# Kiuru-sovelluksen arkkitehtuuri

Miika Nurminen (minurmin@cc.jyu.fi)

9. kesäkuuta 2003

**Tiivistelmä**

Tenttivastauksessa käsitellään Korppi-järjestelmän ja erityisesti Kiuru-salivaraussovelluksen arkkitehtuuria. Arkkitehtuuria tarkastellaan Ohjelmistoarkkitehtuurit-kurssilla käsiteltyjen arkkitehtuurityylien ja kurssin tenttitilaisuudessa annettujen vastausten näkökulmasta [OA2003]. Tenttivastaukseen on jälkikäteen tehty vähäisiä korjauksia Kiuru-erikoistyötä varten.

## 1 Sovellusprojektin arkkitehtuuri

Korppi-järjestelmä (katso `https://korppi.it.jyu.fi/`) on viiden eri tietotekniikan sovellusprojektin (katso `http://kotka.it.jyu.fi/`) työn tulos. Järjestelmää on jatkokehitetty myös sovellusprojektien ulkopuolella. Järjestelmän perusrakenne määriteltiin Kotka-projektissa [Kotka], mutta jokainen myöhemmistä projekteista on kehittänyt ja muokannut arkkitehtuuria edelleen omassa moduuulissaan.

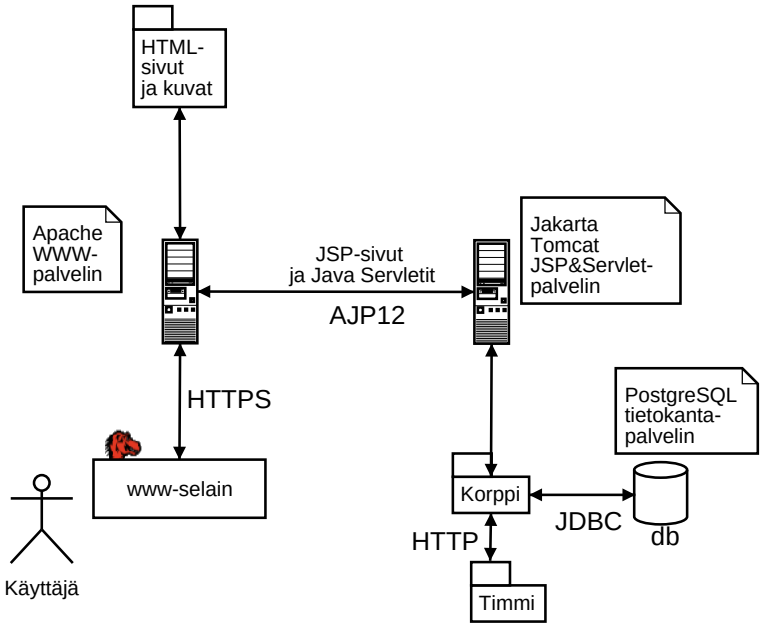
Korppi-järjestelmä noudattaa karkealla tasolla 3-kerrosarkkitehtuuria: käyttöliittymä muodostuu HTML- ja JSP- sivuista, sovelluslogiikka on pääosin Tomcat-palvelimella sijaitsevissa Java-pavuissa ja kaikki pysyvä tieto on PostgreSQL-tietokantapalvelimella. Järjestelmän rakenne on kuvan 1 mukainen [Kiuru].

Liitäntä Timmi-järjestelmään suunniteltiin ja osittain toteutettiin Kiuru-projektina aikana, mutta siitä luovuttiin Timmin alasajon myötä. Vastauksessa käsitellään sovelluksen arkkitehtuuria olettaen, että liitäntä olisi käytössä, koska liitännän määrittäminen oli keskeinen osa projektia.

### 1.1 Korppi-järjestelmästä ja aiemmista projekteista

Kerrosarkkitehtuurin lisäksi Korppi-järjestelmän keskeisenä tavoitteena on ollut modulaarisuus käyttöliittymän ja sovelluskerroksen toimintojen suhteen. Järjestelmän moduuleja ovat mm. portaali (käyttäjän tunnistus ja viestijärjestelmä), kurssikirjanpito, kalenteri, opinnäytteiden hallinta ja salivarausjärjestelmä. Moduulien toteuttaminen on ollut projektiryhmien ja jatkokehityksen vastuulla.

