Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Tahvo Repo

Opinnäytetyö

Näkökulmia EJB-komponenttikerroksen päivitykseen Java EE –sovelluksessa

Työn ohjaaja Maritta Hoffren, FM
Työn tilaaja Logia Software Oy, valvojana Merja Järvinen
Tampere 05/2009

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Tahvo Repo
Työn nimi	Näkökulmia EJB-komponenttikerroksen päivitykseen Java EE –sovelluksessa
Sivumäärä	40
Valmistumisaika	29.05.2009
Työn ohjaaja	Maritta Hoffren
Työn tilaaja	Logia Software Oy

TIIVISTELMÄ

Tämän työn toimeksiantaja on työnantajani, pääosin Tampereella toimiva logistiikan IT-ratkaisuja tarjoava yritys, Logia Software Oy. Osallistun työtehtävissäni ParLet -sovelluksen jatkokehitykseen. ParLet on Itella Oyj:n tilaama kansainvälisen postiliikenteen rekisteröinti- ja seurantajärjestelmä. ParLetin liiketoimintakerros on toteutettu EJB 2.0 -tekniikalla, jota voidaan pitää ominaisuuksiltaan vanhentuneena. Tulevaisuudessa ParLetin suoritusympäristöön suoritetaan versionnostoja. Versionnostot tekevät mahdolliseksi siirtymisen uudempaan EJB 3.0 -versioon, joten toimeksiantaja näki siirtymään liittyvien tekijöiden kartoituksen tarpeelliseksi.

Tavoitteenani oli kartoittaa mahdolliseen päivitysprosessiin liittyviä tekijöitä, tuottaa kuvaus työn luonteesta sekä esitellä käytettävissä olevia päivitysmalleja ja ongelman rajausmahdollisuuksia. Työn tarkoitus on antaa lukijalle kuva siitä, millainen päivitysprosessi olisi todellisuudessa. Tämän pohjalta toimeksiantaja osaisi määrittää tulevaisuuden toimenpiteitä entistä paremmin.

Pyrin työssäni tarkastelemaan tekniikoita ParLetissa käytettyjen ratkaisujen kannalta. Lähdeaineistona käytin ParLetin osalta sovelluksen kehitysdokumentteja ja asiantuntijahaastatteluja. Java- ja EJB -tekniikoita käsittelevinä lähteinä käytin Sun Microsystemsin spesifikaatioita sekä aiheeseen liittyviä kirjoja ja verkkoartikkeleita.

Lähdemateriaalin pohjalta laadin yhteenvedon käytettävissä olevista päivitysmahdollisuuksista hyvine ja huonoine puolineen. Lisäksi pystytin uutta tekniikkaa toteuttavan koeympäristön, johon toin komponentin olemassa olevasta ParLetin kehitysversiosta.

Siirtyminen EJB 3.0 -tekniikkaan olisi työmäärältään suuri prosessi, ja päivitys voitaisiin suorittaa monin eri tavoin. Toivon tässä työssä esillenousseiden tekijöiden antavan toimeksiantajalle suuntaa ja arvokasta taustatietoa tulevaisuuden toimenpiteitä arvioidessa. Vaikka suurten sovellusten migraatiot ovat aina tapauskohtaisia, voidaan tämän työn tuloksia pitää suuntaa-antavina myös muiden EJB 2.x -pohjaisten sovellusten päivitystarvetta ja -strategiaa suunniteltaessa.

Avainsanat ohjelmistoarkkitehtuurit, Java EE, EJB, migraatio

TAMK University of Applied Sciences
Busines Information Systems

Writer	Tahvo Repo
Thesis	Viewpoints on the Migration of EJB Tier in Java EE Application
Pages	40
Graduation date	29.05.2009
Thesis Supervisor	Maritta Hoffren
Co-operating Company	Logia Software Inc.

ABSTRACT

This thesis was commissioned by my employer, Logia Software Inc., which is a Finnish software company specialising in logistic solutions. In my work, I participate in the development of application ParLet. It is an application for follow-up, registering and maintaining international mail flow, commissioned by Itella Corporation, Finland. The business logic layer of ParLet is based on EJB 2.0 technology, which may be considered slightly outdated in terms of technology. In the future, when the current technical environment will be facing version updates, it would possible to move onto the present version, EJB 3.0. This is why the employer saw it necessary to research the facts concerning the version migration.

My goal was to map the factors of possible migration process, produce a description of the nature of the work and introduce some methods to handle the process. After digesting the information in this thesis, the reader should have a clear picture of the conventions of the process. This should help the employer in making any future decisions concerning this topic.

I aimed to investigate EJB technologies in light of the solutions used in ParLet. I used ParLet development specifications as source material, and utilised the development team as consultants. Regarding Java and EJB technologies, I turned to Sun Microsystems specifications as well as tutorial books and network articles by experts of the field.

From the base of the source materials and my own conclusions I established a round-up of the migration methods and provided them with benefits and downsides. In addition, I built an experimental environment utilising the new EJB version, provided with a component from the existing development version of ParLet.

Migration to EJB 3.0 in ParLet would be a large-scale operation in the terms of workload, but there are plenty of methods to execute the work. I hope the results found from this thesis works will provide the employer with valuable information on the topic, and will be of help in making the future decisions. Although the migration of a large-scale application is always more or less case-specific, the results of this thesis can be considered suggestive whenever defining the need and strategy for migration of EJB 2.x based applications.

Keywords Software Architecture, Java EE, EJB, Migration

Sisällysluettelo

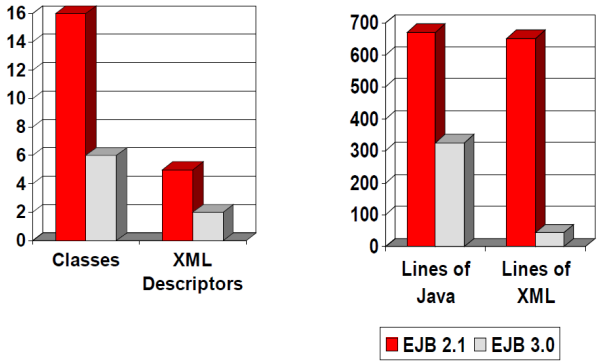
1 Johdanto

Opinnäytetyöni toimeksiantaja on työnantajani, pääosin Tampereella toimiva logistiikan IT -ratkaisuihin erikoistunut yritys, Logia Software Oy. Vuonna 1975 toimintansa Tietonauha Oy -nimellä aloittanut yhtiö on nykyisin osa Itella-konsernia. Tällä hetkellä yritys työllistää noin 70 työntekijää.

Osallistun työtehtävissäni ParLet -sovelluksen jatkokehitykseen. Kyseessä on Itella Oyj:n tilaama kansainvälisen postiliikenteen rekisteröinti- ja seurantajärjestelmä. Sovellus on toteutettu Java-ohjelmointikielellä, Java EE -määritystä noudattaen. ParLetin liiketoimintakerros noudattaa vuodelta 2001 peräisin olevaa EJB 2.0 -spesifikaatiota, jota voidaan pitää tekniikaltaan vanhentuneena. Uusin EJB -versio on vuonna 2006 ilmestynyt 3.0. ParLet joudutaan tulevaisuudessa, todennäköisesti vuoden 2009 loppuun mennessä, siirtämään uudelle sovelluspalvelimelle. Uusi sovelluspalvelin mahdollistaa uuden komponenttitekniikan käyttöönoton, minkä vuoksi uuteen tekniikkaan perehtyminen on ajankohtaista.

