ja käyttäjien autentikoinnin tietokannasta JDBC:n avulla. Näiden käyttöön ei kuitenkaan ollut tässä tapauksessa tarvetta, koska sovellus toimii Ilmatieteen laitoksen sisäverkossa.

Sovelluksessa on tällä hetkellä kaksi käyttäjäryhmää eli tavalliset käyttäjät ja tuotantoajojen ylläpitäjät. Tavalliset käyttäjä voivat katsella tietoja ja luoda raportteja, mutta eivät muokata tietoja. Tähän luokkaan kuuluvat esimerkiksi käyttäjät, joilla on tarve seurata tuotantoajojen suoritusaikoja, mutta joille ei haluta antaa oikeuksia SMS-valvontaohjelmaan. Tuotantoajojen ylläpitäjät voivat käyttää selainkäyttöliittymän kaikkia ominaisuuksia.

6.3.7 Raportit

Suoritusaikojen tarkastelun lisäksi selainkäyttöliittymä mahdollistaa laajempien raporttien luomisen. Raportteja voidaan luoda ajon, tuotantoketjun tai kaikkien ajojen perusteella. Raporttien ajanjakso voi olla vuorokausi, kuukausi tai vapaavalintainen ajanjakso. Raportit ovat joko graafisia tai tekstimuodossa. Graafisiin raportteihin kuuluvat viiva- ja pylväsdiagrammit sekä Gantt-kaaviot. Kuvassa 18 on esitetty tuotantoketjun perusteella luotu kuukausiraportti.

Graafiset raportit luodaan PNG-tiedostomuodossa (Portable Network Graphics). PNG-kuvatiedostomuoto tukee mm. häviötöntä tiivistystä, 32-bitin väritarkkuutta, läpinäkyvyyttä ja kuvatiedoston tarkentumista asteittain. PNG-kuvatiedostomuodon suuri etu on myös riippumattomuus patenteista. Useimmat nykyaikaiset www-selainohjelmat tukevat PNG-tiedostomuotoa.



Kuva 18. Tuotantoketjun perusteella luotu graafinen kuukausiraportti.

Raportit luodaan käyttäen tuomo.report-paketin luokkia. Graafiset raporttien luodaan käyttäen tuomo.report-paketin luokkia, jotka käyttävät JFreeChart-luokkakirjastoa [35]. JFreeChart osoittautui erittäin monipuoliseksi ohjelmointikirjastoksi graafisten kuvaajien luomiseen. JFreeChart sisältää paljon erityyppisiä kuvaajia, joiden käyttöön on tarjolla monipuoliset ohjelmointirajapinnat.

JFreeChart perustuu lisäksi avoimeen lähdekoodiin, joten ohjelmointikirjastoa voi käyttää ja muokata vapaasti. Ohjelmointirajapinnan vapaasti saatavilla oleva dokumentaatio rajoittuu kuitenkin metodien kuvauksiin ja esimerkkisovelluksiin. Varsinainen dokumentaatio maksaa n. $40 kehittäjää kohden ja rajoittamaton dokumentaatio maksaa n. $250. Dokumentaation hinta on siis suhteellisen kallis. Sovelluksen graafisten raporttien toteuttamista varten tarvittavat ominaisuudet opiskeltiin esimerkkisovelluksista ja ohjeita löytyi myös JFreeChartin kotisivujen keskustelufoorumeista. JFreeChartin virallisten dokumenttien puuttuminen hidastaa

ohjelmointirajapintaa käyttävien sovellusten kehittämistä merkittävästi. JFreeChart on selvästi monipuolisin graafisten kuvaajien ohjelmointirajapinta Javalle, joten sen käyttäminen oli perusteltua.

6.4 Toteutus ja käyttöönotto

Sovelluksen suunnittelu alkoi kesä- ja heinäkuun vaihteessa vuonna 2003. Sovelluksen varsinaisesta suunnittelusta ja ohjelmoinnista vastasi yksin diplomityöntekijä, mutta sovelluksen toteuttamista ohjasivat Pirkko Kilpeläinen, Anu Lahdensuu, Timo Kuoremäki ja Jari Winberg Ilmatieteen laitoksen tietohallinnosta. Suunnitteluvaiheeseen kuului tuotantojärjestelmän toimintaan perehtyminen, relaatiotaulujen ja tietokannan eheyssääntöjen määrittely ja oliomallin suunnittelu UML-kuvauskielen avulla. Varsinainen ohjelmointi alkoi syyskuussa.