Periaatteessa järjestelmän pitäisi toimia, vaikka jokin moduuli otetaan pois, mutta käytännössä moduulien välillä on kytköksiä. Järjestelmän ytimen muodostavat portaali- ja kurssikirjanpitomoduulit. Kalenteri ja opinnäytteiden hallinta ovat suhteellisen riippumattomia toisistaan, mutta käyttävät osittain samoja tietokantatauluja kuin kurssikirjanpito.




Kuva 1: Korppi-järjestelmän rakenne.

Salien varausmoduulista on yhteys sekä kalenteriin että kurssikirjanpitoon. Jokainen projekti on joutunut lisäämään uusia tauluja tietokantaan ja myös muokkaamaan olemassaolevia.

Sovelluskerroksen sisäinen totetutus poikkeaa jonkin verran eri projekteissa. Kotka- ja Korppi-projekteissa Java-papuihin toteutettiin yleiskäyttöiset aliohjelmat, joita kutsuttiin JSP-sivuilta. Käytännössä myös huomattava osa sovelluslogiikkaa (jopa SQL-lauseita) on sijoitettuna suoraan JSP-sivuille osin ajanpuutteen, osin puutteellisen arkkitehtuurisuunnittelun takia [Kotka, Korppi].

Kolibri-projektissa suurin osa koodista kirjoitettiin suoraan JSP-sivuille, johon oli syynä ajanpuute ja tarve saada aikaan toimiva sovellus. Myös uusia Java-papuja toteutettiin, mutta kaikki käyttöliittymäkoodi toteutettiin servlettien tapaan `out.println()`-tulosteilla, hyödyntämättä JSP-sivujen mahdollisuuksia. Tämä tekee Kolibri-projektin sivuista vaikeahkosti luettavia ja ylläpidettäviä [Kolibri].

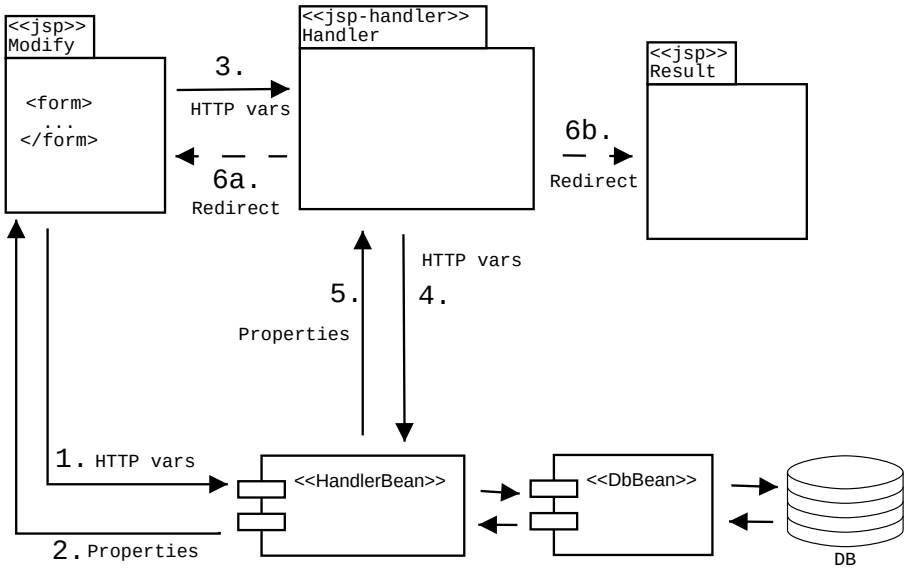
Myöhemmistä projekteista Koppelo meni tavallaan toiseen ääripäähän hylkäämällä Java-pavut kokonaan ja sijoittamalla sekä käyttöliittymään että sovelluslogiikkaan liittyvän koodin kustomoituihin JSP-tageihin. Ratkaisu toimi projektin puitteissa hyvin, mutta on aiheuttanut muilla sivuilla yhteensopivuusongelmia, jos on ollut tarvetta vaihtaa tietoa Java-papujen ja kustomoitujen tagien välillä [Koppelo].