EJB (Enterprise JavaBeans) on komponenttiarkkitehtuuri komponenttipohjaisten hajautettujen sovellusten kehittämiseen ja jakeluun. Spesifikaatiota tukevat sovellukset ovat skaalautuvia, tukevat transaktioita ja mahdollistavat usean yhtäaikaisen käyttäjän palvelun. Kun sovellus on kerran kirjoitettu, se voidaan sijoittaa mille tahansa EJB:tä tukevalle palvelinalustalle. (DeMichiel & Keith, 2001:25.)

EJB 3.0 uudistaa tekniikkaa monin tavoin aiempiin verrattuna. Se yksinkertaistaa ohjelmointitapaa, tukee paremmin Java-pohjaisia standardeja sekä vähentää huomattavasti tarvittavan ohjelmakoodin määrää. Kuvio 1 esittää Depu Pandan suorittaman selvityksen tuloksia koodin määrän vähenemisestä.



Kuvio 1: Koodin ja deskriptorien määrän väheneminen versiomigraation tuloksena. (Panda, Losing weight is easier than you think)

Tämän mittakaavan muutos koodimäärän vähenemisessä parantaisi sovelluksen suorituskykyä ja helpottaisi ohjelmakoodin ylläpitoa huomattavasti.

Tutkin opinnäytetyönäni, mitä tekijöitä EJB 3.0 -tekniikkaan siirtymiseen ParLetissa liittyy. ParLetin kehitysryhmällä ei ollut työtä aloitettaessa omakohtaista kokemusta EJB 3.0 -tekniikasta. Tarkoituksena oli kartoittaa mahdolliseen päivitysprosessiin liittyviä tekijöitä, tuottaa ohjeistusta komponenttien muokkaamiseen sekä esitellä käytettävissä olevia päivitysmalleja ja ongelman rajausmahdollisuuksia. Työn tarkoitus on antaa lukijalle kuva siitä, millainen päivitysprosessi olisi käytännössä. Tämän pohjalta toimeksiantaja osaisi määrittää tulevaisuuden toimenpiteitä entistä paremmin.

Työ aloitettiin tutustumalla tausta-aineistoon. Taustatietoa tarvittiin EJB -kerroksen päivityksestä yleisellä tasolla sekä ParLet-sovelluksen teknisistä ratkaisuperiaatteista. EJB -päivitystä koskevaa olemassaolevaa tietoa haettiin alan kirjallisuudesta. ParLetin toiminnan osalta työssä tarvittava informaatio pyrittiin kartoittamaan asiantuntijoita haastattelemalla, sovelluksen teknistä dokumentaatiota tulkitsemalla ja lähdekooditiedostoja tutkimalla. Olemassa olevaa teoriaa uuteen versioon siirtymisessä käytettävistä menetelmistä pyrittiin tarkastelemaan ParLetiin soveltuvien teorioiden osalta. ParLetin liittyvää aineistoa hankittiin sovelluksen kehitysdokumenttien ja asiantuntijahaastattelujen avulla. Java- ja EJB -tekniikoita käsittelevinä lähteinä käytettiin Sun Microsystemsin spesifikaatioita sekä aiheeseen littyviä kirjoja ja verkkoartikkeleita.

Tutkittuani ParLetin liiketoimintakerroksen tilaa tein yhteenvedon käytettävissä olevista päivitysmenetelmistä. Listasin mahdolliset menetelmät ja arvioin niiden hyviä ja huonoja puolia. Koska EJB 3.0:n syntaksi poikkeaa huomattavasti vanhempien versioiden syntaksista, pystytin uutta versiota käyttävän koeympäristön, johon toin komponentin olemassa olevasta ParLetin kehitysversiosta. Päivitin tämän komponentin uutta tekniikkaa hyödyntävään muotoon, jotta kehittäjäryhmä saisi käytännön esimerkin uuteen tekniikkaan siirtymisestä.

Tämän raportin luvut 2 – 4 esittelevät kohdesovelluksen toimintaa, arkkitehtuuria ja teknistä alustaa. Luvussa 5 tarkastellaan EJB-tekniikan periaatteita ja siirtymistä 2.0-versiosta 3.0-versioon. Luku 6 esittelee suorittamieni selvitysten kulkua ja tuloksia.

Vaikka suurten sovellusten liiketoimintakerrosten päivitykset ovat aina tapauskohtaisia, voidaan tämän tutkimuksen tuloksia pitää suuntaa-antavina myös muiden EJB 2.x -pohjaisten sovellusten päivitystarvetta ja -strategiaa suunnitellessa. Tutkittavat päivitysmenetelmät sekä EJB 3.0 -version uudet ominaisuudet todennäköisesti kiinnostavat useita vanhempia versioita käyttäviä tahoja. Vaikka EJB:n päivitystyöstä on jo kirjoitettu aiempaa teoriaa, auttaa tämän työn puitteissa esiteltävä yksittäinen esimerkkitapaus hahmottamaan erityisesti suurten sovellusten päivittämisessä kohdattavia ongelmia.

2 Kohdejärjestelmä

ParLet on Itella Oyj:n tilaama kansainvälisen postiliikenteen rekisteröinti- ja seurantajärjestelmä. Sovelluksella hoidetaan kaikki ulkomaan postiliikenteen käsittelyyn liittyvät toiminnot. Sitä käytetään muun muassa maahan saapuvien ja maasta lähtevien lähetysten kirjaamiseen, kuljetusreittien suunnitteluun sekä laskutuksessa käytettävien raporttien tulostamiseen. Toiminnallisuus jakautuu lähtevän, saapuvan ja kauttakulkevan postin käsittelyyn sekä ulkomaan postia koskevaan sanomaliikenteeseen. (Haastattelu, 2.5.2008)

Postin käytäntöjen mukaan erikseen määritellyt postilähetykset (paketit, kirjatut kirjeet, vakuutettu posti ja postiennakkolähetykset) kirjataan järjestelmään. Itella Oyj:n liiketoiminnan kannalta ParLet on kriittinen sovellus, sillä ilman toimivaa käsittelyjärjestelmää ulkomaille lähtevä ja kotimaahan saapuva posti eivät etene. Postinkäsittelykeskuksiin saapuu päivässä tuhansia kirjeitä, noin 1500 pakettia sekä 50 – 70 lähetyserää. (Logia Software Oy, 2004) Järjestelmä on käytössä 24 tuntia vuorokaudessa 365 päivänä vuodessa. Sovelluksen käyttäjiä on 113, joista yhtäaikaisia käyttäjiä ajankohdasta riippuen 20 – 30.