Sovelluksen toteuttaminen sujui ilman suuria ongelmia, vaikka käytetyt menetelmät olivat osittain uusia. Erityiset servlet-ohjelmoinnin ja Tomcat-sovelluspalvelimen oppiminen vei aikaa. Servletit ja Tomcat-sovelluspalvelin osoittautuivat kuitenkin hyväksi ratkaisuksi. Tomcatiin liittyvä ainoa ongelma oli, että sitä ei saatu toimimaan luotettavasti Ilmatieteen laitoksen OpenMosix-tyyppisessä Linux-klusterissa. Osoittautui myös, että Java-sovellusten toiminta on testattava jokaisessa ympäristössä, vaikka periaatteessa Java on riippumaton laitealustoista ja käyttöjärjestelmistä. Saman Java-määrityksen toteuttavien virtuaalikoneiden toteutuksissa on kuitenkin pieniä poikkeavuuksia ja jopa ohjelmointivirheitä. Sunin oman virtuaalikoneen Linux-versiossa 1.4.1 oli virhe, joka aiheutti ongelmia tiettyjen päivämäärien käsittelyssä. Tämän takia sovelluksen eräs metodi laski päivämäärän 26.10.2003 kellonaikojen 2:32:37 ja 3:31:28 väliseksi aikaeroksi 1:48:51, vaikka oikea aikaero on 0:48:51. Muilla päivämäärillä virhettä ei esiintynyt ja ongelma on korjattu kyseisen virtuaalikoneen versioon 1.4.2. Muita yhteensopivuusongelmia ei esiintynyt.

Alusta alkaen oli selvää, että sovelluksen toteuttamisen aikataulu on tiukka ja kaikkia ominaisuuksia ei välttämättä ehditä toteuttamaan. Lopulta lähinnä ajojen ja tuotantoketjujen muutosten kirjaaminen sekä osa raporteista jäi toteuttamatta. Nämä ominaisuudet olisi voitu toteuttaa, mutta silloin sovelluksen testaamiselle ei olisi jäänyt

tarpeeksi aikaa. Sovelluksen luotettavuus katsottiin kuitenkin tärkeäksi ominaisuudeksi. Kaikki keskeisimmät ominaisuudet toteutettiin aikataulun puitteissa.

Tätä kirjoitettaessa tammikuun 2004 puolivälissä sovelluksen kaikki ominaisuudet on toteutettu ja sovellus on testikäytössä. Tässä vaiheessa pyritään parantamaan sovelluksen yksityiskohtia ja korjaamaan mahdollisia virheitä. Tähän mennessä sovelluksen toimintaa on testattu lähinnä seuraamalla HIRLAM-sääennustusmallia (High Resolution Limited Area Model) ohjaavan tuotantoketjun suoritusaikoja. HIRLAM on Suomessa ja useissa muissa Euroopan maissa käytetty sääennustusmalli. HIRLAM-tuotantoketjut soveltuivat hyvin sovelluksen testaamiseen. HIRLAM-tuotantoketjut koostuvat useasta osasta ja säämallin prosessointi kestää suhteellisen kauan. Myös suuri osa muusta tuotannosta on riippuvainen säämallin valmistumisesta. HIRLAM-sääennustemallit lasketaan neljä kertaa vuorokaudessa.

Tavoitteena on, että sovellus on riittävän luotettava tuotantokäyttöön tammikuun loppuun mennessä.

6.5 Jatkokehitys

Sovelluksen jatkokehityksestä ei ole päätetty, mutta sovellusta olisi mahdollista kehittää usealla tavalla. Suurin osa jatkokehitysmahdollisuuksista liittyy selainkäyttöliittymään ja sovelluksen oliomalliin, koska SMS-lokitiedostoa käsittelevä ohjelma ja ajojen suoritusaikoja kirjaava ohjelma ovat toiminnaltaan yksinkertaisia.