## 1.2 Kiuru-projektista

Kiuru-projektin yhtenä osatavoitteena oli selkeyttää järjestelmän arkkitehtuuria (tosin ryhmän jäsenet eivät siinä vaiheessa tunteneet *Ohjelmistoarkkitehtuuri*-käsitteen merkitystä). Tavoitteena oli aiempia projekteja selkeämpi sovelluslogiikan ja käyttöliittymäkoodin erottaminen toisistaan ja JSP-sivuille kirjoitetun Java-koodin minimoiminen. Lisäksi luotiin uusia komponentteja tietokannan ja merkkijonojen käsittelyyn sekä HTML-koodin generointiin. Kiuru-projektia jatkavan erikoistyön yhteydessä komponenttien kehitystä on

jatkettu edelleen.

Kaaviossa 2 esitetään vaiheittain tietojen käsittelylogiikka johonkin tietokohteeseen tehtävien pyyntöjen ja muokkausten osalta. Kuvaus on esimerkki Kiuru-sovelluksen arkkitehtuurista ja on muokattu Kiuru-projektin sovellusraportin [Kiuru] pohjalta.




Kuva 2: JSP-sivujen käsittely.

Oletetaan, että `Modify`-sivu on aluksi saanut HTTP-parametrina tiedon siitä, mitä sivulla tehdään (esim. lisäys tai muokkaus) ja käsiteltävän tietokohteen tunnisteen. Pyynnön käsittelyvaiheet ovat seuraavat:

1. JSP-sivun HTTP-parametrit asettavat `Handler`-pavun vastaavat ominaisuudet (tieto siitä, mitä sivulla tehdään on myös ominaisuutena). `HandlerBean` hakee tarvittaessa tiedot tietokannasta `Dbbean`-pavun avulla.
2. Sivun tiedot generoidaan `HandlerBean`-pavun ominaisuuksien pohjalta.
3. Käyttäjän syöttämät `Modify`-lomakkeen tiedot lähetetään HTTP-muuttujina `JSP-Handler` -sivulle.
4. HTTP-parametrit asettavat `HandlerBean`-pavun vastaavat ominaisuudet. Jos syötteet ovat kelvollisia, `HandlerBean` päivittää tarvittaessa tietokannan tiedot.
5. `HandlerBean` palauttaa ominaisuuksissaan tiedon, onko päivitys onnistunut ja ovatko syötteet kelvollisia.
6. a) Jos tietojen päivitys ei onnistunut, `JSP-Handler`-sivu ohjaa kutsun takaisin `Modify`-sivulle ja ilmoittaa käyttäjälle virheistä.
   b) Muuten `JSP-Handler` ohjaa käyttäjän uudelle sivulle, jossa hän voi jatkaa.

Korppi-järjestelmän ulkopuolelta Kiuru-sovellus käyttää Timmi-salivarausjärjestelmää ja pdflatex-ohjelmaa. Timmi-järjestelmän kanssa kommunikoidaan HTTP-pyynnöillä, vastaukset tulevat XML-muotoisina viesteinä, jotka käsitellään W3C:n DOM-rajapintaa tukevalla jäsentimellä. pdflatex-ohjelmaa käytetään tulostettavien lukujärjestysten generointiin. Ohjelmaa käytetään Javan `Runtime.exec()`-metodin kautta kutsuttavana ajonaikaisena prosessina.

# 2 Sovelluksen onnistuminen

Luvussa käsitellään ensin järjestelmän yleisiä laatuvaatimuksia ja sitten niiden toteutumista Kiuru-projektissa.

## 2.1 Järjestelmän laatuvaatimukset

Korppi-järjestelmän keskeisimpiä laatuvaatimuksia ovat siirrettävyys, laajennettavuus, tietoturvallisuus ja käytettävyys [Kotka]. Näistä siirrettävyys ilmenee pyrkimyksenä käytttää standardin mukaista SQL:ää, HTML:ää sekä vakiintuneita palvelimia ja tekniikoita. Myös Java-kieli valittiin alunperin toteutuskieleksi siirrettävyytensä takia.