ParLetilla on useita eri käyttäjäryhmiä: peruskäyttäjä, ylläpitäjä, pääkäyttäjä, suunnittelija ja tilastointitoimintoja hyödyntävä STAT-käyttäjä. Jokaisella käyttäjällä on pääsy vain omalle käyttäjäryhmälleen sallittuihin toimintoihin. Suurin käyttäryhmä ovat peruskäyttäjät, joita on kolmessa eri lajittelukeskuksessa yhteensä 70. Heidän tehtävänään on esimerkiksi kirjata saapuvat ja lähtevät lähetykset sovelluksen käyttöliittymän avulla tietokantaan. (Logia Software Oy, 2004)

Tiedon säilytys tapahtuu kahdessa eri tietosäiliössä: aktiivi- ja historiatietokannassa. Aktiivikannassa tietoa säilytetään kolmen kuukauden ajan ja historiakannassa 18 kuukautta. Nyrkkisääntönä voidaan pitää, että tietoja ylläpitävät toiminnot suoritetaan vain aktiivikantaan. Historiakannan pääasiallinen tarkoitus on toimia vain luettavana tietovarastona. Kumpaakin tietokantaa kohti suoritettavia toimintoja varten on oma erillinen käyttöliittymänsä.

3 Java EE ohjelmistokehitysalustana

ParLetin kaltaisten yritystason ohjelmistojen kehittäminen olisi työlästä ilman asianmukaisia menetelmiä. Java Platform, Enterprise Edition määrittelee standardin, jota noudattamalla järeiden yritystason sovellusten toteuttamisessa päästään mitä todennäköisimmin haluttuun lopputulokseen.

Tämän luvun tavoitteena on perehtyä käsitteisiin, jotka lukijan tulisi hahmottaa Java EE -spesifikaation osalta opinnäytetyöni tuloksia tulkitakseen. Kokonaisuudessaan spesifikaatio on hyvin laaja, joten keskityn kuvaamaan tekniikkaa etupäässä ParLetin näkökulmasta.

3.1 Java EE:n Tavoitteet

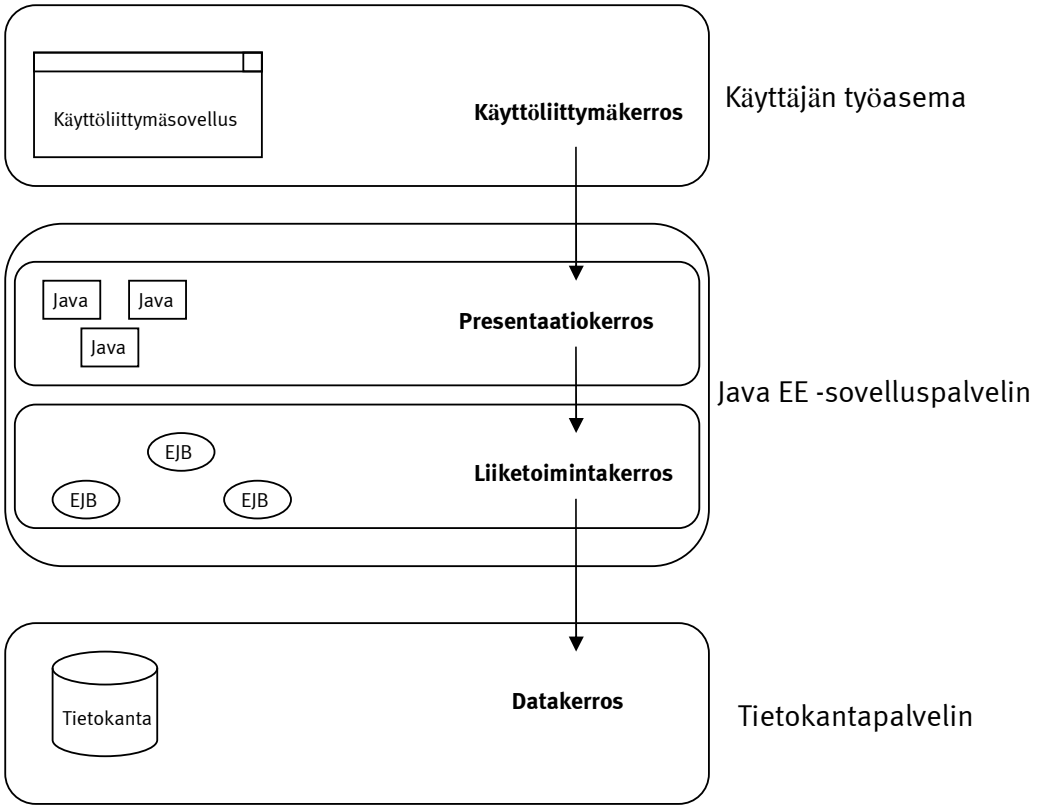
Shannon (2006) erittelee yritystason sähköisille palveluille kolme päävaatimusta: palvelun tulee olla laajalti saavutettavissa, turvallinen ja luotettava sekä skaalautuva. Java EE pyrkii toteuttamaan nämä vaatimukset monitasoisten ohjelmistoarkkitehtuurien avulla. Useiden tasojen ansiosta sovellusten luominen, ylläpitäminen ja uutta käyttötarkoitusta varten laajentaminen pystytään toteuttamaan kohdistetusti ja edullisesti.

Ominaisuudet, jotka tekevät yritystason sovelluksista tehokkaita, tekevät sovelluksista usein myös monimutkaisia. Esimerkiksi sovelluksen tietoturvan ja vakauden varmistaminen vaatii paljon infrastruktuurista ohjelmakoodia. Java EE -ohjelmistokehitysalusta on suunniteltu vähentämään kehitystyön monimutkaisuutta tarjoamalla standardin ohjelmointimallin sekä ajonaikaisen ympäristön, joten ohjelmoijat voivat keskittyä sovelluksen varsinaisen toiminnallisuuden kehittämiseen. (Sun Microsystems, 2006b: 12)

3.2 Monikerrosarkkitehtuuri

ParLet noudattaa tyypillisen Java EE -sovelluksen tavoin monikerrosarkkitehtuuria (multi-tier architecture). Sovelluksen toiminnallisuus on jaettu eristettyihin toiminnallisiin osiin, kerroksiin. Usein monikerroksiset sovellukset sisältävät asiakas-, väli- ja datakerroksen. Asiakaskerros koostuu asiakasohjelmasta, joka muodostaa kutsuja välikerrokseen. Välikerroksen liiketoimintafunktiot käsittelevät käyttäjien kutsuja ja prosessoivat sovellusdataa, tallettaen sen lopulta datakerrokseen pysyvää säilytystä varten. (Sun Microsystems, 2006b: 12).

Kuvio 2 esittelee tyypillistä monikerrosarkkitehtuuria, jossa sovellus on hajautettu fyysisesti kolmelle eri laitteelle. Kuvatussa esimerkkitapauksessa presentaatiokerros huolehtii käyttöliittymäkoodin tuottamisesta, liiketoimintakerros suorittaa varsinaiset liiketoimintaoperaatiot ja datakerros toimii pysyvänä tietovarastona.



Kuvio 2: Tyypillinen monikerrosarkkitehtuuri havainnollistettuna

3.3 Ohjelmistokomponentit

Ohjelmistokomponentti (application component) -nimitystä käytetään ohjelmistokehityksessä ilmaisemaan sovelluksen itsenäistä osaa, joka suorittaa jonkin yksittäisen toiminnon ja kommunikoi toisten komponenttien kanssa. Java EE -spesifikaatio määrittelee joukon erilaisia ohjelmistokomponentteja, joiden määrityksiä Java EE -sovelluksen tulee noudattaa. ParLet käyttää näistä useita eri komponenttityyppejä, kuten jo mainittuja Enterprise JavaBeans (EJB) -komponentteja. Tyypillisesti EJB-komponentit sisältävät Java EE -sovelluksen varsinaisen liiketoimintalogiikan. (Bill Shannon, 2006: 7).