Selainkäyttöliittymään voitaisiin toteuttaa nyt toteuttamatta jääneet ominaisuudet, kuten muutosten hallinta ja loput raportit. Lisäksi olisi mahdollista parantaa sovelluksen hakuominaisuuksia, jotta esimerkiksi palvelimen vikatilanteissa voitaisiin hakea palvelimessa suoritettavien ajojen vastuuhenkilöt ja asiakkaat. Peräkkäisten tuotantoketjujen lisäksi sovellukseen voitaisiin toteuttaa rinnakkaiset tuotantoketjut, jotta ajojen välisiä riippuvaisuuksia olisi mahdollista tarkastella paremmin. Yleisesti ottaen sovelluksen integraatiota Ilmatieteen laitoksen tuotantojärjestelmään voitaisiin parantaa, jotta sovelluksen tiedot päivittyisivät automaattisesti. Tietojen päivittäminen useaan paikkaan manuaalisesti on työlästä ja vaarana on, että esimerkiksi ajojen tiedot

vanhenevat. Sovellukseen voitaisiin lisäksi mallintaa meteorologisen tiedon jalostamiseen liittyvä käsitemaailma tarkemmin, jolloin erityyppistä ajoista voitaisiin kirjata tarkempaa tietoa.

Myös päinvastainen lähestymistapa olisi mahdollinen, eli sovelluksen yleiskäyttöisyyttä voitaisiin lisätä. Tällöin sovellusta voitaisiin paremmin hyödyntää muissa tarkoituksissa, kuten esimerkiksi ylläpidon rutiinien tarkkailussa tai tutkimuskäytössä aikariippuvaisten algoritmien kehityksessä.

Sovellusta jatkokehitettäessä pitäisi sovelluksen esityskerros eriyttää toimintalogiikasta nykyistä paremmin. Tämä olisi helpompaa nyt, kun käytetyistä menetelmistä on enemmän kokemusta. Automaattisten oliosta relaatioihin kartoitustyökalujen käyttöä kannattaa myös harkita, jos sovelluksen käsitemaailma monimutkaistuu nykyisestä. Nykyisen sovelluksen kehitysajasta suuri osa kului käyttöliittymän toteuttamiseen sekä relaatioiden ja olioiden välisten muunnosten toteuttamiseen. Molemmat asiat on mahdollista automatisoida pidemmälle, jolloin kehitysvaiheen virheet vähenisivät ja sovelluksen ylläpidettävyys paranisi.

7 Yhteenveto

Tässä diplomityössä toteutettiin kaksitasoarkkitehtuurin mukainen oliopohjainen sovellus Ilmatieteen laitoksen tuotantojärjestelmän seurantaan. Sovellus mahdollistaa tuotantojärjestelmän ajojen ja tuotantoajoista koostuvien laajempien tuotantoketjujen suoritusaikojen ja virhetilanteiden tilastollisen seurannan. Sovellukseen on liitetty myös muita ominaisuuksia, kuten ajojen ja tuotantoketjujen asiakkaiden hallinta sekä raporttien luominen suoritusaikojen perusteella. Sovellus luotiin meteorologisen tuotantojärjestelmän tarpeisiin. Vaatimus yleiskäyttöisyydestä johti kuitenkin sovellukseen, jota voidaan käyttää myös muihin tarkoituksiin.

Sovellus koostuu relaatiotietokannasta, SMS-valvontaohjelman lokitiedostoa käsittelevästä ohjelmasta, ajojen suoritusaikoja kirjaavasta ohjelmasta, sovelluksen hallintaan käytettävästä selainkäyttöliittymästä ja kaikille sovelluksen osille yhteisistä luokkakirjastoista. Sovellus toteutettiin kokonaisuudessaan käyttäen Java-ohjelmointikieltä. Selainkäyttöliittymä toteutettiin Java servlettien ja JSP-sivujen avulla.

Sovellus täytti tärkeimmät sille asetetut vaatimukset. Tietokantapohjainen lähestymistapa ajojen suoritusaikojen kirjaamiseen ja seurantaan todettiin toimivaksi. Käytetyt ohjelmistotekniset menetelmät osoittautuivat tehokkaiksi ja monipuolisiksi. Java-servletit ja JSP-sivut mahdollistavat monimutkaisten selainkäyttöliittymään perustuvien sovellusten toteuttamisen täysin oliopohjaisesti. Alustariippumattomuus on myös suuri etu, vaikka sovellusten testaaminen eri ympäristöissä on välttämätöntä. Servlettien, JSP-sivujen ja servlet-ajoympäristöjen opiskeluun kuluu enemmän aikaa kuin yksinkertaisempien menetelmien oppimiseen. Erinomainen suorituskyky ja monipuoliset ohjelmointirajapinnat korostuvat kuitenkin laajoissa verkkosovelluksissa. Myös JDBC-tietokantarajapinta helpottaa merkittävästi tietokantapohjaisten verkkosovellusten toteuttamista.