Laajennettavuus on olennainen ominaisuus, koska eri projektit lisäävät uusia moduuleja Korppi-järjestelmään. Laajennettavuutta tukee tietokantapalvelimen erottaminen muusta järjestelmästä, mikä mahdollistaa esim. erilaisten käyttöliittymien teon. Laajennettavuuden vaatimus saattaa tosin heikentää järjestelmän ylläpidettävyyttä, koska eri projektit tekevät osittain päällekkäistä työtä. Riskinä on, että järjestelmä kasvaa tulevaisuudessa niin laajaksi, etteivät kehittäjät ymmärrä sitä kokonaisuutena.

Koska järjestelmää on tarkoitus käyttää koko yliopiston laajuudella, myös tietoturvallisuus on välttämätön vaatimus. Tietoturvallisuutta tukevat https-protokollan käyttö, salasanan kryptaus ja eri käyttäjäoikeustasojen toteuttaminen. Jokaiselle sivulle voidaan määritellä vaadittava käyttäjätaso, jota ilman sivua ei näytetä. Käyttäjätasot tarkistetaan jokaisen sivupyynnön yhteydessä käyttäjän HTTP-istuntoon perustuen. Vastaavat tarkistukset pyritään tekemään myös sovelluskerroksella tietokantaa käsittelevissä pavuissa.

Käytettävyys on keskeinen edellytys Korppi-järjestelmän yleistymiselle tietotekniikan laitoksen ulkopuolella. Käytettävyys on otettu huomioon erityisesti Korppi-projektissa [Korppi], jossa visuaalinen suunnittelija huolehti sovelluksen käyttöliittymästä. Sovellusarkkitehtuurissa käytettävyyttä tukevat käyttöliittymän ja sovelluslogiikan erottaminen, jolloin JSP-sivujen ulkoasua voi ainakin periaatteessa editoida ilman ohjelmointitaitoa. Käytännössä tämä tosin onnistuu vain harvoilla järjestelmän sivuilla. Käyttäjän muokattavissa olevat CSS-tyylit tukevat myös käytettävyyttä.

## 2.2 Kiuru-sovelluksen onnistuminen

Kiuru-projektissa sovellettiin projektiryhmän itse asettamia suunnitteluperiaatteita. Käyttöliittymän ja sovelluslogiikan erottamista varten luotiin useita Java-papuja tietoalkioiden (ja tietokannan) käsittelyyn, tietojen etsimiseen ja käyttöliittymäelementtien luontiin. JSP-sivujen käsittelylogiikan (katso luku 1.2) ansiosta käyttöliittymäsivujen koodimäärä väheni murto-osaan aiempien projektien sivuista, mutta *Handler*-sivuille sovelluslogiikkaa jäi jonkin verran.

Pienemmästä koodimäärästä huolimatta Java-papujen koodi oli huomattavasti epäselvempää edellisiin projekteihin verrattuna. Syynä tähän oli HTTP-parametrien käsittely suoraan tietokohdetta esittävässä Java-pavussa. Projektiryhmä koki tämän kuitenkin paremmaksi ratkaisuksi kuin laajojen (uudelleenkäyttöön kelvottomien) koodipätkien kirjoittamisen suoraan JSP-sivuille.

Kiuru-projektin tekemät tietokantamuutokset heikensivät järjestelmän siirrettävyyttä ja monimutkaistivat tietokannan rakennetta. Kotka-projektin suunnitelmissa ei ollut alunperin varauduttu salivarausten (erityisesti Timmi-järjestelmän) aiheuttamiin vaatimuksiin,

jotka olivat osin ristiriitaisia kalenterin ja kurssikirjanpidon vaatimusten kanssa (ja käyttävät kaikki samoja tietokantatauluja).

Olemassaolevien moduulien toimivuuden varmistamiseksi Kiuru-projekti loi uusia varauksiin liittyviä tietokantatauluja, joiden toiminta oli osittain päällekkäistä olemassaoleviin tauluihin nähden. Vanhat ja uudet taulut synkronoitiin PostgreSQL:n epästandardilla sääntörakenteella `CREATE RULE`, joka toimii eräissä tietokantapalvelimissa käytettävien herättimien (engl. *trigger*) tapaan. Epästandardit SQL-lauseet dokumentoitiin tietokantaraporttiin [Kiuru], mutta niiden toimivuus ei ole taattua, jos tietokantapalvelinta joudutaan vaihtamaan.