3.4 Sovelluspalvelimet

Java EE -sovellus tarvitsee toimiakseen ajonaikaisen ympäristön. Siinä missä web-palvelin tarjoaa käyttäjilleen web-sivuja, **sovelluspalvelimen** voidaan katsoa tarjoavan sovelluksia.

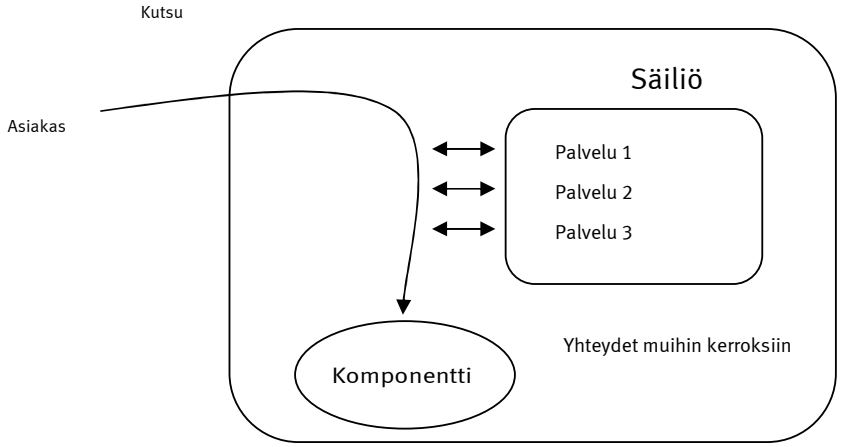
Java EE -sovelluspalvelin on palvelinsovellus, joka tarjoaa asiakasohjelmalle Java EE -alustan ohjelmointirajapinnat ja palvelut. Sovelluspalvelin toimii ajoympäristönä eri ohjelmistokomponenttityypeille, jotka vastaavat aina tiettyä kerrosta monikerrosarkkitehtuurin mukaisessa sovelluksessa. Palvelin tarjoaa komponenteille suunnatut palvelut säiliöiden kautta. (Sun Microsystems, 2006b: 15)

3.5 Säiliöt

Java EE -sovelluskehitysalusta tukee monikerroksista arkkitehtuuria säiliöiden (container) avulla. Nämä huolehtivat vaadittujen palvelujen tarjoamisesta ohjelmistokomponenteille (Shannon 2006: 5).

Säiliö toimii rajapintana komponentin ja Java EE -alustan tarjoaman alemman tason toiminnallisuuden kanssa (Sun Microsystems, 2006b: 15). Komponentit eivät koskaan ole yhteydessä suoraan toisiinsa, vaan käyttävät säiliön tarjoamia protokollia ja metodeja kommunikoidessaan muiden komponenttien tai Java EE -alustan tarjoamien palveluiden kanssa. Kaikille komponenttityypeille on olemassa oma säiliönsä. (Shannon 2006: 8) Esimerkiksi EJB-säiliö toimii rajapintana EJB komponenttien ja Java EE -sovelluspalvelimen välillä. Se sijaitsee fyysisesti sovelluspalvelimella, ja huolehtii sovelluksen EJB-komponenttien suorituksesta.

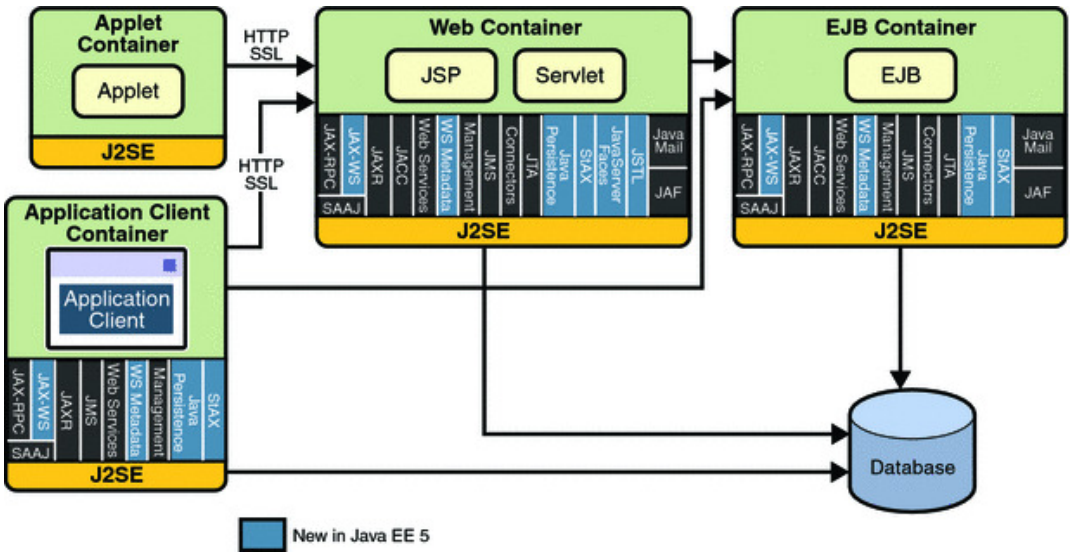
Kuvio 3 sittelee säiliön rakennetta yleisellä tasolla. Asiakas kutsuu komponenttia säiliön kautta. Säiliö huolehtii tarvittavien palvelujen suorittamisesta ja komponentin mahdollisesti muihin kerroksiin suorittamien kutsujen edelleen ohjaamisesta.



Kuvio 3: Säiliön rakenne

3.6 Ohjelmointirajapinnat

Java EE tarjoaa sovelluskehittäjän käyttöön useita ohjelmointirajapintoja (Application Programming Interface / API). Näiden rajapintojen tarjoamat palvelut kattavat kaikki yleisimmät yritystason sovelluksissa käytettävät ominaisuudet. Kuvio 4 esittelee Java EE:n ohjelmointirajapinnat säiliöittäin jaoteltuina.