Suuri osa sovelluksen kehitykseen kuluneesta ajasta käytettiin olioiden ja relaatiotaulujen välisten muunnosten sekä käyttöliittymän toteuttamiseen servlet-luokkien ja JSP-sivujen avulla. Jatkossa kannattaa käyttää näiden asioiden suunnitteluun enemmän aikaa sekä hyödyntää automaattisia työkaluja ja menetelmiä mahdollisimman

paljon. Erityisen tärkeää olisi eriyttää käyttöliittymän esityskerros toimintalogiikasta. Tähän Servlet- ja JSP-teknologiat todettiin osittain puutteellisiksi ja MVC-mallin toteuttaminen kyseisillä tekniikoilla vaatii huolellista suunnittelua. Toteutettu kaksitasoarkkitehtuuri oli kuitenkin hyvä kompromissi sovelluskehitykseen käytetyn ajan ja sovelluksen ylläpidettävyyden suhteen.

Lähteet

[1] E.F Codd. *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, Vol. 13, Number 6, pp. 377-387, 1970.

[2] Ari Hovi, Jouni Huotari, Tapio Lahdenmäki. *Tietokantojen suunnittelu ja indeksointi*. Docendo Finland Oy, 2003.

[3] Rame Elmasri, Shamkant B. Navathe. *Fundamentals of Database Systems.* The Benjaming/Cummings Publishing Company, 1994.

[4] Mark Wutka. *Java J2EE*. Edita Publishing Oy, 2002.

[5] Thomas Connolly, Carolyn Begg, Anne Strachan. *Database Systems – A Practical Approach to Design, Implementation and Management*. Addison-Wesley, 1998.

[6] Kai Koskimies. *Pieni oliokirja*. Suomen Atk-kustannus Oy, 1998.

[7] Roger S. Pressman. *Sofware Engineering – A Practitioner's Approach*, McGraw-Hill, 1997.

[8] Ari Hämäläinen. *Oliotietokannan muuttaminen relaatiotietokannaksi*. Diplomityö, Lappeenrannan teknillinen korkeakoulu, tietotekniikan osasto, 2002.

[9] Oracle AS TopLink. [www-dokumentti] Saatavissa http://otn.oracle.com/products/ias/toplink/index.html. Viitattu 1.10.2003.

[10] Pekka Niskanen, Mikko Kontio, Kimmo Vierimaa. *Inside Enterprise Java*. IT Press, 2000.

[11] Akihiro Saimi, Tsutomu Syomura, Hiroshi Suganuma, Itaru Ishida. *Presentation layer framework of web application systems with server-side Java Technology*. IEEE Computer Software and Applications Conference, pp. 473-478, Taipei, Taiwan, 25-28.10.2000.

[12] Juha Peltomäki. *WWW-ohjelmointi.* Teknolit Oy, 1998.

[13] Alan R. Williamson. *Java Servlets by Example*. Manning Publications Co, 1999.

[14] Server-Side Includes and its Extensions. [www-dokumentti] Saatavissa http://tech.irt.org/articles/js166/#intro. Viitattu 22.11.2003.

[15] Matt Zandstra. *PHP Trainer Kit*. IT Press, 2001.

[16] Bahador Ghahramani, Mark A. Pauley. *Java in High Performance Environments*. IEEE Computer, Volume 36, Issue 9, pp. 109-111, 2003.

[17] The Jakarta Site – Apache Tomcat. [www-dokumentti] Saatavissa http://jakarta.apache.org/tomcat/. Viitattu 23.11.2003.

[18] JavaExchange.com. [www-dokumentti] Saatavissa http://www.javaexchange.com/. Viitattu 7.12.2003.

[19] Java Servlet 2.4 Specification – Final Release. [www-dokumentti] Saatavissa http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html. Viitattu 7.12.2003.

[20] W3C Recommendation, Extensible Markup Language (XML) 1.0 (Second Edition). [www-dokumentti] Saatavissa http://www.w3.org/TR/REC-xml. Viitattu 13.12.2003.

[21] Netcraft: Java Servlet Engines. [www-dokumentti] Saatavissa: http://news.netcraft.com/archives/2003/04/10/java_servlet_engines.html. Viitattu 13.12.2003.

[22] The Apache Struts Web Application Framework. [www-dokumentti] Saatavissa http://jakarta.apache.org/struts/. Viitattu 13.12.2003.

[23] The Java Servlet API White Paper. [www-dokumentti] Saatavissa http://java.sun.com/products/servlet/whitepaper.html. Viitattu 13.12.2003.