Käytettävyyteen liittyvänä laatuvaatimuksena on myös mahdollisuus järjestelmän monikielistämiseen. Kiuru-projektin kuluessa järjestelmässä oli käytössä kaksi rinnakkaista monikielistystekniikkaa (tietokantaan tai `Translation`-papuun perustuvaa), joista ensimmäisestä ollaan luopumassa ja toisen toimivuudesta monimutkaisilla lauseilla ei ollut kokemuksia. Kiuru-projektin aikana `Translation`-pavun käytöstä ei saatu yhtenäistä linjausta aikaan, jonka seurauksena Kiuru-projektin kaikki sivut eivät toimi kunnolla eri kielillä.

# 3 Parannusehdotuksia arkkitehtuuriin

Järjestelmän perusrakenne kolmikerrosarkkitehtuurina on toimiva, mutta sen soveltaminen eri sovelluskerroksen moduuleissa on vaihtelevaa. Ylläpidettävyyden parantamiseksi kaikkien moduulien koodit pitäisi muuntaa niin, että sovelluslogiikka ja käyttöliittymäkoodi ovat aidosti toisistaan erotettuja ja eri moduulit käyttävät yhteisiä komponentteja nykyistä laajemmassa määrin.

Tietokannan käsittely on tällä hetkellä hajallaan ympäri järjestelmää. Tietokantaa tosin hallitaan yhdellä `DB`-pavulla, mutta SQL-lauseiden luonti ja tietokannan vasteiden käsittely hoidetaan hyvin vaihtelevasti. SQL-lauseiden käsittely pitäisi poistaa JSP-sivuilta kokonaan ja tietokannasta tulevat tiedot pitäisi eristää omaksi kerroksekseen nykyistä laajemmin. Tällä hetkellä järjestelmä on hyvin herkkä tietokantaan tehtäville muutoksille. Esimerkiksi vanhasta `Translation`-tauluihin perustuvasta käännösjärjestelmästä luopuminen johti kymmenien taulujen poistamiseen ja satojen (?) SQL-rivien korjaamiseen käsin (Korppi-ylläpito tietänee tarkat tilastot).

Tietokannan käsittelylogiikan lisäksi myös JSP-sivujen käsittelylogiikka pitäisi yhtenäistää. Luvussa 1 on mainittu eri projekteissa sovellettuja tekniikoita, joista itse suosittelen (puolueellisesti) Kiuru-projektissa käytettyä. Projektin aikana huomattava osa toteutuksen alkuvaihetta meni lähinnä JSP-sivujen käytön opetteluun. Yhtenäinen käytäntö helpottaisi järjestelmän ylläpidettävyyttä ja tulevien projektien toimintaa.

Järjestelmän kasvaessa testauksen merkitys kasvaa entisestään. JSP-sivujen debuggaus ja testaus on tähän asti hoidettu suurilta osin käsin, mikä heikentää järjestelmän luotettavuutta järjestelmää ylläpidettäessä ja laajennettaessa. Järjestelmän arkkitehtuuria pitäisi muokata niin, että automaattinen testaus helpottuu. Java-papujen yksikkötestaus on toki helppoa, mutta integrointi- ja järjestelmätestaus on tällä hetkellä työlästä. Käsittääkseni Java-kielellä on toteutettu apuvälineitä testausta varten, mutta en ole vielä tutustunut niihin tarkemmin. Vesa Lappalainen on myös ehdottanut vakiomuotoista testisarjaa, joka suoritettaisiin ennen jokaista järjestelmän päivitystä (esim. kurssin lisäys, tapahtuman lisäys, salivaraus ym. perustoiminnot).

Eräs Kotka-projektin [Kotka] suunnitelmissa mainittu ja erikoistyöryhmässä pohdittu

mahdollisuus olisi muuttaa Korppi-järjestelmä aidoksi J2EE-sovellukseksi ja Java-papujen muokkaaminen EJB-komponenttimallin mukaiseksi. Tämä parantaisi järjestelmän laajennettavuutta, ylläpidettävyyttä ja eri kerroksilla olevien tietojen erottamista toisistaan. Projektin työläyden ja käytettävissä olevien resurssien (rahoitus, opiskelijaprojektien aikataulu) puutteen takia tämä ei kuitenkaan liene realistinen ratkaisu ainakaan lyhyellä aikavälillä.