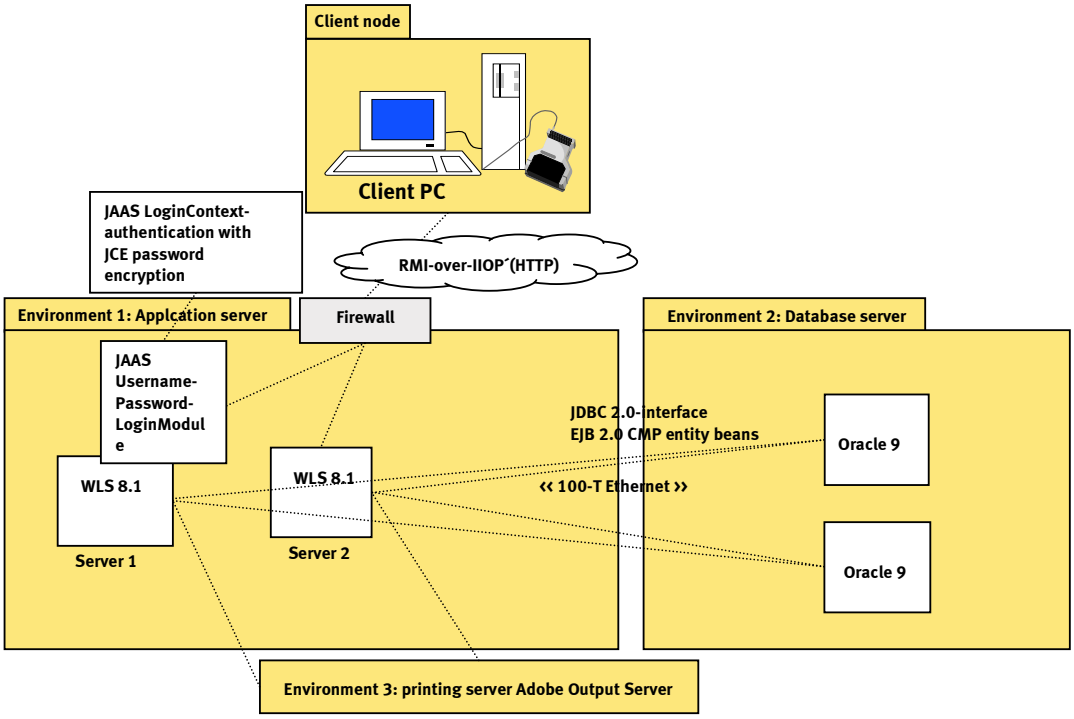


Kuvio 4: Java EE:n tarjoamat ohjelmointirajapinnat säiliöittäin (Sun Microsystems, 2007)

Suurin osa rajapinnoista sijoittuu EJB- ja Web-säiliöihin. Luonteeltaan näiden rajapintojen tarjoamat palvelukokonaisuudet ovat tyypillisesti tiedon siirtoon ja hallinnointiin liittyviä. ParLet käyttää hyväkseen pääasiassa EJB-säiliötä. Muita sovelluksessa hyödynnettäviä rajapintoja ovat Java Message Service (JMS), Java Naming and Directory Interface (JNDI) ja Java Persistence Api (JPA). Näistä viimeksi mainittuun syvennytään tarkemmin EJB:n yhteydessä.

4 ParLetin sovellusarkkitehtuuri

ParLet on toteutettu Java EE -spesifikaation mukaisen sovellusalustan päälle. Noin 1900 Java-luokkaa käsittävä liiketoimintakerros sijaitsee erillisellä sovelluspalvelimella. Tämä kerros on yhteydessä omilla palvelimillaan toimiviin tietokantoihin. Käyttäjä lataa työasemalleen internetin välityksellä Java Swing -pohjaisen käyttöliittymäsovelluksen. Käyttäjän kutsut ohjautuvat käyttöliittymäkerroksesta verkon yli liiketoimintakerroksen EJB -säiliölle, joka asianmukaiset toimenpiteet suoritettuaan huolehtii mahdollisesti edelleen tietokantapalvelimelle suoritettavista kutsuista. Kuvio 5 esittelee sovelluksen fyysistä sijoittelua verkkoympäristöön.

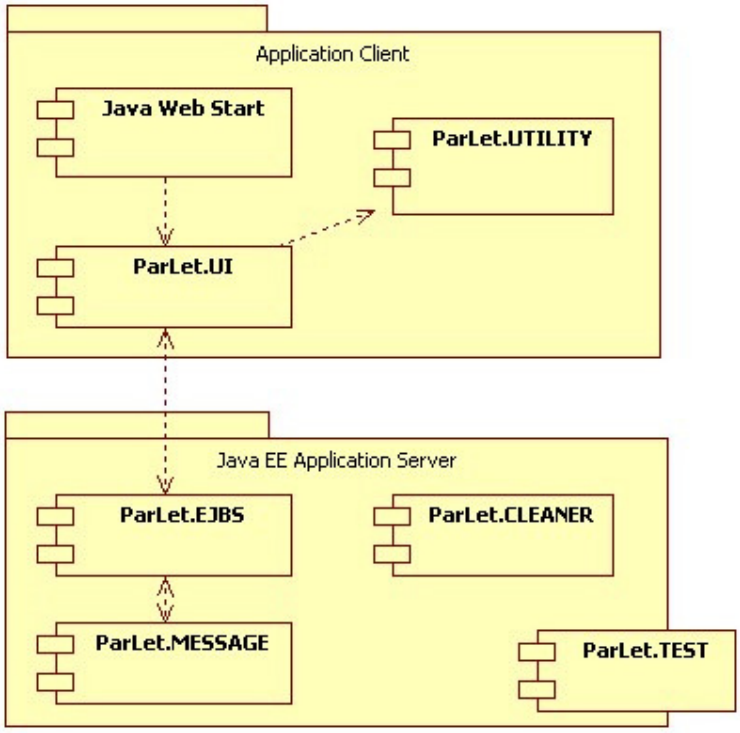


Kuvio 5: ParLetin sijoittelukaavio (Logia Software, 2006)

Sovelluspalvelimena toimii BEA WebLogic Application Server 8.1. Sekä aktiivi- että historiatietokannat ovat Oraclen tietokantoja.

ParLetin moduulit

ParLetin ohjelmakoodi jakautuu useisiin **moduuleihin**, joista osa sijoittuu käyttöliittymäkerrokseen ja osa liiketoimintakerrokseen. Kullakin moduulilla on itsenäinen tehtävä sovelluskokonaisuudessa. Kuvio 6 esittelee eri moduulien loogista sijoittumista.



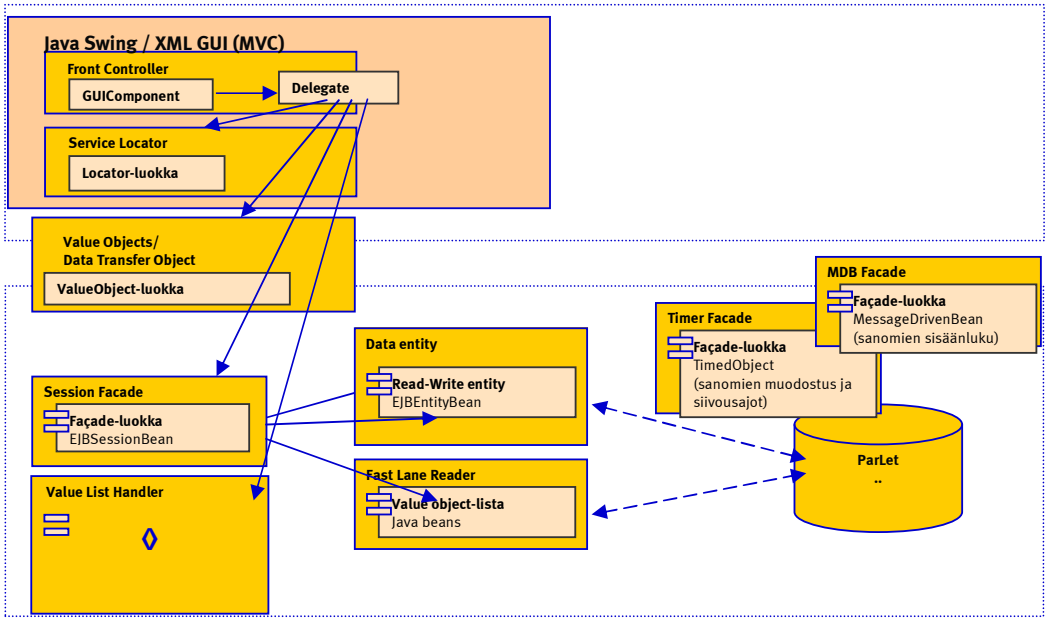
Kuvio 6: ParLetin moduulit komponenttikaaviona.