[24] Ilmatieteen laitoksen www-sivut. [www-dokumentti] Saatavissa http://www.fmi.fi. Viitattu 14.12.2003.

[25] Ilmatieteen laitoksen liiketoiminnan yhtiöittämisen edellytykset. Liikenneministeriön julkaisu. [www-dokumentti] Saatavissa http://www.mintc.fi/lvm_old/data/www/sivut/dokumentit/julkaisu/julkaisusarja/2001/4801.htm. Viitattu 14.12.2003.

[26] IBM eServer Cluster 1600. [www-dokumentti] Saatavissa http://www.csc.fi/metacomputer/p690.html.fi. Viitattu 20.12.2003.

[27] TOP500 List for November 2003. [www-dokumentti] Saatavissa http://www.top500.org/list/2003/11/. Viitattu 20.12.2003.

[28] ECMWF Manuals: SMS Documentation. [www-dokumentti] Saatavissa http://www.ecmwf.int/publications/manuals/sms/documentation/index.html. Viitattu 20.12.2003.

[29] ECMWF Newsletter Number 84 – Summer 1999. [www-dokumentti] Saatavissa http://www.ecmwf.int/publications/newsletters/pdf/84.pdf. Viitattu 20.12.2003.

[30] Timo Saarinen. *Verkkokäyttöliittymäohjelmiston toteuttaminen TradeXpress-tietojärjestelmään*. Diplomityö, Lappeenrannan teknillinen korkeakoulu, tietotekniikan osasto, 2000.

[31] Katherine Betz, Avraham Leff, James T. Rayfield. *Developing Highly-Responsive User Interfaces with DHTML and Servlets.* IEEE Performance, Computing, and Communications Conference, pp. 437-443, Phoenix, USA, 20-22.2.2000.

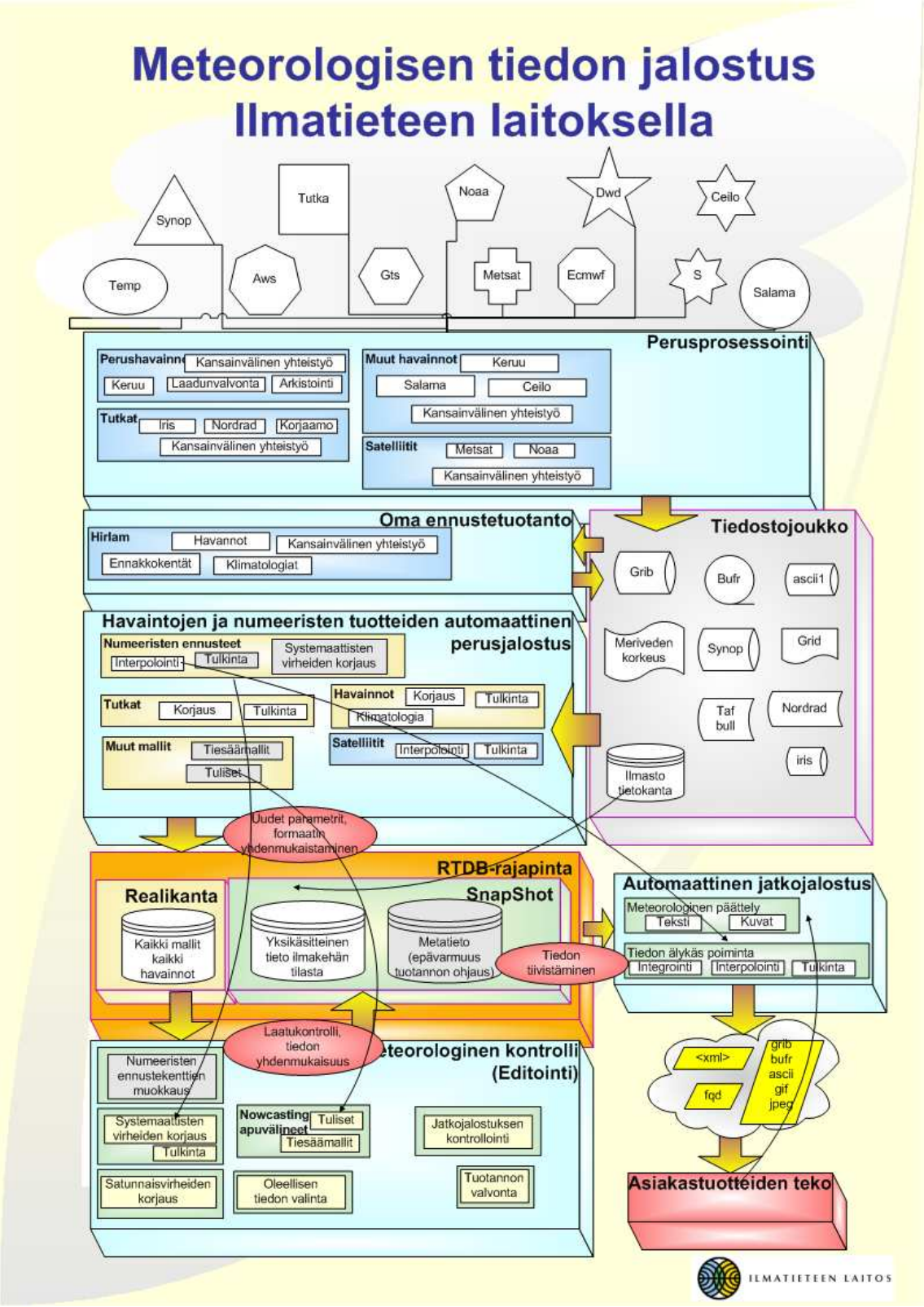
[32] The NTP FAQ and HOWTO. [www-dokumentti] Saatavissa http://www.ntp.org/ntpfaq/NTP-a-faq.htm. Viitattu 21.12.2003.