Helposti toteutettavissa olisi ainakin kunnollisen ”Korppi-koodausstandardin” ja järjestelmän moduulien riippuvuussuhteiden määrittely, jota ainakin tulevat projektit voisivat noudattaa. Tämänhetkinen, epävirallinen koodausstandardi on saatavilla WWW-sivulta `https://jamshedpur.it.jyu.fi/korppicoding.txt`.

# 4 Tenttivastausten pohdintaa

Kirjoittajan mielestä tietoisuus ohjelmistoarkkitehtuureista olisi helpottanut Kiuru-projektin (ja varmasti myös aiempien projektien) toimintaa. Projektin aikana tuotettavissa suunnitelmissa ja raporteissa keskityttiin toisaalta käyttötapauksiin, toisaalta käyttöliittymään, toisaalta tietokantaan ja loppuvaiheessa yksittäisten Java-luokkien dokumentointiin. Väliin jää jälkeenpäin katsottuna arkkitehtuurin mentävä aukko, järjestelmän yleisen rakenteen kuvaaminen tikku-ukkokaavioita (ks. esim. kuva 1) tarkemmalla, mutta luokkakaavioita abstraktimmalla tasolla.

Arkkitehtuurityylien tunnistaminen olemassaolevasta sovelluksesta voi olla hankalaa etenkin, jos sovellus on alunperin tehty ilman selkeää näkemystä arkkitehtuurista. Vaikka tällainen näkemys olisikin olemassa, monisteessa esitetyt tyylit ovat mielestäni puhtaudessaan epärealistisia todellisiin sovelluksiin verrattuna. Uskon, että oikeiden järjestelmien arkkitehtuuri tulee olemaan useimmiten mukautettu (tai sovellusaluekohtainen), jossa on toki havaittavissa elementtejä eri tyyleistä.

Arkkitehtuurit pitäisi saada *näkymään* tulevissa sovelluksissa. Tämä edellyttää mielestäni joko

1. selkeän kuvauksen muodostamista tietystä arkkitehtuurityylistä toteutukseen (vrt. arkkitehtuurikielten lupaus), tai
2. tiettyä arkkitehtuurityyliä noudattavan sovelluskehyksen käytön järjestelmää toteutettaessa (vrt. Borlandin käyttöliittymäkomponentit tapahtumanpohjaisessa järjestelmässä).

Haluaisin edelleen nähdä, kuinka käytännössä onnistuu olemassaolevan (ja pahimmillaan spagettimaisen) sovelluksen arkkitehtuurin muuntaminen kirjoittamatta koko sovellusta uudelleen! Toivon, että se olisi mahdollista ja ehkä vielä joskus automaattista.

# Lähteet

[Kiuru] Hilpinen Toni, Koivuniemi Marko, Mäkinen Jussi ja Nurminen Miika, ”Kiuru-projektin projektikansio”, Jyväskylän yliopisto, Tietotekniikan laitos, 2002.

[Kolibri] Jaakkola Mia, Juutinen Sanna, Lupari Matti ja Nieminen Mikko, ”Kolibri-projektin projektikansio”, Jyväskylän yliopisto, Tietotekniikan laitos, 2001.

[Koppelo] Hillebrand Minna, Silván Markus, Vanhanen Antti ja Ylitalo Marko, ”Koppelo-projektin projektikansio”, Jyväskylän yliopisto, Tietotekniikan laitos, 2002.

[Korppi] Lesonen Minna, Pekkanen Hannu, Tawast Tuukka ja Uuksulainen Heikki, ”Korppi-projektin projektikansio”, Jyväskylän yliopisto, Tietotekniikan laitos, 2001.

[Kotka] Horppu Ismo, Mielityinen Markku ja Vire Markku, ”Kotka-projektin projektikansio”, Jyväskylän yliopisto, Tietotekniikan laitos, 2000.

[OA2003] Jonne Itkonen, Ohjelmistoarkkitehtuurit-kurssin kotisivu, Jyväskylän yliopisto, Tietotekniikan laitos, 2003, http://www.mit.jyu.fi/ji/opetus/oa2003/.