Käyttöliittymämoduulit suorittavat käyttäjän interaktioihin liittyviä toimintoja, sovelluspalvelimella sijaitsevien moduulien huolehtiessa sovelluksen liiketoimintaoperaatioista. `ParLet.UI` -pakkaus sisältää kaikki käyttöliittymään liittyvät toiminnot, kuten esimerkiksi grafiikkakomponenttien piirtämisen ja käyttäjän käskyjen poimimisen. Käyttöliittymässä käytetään XML GUI -nimistä avoimen lähdekoodin kirjastoa. Käyttöliittymäkomponentit ja niiden graafinen sijoittelu kuvataan XML -merkinnöillä, mikä helpottaa koodin ylläpidettävyyttä.

ParLetissa käytettyjä suunnittelu- ja arkkitehtuurimalleja

Suunnittelumalli (design pattern) tarkoittaa ohjelmistokehityksessä suunnitteluvaiheessa usein esiintyvän ongelman ratkaisuun kehitettyä uudellenkäytettävää mallia. Se ei tarjoa valmista ohjelmakoodia eikä pureudu algoritmitasolle, vaan toimii pikemminkin kuvauksena tai suuntaviivana rakenteelle, jota sovelluksen suunnitteluvaiheessa noudattamalla päästään haluttuun lopputulokseen. **Arkkitehtuurimallit** (architectural patterns) keskittyvät laajemman mittakaavan tekijöihin. Niitä voidaan pitää eräänlaisina kokonaisuutta palvelevina arkkitehtuurin palasina.

Kuvio 7 esittää ParLetissa käytettyjä suunnittelu- ja arkkitehtuurimalleja. Sen avulla voidaan hahmottaa sovelluksen teknisiä ratkaisuja.



Kuvio 7: ParLetissa käytettäviä suunnittelumalleja (Logia Software, 2006)

Käyttöliittymän rakenne noudattaa MVC (Model-View-Controller) -arkkitehtuurimallia, joka mahdollistaa käyttöliittymäkoodin, liiketoimintalogiikan ja datan käsittelylogiikan erottamisen toisistaan.

Service Locator -suunnittelumallin avulla toteutetaan EJB-liiketoimintakomponenttien haut sekä parannetaan suorituskykyä tarvittavilla välimuistiratkaisuilla. **Tiedonsiirto-olioita** (Data Transfer Object) hyödynnetään tiedon siirrossa Java-client-kerroksen sekä

EJB-komponenttien välillä. Luokat edustavat ParLet-järjestelmän liiketoimintakäsitteitä ja niiden välisiä suhteita.

Istuntofasadi (Session Facade) -mallin avulla toteutetaan kaikki tiedon käsittelyrutiinit (tallennus, ylläpito, tiedustelut). Istuntokomponentit kontrolloivat järjestelmän liiketoimintatransaktioita. **Ajastinfasadi** (Timer Facade) huolehtii ajastettujen toimintojen suorittamisesta sanomien avulla.

5 Enterprise JavaBeans

Java EE -sovellusten kehittäjät törmäävät työssään monesti tiettyihin monimutkaisia tietorakenteita edellyttäviin infrastruktuurisiin ongelmiin. EJB on komponenttiarkkitehtuuri, jonka perimmäinen tarkoitus on tarjota sovelluskehitystä helpottava standardi palvelinpuolen komponenttimalli hajautetuille sovelluksille. Samalla se tarjoaa valmiit toteutukset joidenkin monimutkaisten matalan tason toiminnallisuuksien - esimerkiksi transaktioiden ja tietoturvan - toteuttamiseen, jolloin kehittäjä voi keskittyä suuremmissa määrin varsinaisen liiketoimintalogiikan toteuttamiseen.

EJB-spesifikaatio kehitettiin IBM:llä vuonna 1997. Sun Microsystems adoptoi EJB 1.0- ja 1.1 -versiot itselleen ja kehitti edelleen 2.0 (JSR 19) ja 2.1 (JSR 153) -versiot vuosina 2001 ja 2003. Uusin 3.0-versio (JSR 220) on julkaistu vuonna 2006. (The Java Community Process (SM) Program, 2008.) Aiempien versioiden keskinäisiä muutoksia voidaan pitää pienehköinä, mutta 3.0-versio tuo tekniikkaan runsaasti uudistuksia.

Tämän luvun tavoitteena on esitellä EJB pähkinänkuoressa sekä erityisesti paneutua uusimman 3.0 -version tuomiin muutoksiin ParLetissa käytettävään 2.0 -versioon nähden.

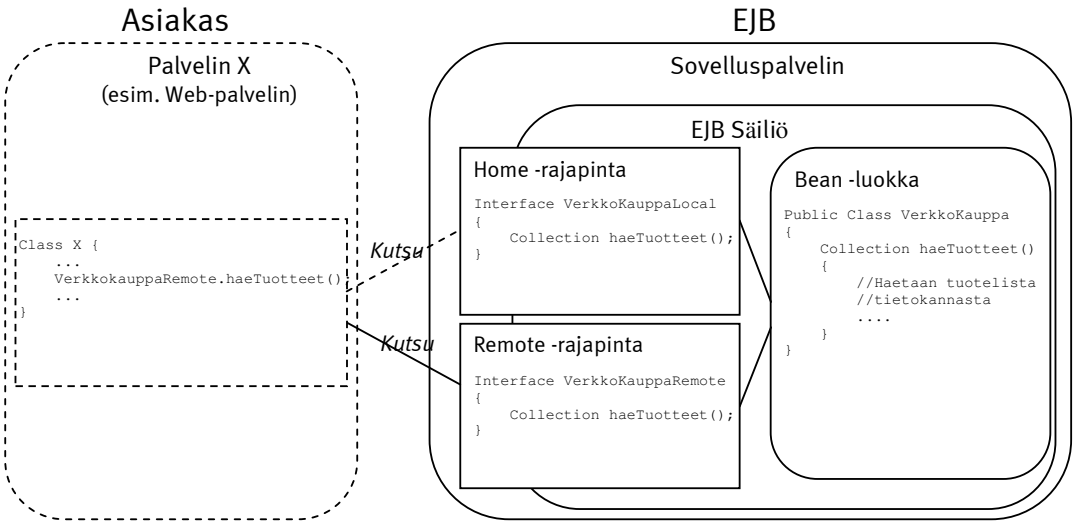
5.1 Arkkitehtuuri

EJB-komponentti (engl. enterprise bean) on sovelluspalvelimella omassa säiliössään toimiva komponentti, joka kätkee sisälleen palasen sovelluksen liiketoimintalogiikkaa. Liiketoimintalogiikka muodostuu ohjelmakoodista, joka suorittaa ohjelman varsinaisen toiminnallisuuden. (Sun Microsystems, 2006a) Esimerkiksi verkkokauppasovelluksen EJB -komponentit voisivat sisältää liiketoimintafunktioita kuten `haeTuotteet()`, `muodostaTilaus()` sekä `tarkistaTuotteenSaatavuus()`. EJB-arkkitehtuurissa liiketoimintafunktioiden rungot sijaitsevat bean-luokassa, mutta niitä käyttävä asiakas kutsuu niitä komponenttirajapintojen kautta.