[33] Lire documentation. [www-dokumentti] Saatavissa http://logreport.org/doc/lire/. Viitattu 28.12.2003.

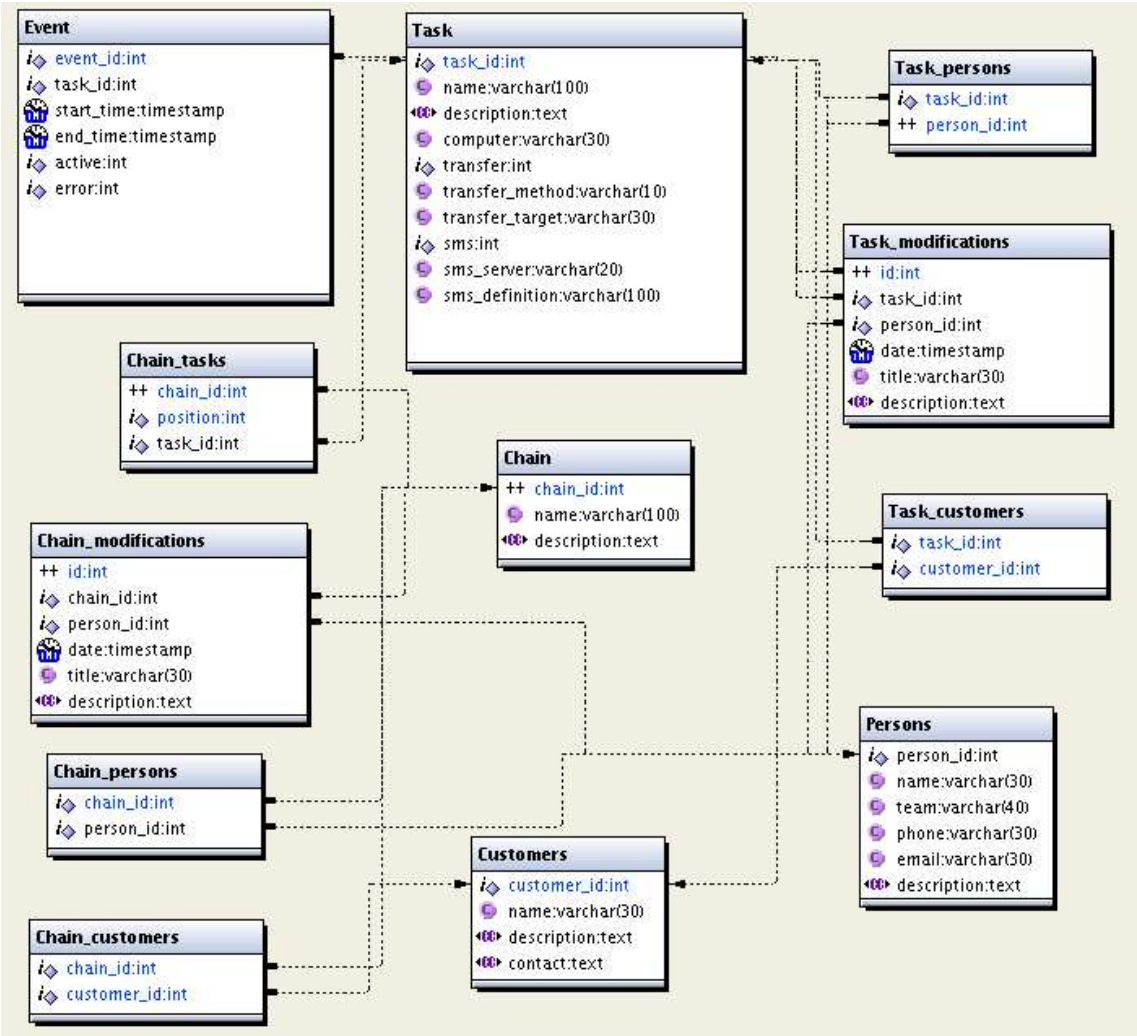
[34] SNTP client in java. [www-dokumentti] Saatavissa: http://alpha.luc.ac.be/Research/Algebra/Members/Java/java.html. Viitattu 1.1.2004.