Kuvio 8 havainnollistaa EJB:n arkkitehtuuria. Varsinainen EJB-komponentti muodostuu tässä tapauksessa local- ja remote-rajapinnoista sekä bean-luokasta. Verkkokaupan tapauksessa asiakas-luokka voisi olla esimerkiksi web-palvelimella sijaitseva java-luokka, joka huolehtii HTML-komponenttien muodostamisesta ja

käyttäjän käskyjen välittämisestä. Kun käyttäjä pyytää käyttöliittymän välityksellä nähdäkseen tuotelistan, asiakasluokka suorittaa `haeTuotteet()` -funktion kutsun sovelluspalvelimelle. Jos asiakasluokka ja EJB-säiliö sijaitsevat fyysisesti samassa virtuaalisessa Java-ympäristössä, metodia kutsutaan local -rajapinnan kautta. Näin ollen samassa säiliössä pyörivät muut EJB-komponentit voivat kutsua funktiota local-rajapinnan kautta. Jos taas kutsu suoritetaan esimerkiksi verkon yli palvelimelta toiselle, joudutaan käyttämään remote-rajapintaa.

Kuvion 8 osoittamassa esimerkkitapauksessa Bean-luokan `haeTuotteet()` -metodi suorittaisi tuotehaun tietokantaan. Komponentti voisi tehdä tämän hyödyntäen EJB-säiliön tarjoamia palveluja, kuten esimerkiksi Java Transaction API:a tai Java Persistence API:a.



Kuvio 8: EJB:n arkkitehtuuri yleisellä tasolla.

EJB:n komponenttimalli sisältää useita eri tavoin toimivia oliokategorioita, jotka poikkeavat toisistaan erityisesti kutsuntatavan ja tilallisuuden osalta. Sun Microsystems jakaa EJB 2.0 -spesifikaatiossaan (2001:43) oliotyypit seuraavasti:

- Tilattomia palveluja tarjoava olio
- Asynkronisesti JMS-sanomien avulla suoritettava, tilattomia palveluja tarjoava olio
- Keskustelun mahdollistavaa istuntodataa tallettava olio, joka säilyttää tilansa asiakkaan kutsujen ajan
- Liiketoimintaoliota edustava entiteettiolio, joka on yhteinen kaikille asiakkaille

Näiden oliokategorioiden toteuttamista varten EJB tarjoaa erilaisia komponenttityyppejä: sessiokomponentti, entiteettikomponentti ja sanomaohjattu komponentti. Näiden komponenttien toteutustavoissa esiintyy kuitenkin joitain olennaisia versiokohtaisia eroja EJB 2.0- ja 3.0 -versioiden välillä, joten seuraavaksi on aiheellista tarkastella näitä omina kokonaisuuksinaan.

5.2 EJB 2.0

ParLetissa käytettävä EJB 2.0 –arkkitehtuuri sisältää kolme eri komponenttityyppiä: sessiokomponentti, entiteettikomponentti ja sanomaohjattu komponentti. Sessiokomponentteja on kahdenlaisia: tilallisia (Stateful bean) ja tilattomia (Stateless bean). Sanomien avulla toimivaa komponenttityyppiä edustaa sanomaohjattu komponentti (Message-driven bean). Liiketoimintamallia edustava entiteettikerros toteutetaan entiteettikomponenteilla (Entity beans).

Sessiokomponentteja käytetään suorittamaan jokin yksittäinen liiketoimintaoperaatio, esimerkiksi tuotteen haku tietokannasta. Tilaton komponentti ei nimensä mukaisesti säilytä tilaansa asiakasluokan kutsujen välillä, joten se sopii hyvin yksittäisiin hakuihin. Jos komponentin tulee säilyttää itseensä dataa, kuten esimerkiksi ostoskorikomponentin tapauksessa, tulee käyttää tilallista sessiokomponenttia. Sessiokomponentteja voidaan myös käyttää istuntofasadina.

Sanomaohjattuja komponentteja voidaan kutsua sessiokomponenteista poiketen asynkronisesti eli ei-reaaliaikaisesti. Ne käyttävät Java Messaging Service (JMS) –ohjelmointirajapintaa sanomaliikenteen välitykseen. Komponentit toimivat yksinkertaisina tapahtumankäsittelijöinä, jotka kuuntelevat sanomajonoja. EJB-säiliö tarkkailee JMS-jonojen tilaa ja huolehtii sanomaohjattujen komponenttien suorittamisesta. Tätä tekniikkaa käytetään esimerkiksi sähköpostipalvelimissa.

EJB 2.0 toteuttaa entiteettikerroksen entiteettikomponenteilla. Entiteettikomponentti edustaa yksittäistä liiketoimintamallin osaa, esimerkiksi yksittäistä tietokantataulua. Komponentti pitää sisällään tietojäsentasolla tietokantaan kohdistuvat toiminnot, kuten datan päivityksen ja tallennuksen. Entiteettikomponenteilla voidaan siis toteuttaa sovelluksen koko tietomalli Java-koodin avulla, jolloin tietokannan data on helposti kartoitettavissa ohjelmakoodiin. Tietokantaan kohdistuvat operaatiot toteutetaan **EJB Query Language** (EJB QL) -kielellä, joka muistuttaa syntaksiltaan monille tuttua SQL -kieltä, mutta on tietokanta-alustariipumaton.

EJB 2.0 -versiossa komponenttien tulee periytyä komponenttityyppiä vastaavasta yläluokasta. Näin ollen niiden tulee myös toteuttaa tiettyjä olion elinkaareen liittyviä toimintoja. Kuvio 9 esittää yksinkertaista sessiokomponenttia. Komponentilla on EJBHome-luokasta periytyvä kotirajapinta (`Interface VerkkokauppaHome`), jossa luodaan viittaus komponenttiin. Varsinainen liiketoimintarajapinta (`Interface Verkkokauppa`) periytyy EJBObject-luokasta. Liiketoimintarajapinnan esittämän metodin toteutus löytyy komponenttiluokasta (`VerkkokauppaBean`). Tämä luokka periytyy SessionBean-luokasta, joten sen tulee toteuttaa useita yläluokassa määriteltyjä metodeja kuten `ejbCreate()`, `ejbActivate()`, `ejbPassivate()` ja `ejbRemove()`. Näitä metodeja tarvitaan komponentin elinkaaren hallinnointiin.

```
Public Interface VerkkokauppaHome
    extends EJBHome {
    public Verkkokauppa create()
        throws CreateException,
            RemoteException;
}
```

```
Public Interface Verkkokauppa
    extends EJBObject {
    Collection haeTuotteet()
        throws
RemoteException;
```

```
Public Class VerkkokauppaBean extends SessionBean {

    Collection haeTuotteet() throws RemoteException {
        ....
    }

    public void ejbCreate(){}

    public void ejbActivate(){}

    public void ejbPassivate(){}

    public void ejbRemove(){}

    public void setSessionContext(SessionContext context){}
```

Kuvio 9: EJB 2.0 -komponentin koodilistaus.