[35] JFreeChart. [www-dokumentti] Saatavissa http://www.jfree.org/jfreechart/index.html. Viitattu 1.1.2004.

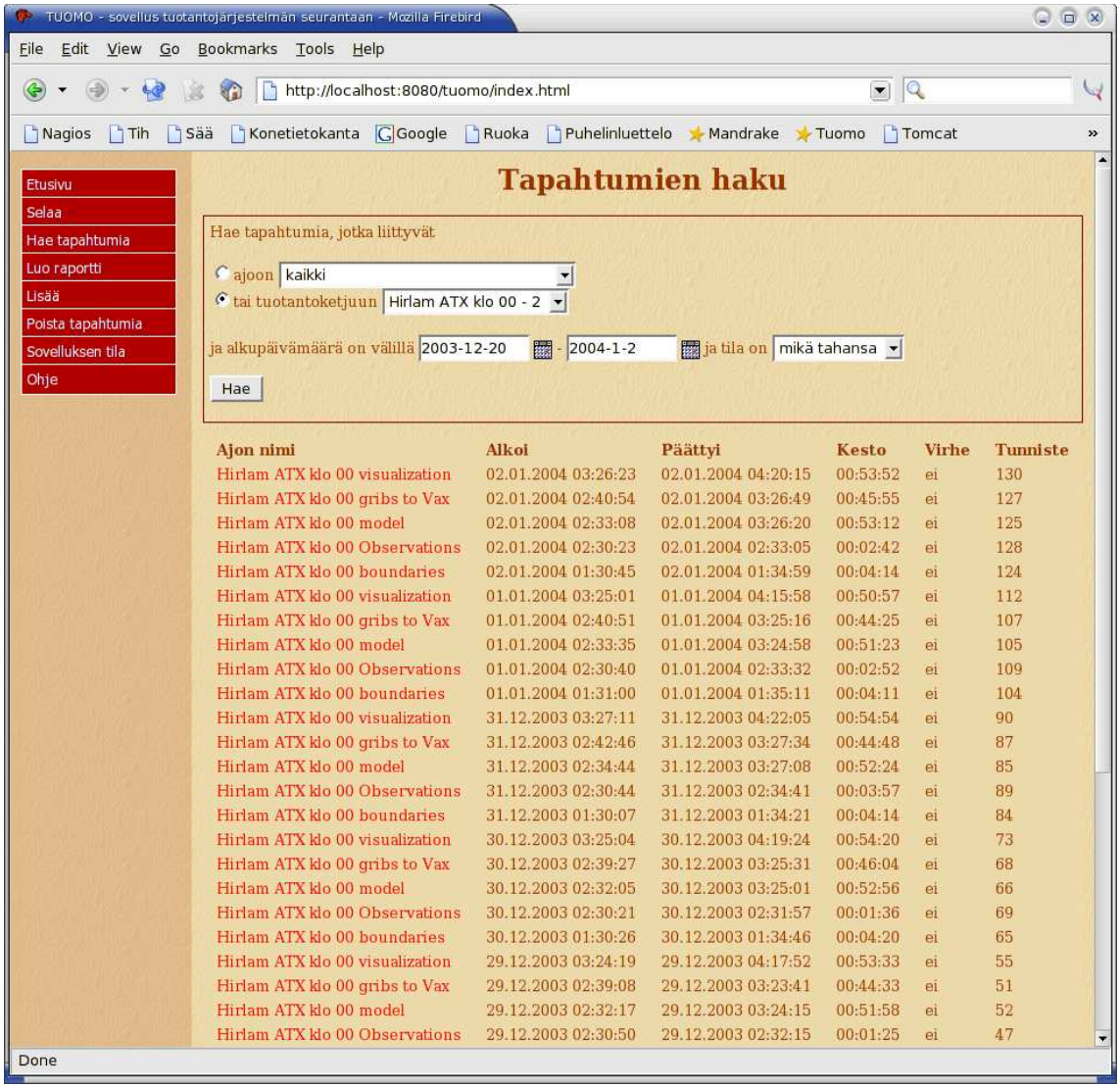
Liite 1: Meteorologisen tiedon jalostus Ilmatieteen laitoksella



Liite 2: Sovelluksen tietokantataulut



Liite 3. Kuvaruutukaappaus sovelluksen käyttöliittymästä



Ajon nimi	Alkoi	Päättyi	Kesto	Virhe	Tunniste
Hirlam ATX klo 00 visualization	02.01.2004 03:26:23	02.01.2004 04:20:15	00:53:52	ei	130
Hirlam ATX klo 00 gribs to Vax	02.01.2004 02:40:54	02.01.2004 03:26:49	00:45:55	ei	127
Hirlam ATX klo 00 model	02.01.2004 02:33:08	02.01.2004 03:26:20	00:53:12	ei	125
Hirlam ATX klo 00 Observations	02.01.2004 02:30:23	02.01.2004 02:33:05	00:02:42	ei	128
Hirlam ATX klo 00 boundaries	02.01.2004 01:30:45	02.01.2004 01:34:59	00:04:14	ei	124
Hirlam ATX klo 00 visualization	01.01.2004 03:25:01	01.01.2004 04:15:58	00:50:57	ei	112
Hirlam ATX klo 00 gribs to Vax	01.01.2004 02:40:51	01.01.2004 03:25:16	00:44:25	ei	107
Hirlam ATX klo 00 model	01.01.2004 02:33:35	01.01.2004 03:24:58	00:51:23	ei	105
Hirlam ATX klo 00 Observations	01.01.2004 02:30:40	01.01.2004 02:33:32	00:02:52	ei	109
Hirlam ATX klo 00 boundaries	01.01.2004 01:31:00	01.01.2004 01:35:11	00:04:11	ei	104
Hirlam ATX klo 00 visualization	31.12.2003 03:27:11	31.12.2003 04:22:05	00:54:54	ei	90
Hirlam ATX klo 00 gribs to Vax	31.12.2003 02:42:46	31.12.2003 03:27:34	00:44:48	ei	87
Hirlam ATX klo 00 model	31.12.2003 02:34:44	31.12.2003 03:27:08	00:52:24	ei	85
Hirlam ATX klo 00 Observations	31.12.2003 02:30:44	31.12.2003 02:34:41	00:03:57	ei	89
Hirlam ATX klo 00 boundaries	31.12.2003 01:30:07	31.12.2003 01:34:21	00:04:14	ei	84
Hirlam ATX klo 00 visualization	30.12.2003 03:25:04	30.12.2003 04:19:24	00:54:20	ei	73
Hirlam ATX klo 00 gribs to Vax	30.12.2003 02:39:27	30.12.2003 03:25:31	00:46:04	ei	68
Hirlam ATX klo 00 model	30.12.2003 02:32:05	30.12.2003 03:25:01	00:52:56	ei	66
Hirlam ATX klo 00 Observations	30.12.2003 02:30:21	30.12.2003 02:31:57	00:01:36	ei	69
Hirlam ATX klo 00 boundaries	30.12.2003 01:30:26	30.12.2003 01:34:46	00:04:20	ei	65
Hirlam ATX klo 00 visualization	29.12.2003 03:24:19	29.12.2003 04:17:52	00:53:33	ei	55
Hirlam ATX klo 00 gribs to Vax	29.12.2003 02:39:08	29.12.2003 03:23:41	00:44:33	ei	51
Hirlam ATX klo 00 model	29.12.2003 02:32:17	29.12.2003 03:24:15	00:51:58	ei	52
Hirlam ATX klo 00 Observations	29.12.2003 02:30:50	29.12.2003 02:32:15	00:01:25	ei	47