Käytönkuvaimet

EJB 2.0 -versiossa ei voida pelkän Java-koodin avulla määritellä kaikkia komponentin ominaisuuksia. Komponentin toteuttajan tulee luoda erillisiä **käytönkuvaimia** (deployment descriptor) määrittääkseen komponentille kaiken EJB-säiliön tarvitseman informaation. Käytönkuvaimet ovat XML-tiedostoja, joiden syntaksi noudattaa EJB-spesifikaation määrittelemiä sääntöjä. EJB 2.0 pohjautuu vahvasti XML-pohjaisten kuvaimien käyttöön, joten XML-tiedostojen määrä voi pienessäkin sovelluksessa kasvaa suureksi.

Käytönkuvaimilla määritetään kahdenlaista informaatiota: komponenttien rakenteeseen sekä sovelluksen koostamiseen liittyvää tietoa. Rakenteellinen tieto käsittää komponentin nimen, Java-luokan, koti- ja etärajapinnat, komponentin tyypin, ympäristömuuttujat, viittaukset muihin komponentteihin sekä tietoturvarooliviittaukset. Rakenteellinen informaatio on pakollista komponentin toiminnan kannalta. Sovelluksen koostamiseen liittyvää tietoa ovat esimerkiksi sovellukseen kuuluvien komponenttien määrittäminen, tietoturvaroolien esittely sekä säiliön transaktio-asetukset. Toisin kuin rakenteellista tietoa sisältävät kuvaimet, koostamiseen liittyvät kuvaimet eivät ole pakollisia.

EJBGen työmäärän vähentäjänä

EJBGen on Cedric Ceustin BEA WebLogic -sovelluspalvelimille kehittämä koodigeneraattori EJB 2.0 -spesifikaatiota noudattavaa ohjelmakoodia varten. Sen sijaan että sovelluskehittäjä joutuisi muokkaamaan useaa tiedostoa (komponenttiluokka, etä-, lähi- ja kotirajapinnat, käytönkuvain) samanaikaisesti, EJBGenin avulla ohjelmakoodin muokkaus voidaan keskittää pelkästään komponenttiluokkaan. Luokka, muuttujat ja metodit annotoidaan JavaDoc -tageilla, joiden perusteella EJBGen parseroi koodin ja tuottaa tarvittavat tiedostot (EJBGen, 2004). ParLetin kehitystyössä hyödynnetään kyseistä työkalua lähes jokaisessa komponentissa.

EJBGen helpottaa huomattavasti EJB 2.0 -komponenttien kehitystyötä, sillä se vapauttaa ohjelmoijan täysin käytönkuvaimien ja komponenttirajapintojen käsittelystä. EJB 3.0 -spesifikaatiossa työkalua ei kuitenkaan enää ohjelmointimallin yksinkertaistumisen vuoksi tueta. (Programming WebLogic Enterprise JavaBeans: Version 3.0:2-3. 2008).

5.3 EJB 3.0 tekniikan uudistajana

Vaikka EJB oli tehokas ja käytännöllinen tekniikka jo ennen 3.0 -versiota, ohjelmointitapa vanhemmissa versioissa on monimutkainen ja hämmentävä, sillä yksinkertaisimmankin EJB-komponentin luominen edellyttää useiden Java-luokkien ja käytönkuvaimien luomista. Tämä haittasi EJB -tekniikan laajempaa omaksumista kehittäjäpiireissä. (Programming WebLogic Enterprise JavaBeans, 2007: 2-1.)

EJB 3.0 -spesifikaation päätavoite on helpottaa ohjelmointia, tehdä tekniikasta vähemmän alustariippuva, korvata ulkopuoliset käytönkuvaimet ohjelmakoodiin

upotettavilla metatieto-annotaatioilla ja vähentää komponenttirajapintojen määrää. Täten sovelluskehittäjien työ nopeutuisi sovelluksen samalla keventyessä fyysisesti.

Yksi EJB 3.0 -version päätavoitteista on Persistence API-sovelluskehyksen standardointi sekä entiteettikomponentin ohjelmointitavan ja olio-relaatio (O/R) -kartoitusmallin yksinkertaistaminen (Programming WebLogic Enterprise JavaBeans, 2007: 2-1). Persistence-sovelluskehys huolehtii tietojen säilytyksestä tietokannassa olioparadigman mukaisten riippuvuussuhteiden mukaisesti.

Muutoksia ohjelmointitavassa

EJB 3.0 -version komponenttityyppejä on uudistettu joiltakin osin. Sessiokomponentteja ja sanomaohjattuja komponentteja käytetään edelleen, mutta entiteettikomponentit on korvattu täysin Java Persistence API:n entiteeteillä. Näin ollen varsinaisia EJB-komponenttityyppejä on vain kolme: tilaton sessiokomponentti, tilallinen sessiokomponentti sekä sanomaohjattu komponentti. Näiden toimintaperiaate on ennallaan, mutta rakenne on yksinkertaistunut huomattavasti arkkitehtuuriuudistuksen myötä.

Kuvio 10 esittelee EJB 3.0 -listauksena kuviossa 9 esitellyn komponentin. EJB 2.0 -versiossa komponenttiin luotiin viittaus kotirajapinnan (home interface) avulla. Nyt kotirajapintaa ei enää tarvita, vaan viittaus komponenttiin syntyy perinteisen olion tapaan. Kuviota tarkastelemalla voimmekin havaita, että rajapinnat ja luokat eivät enää periydy EJB-sidonnaisista yläluokista, vaan ovat tavallisia Java-olioita, **POJO**:ja (Plain Old Java Object). Tämä mahdollistaa komponenttien testaamisen EJB-säiliön ulkopuolella.

EJB 2.x -versioiden käytönkuvaimien runsaus aiheutti päänvaivaa ohjelmoijalle. EJB 3.0:n myötä käytönkuvainten käyttö siirtyy suurilta osin historiaan. Aiemmin käytönkuvaimissa määritellyt ominaisuudet määritellään tästä lähtien suoraan Java-koodiin **annotaatioiden** avulla. Annotaatiot toimivat eräänlaisina leimoina, joilla luokkiin, rajapintoihin ja metodeihin voidaan määritellä lisäominaisuuksia. Esimerkiksi kuviossa 10 luokka VerkkokauppaBean on annotoitu tilattomaksi komponentiksi `@Stateless` -annotaatiolla. Annotaatioita tuetaan Javan 5.0 -versioista lähtien.

```
@Local
Public Interface VerkkokauppaLocal
{
    Collection haeTuotteet();
}
```

```
@Remote
Public Interface Verkkokauppa
{
    Collection haeTuotteet();
}
```

```
@Stateless
Public Class VerkkokauppaBean
{

    Collection haeTuotteet() {
        ....
    }

'
```

Kuvio 10: EJB 3.0 –komponentin koodilistaus.

Aivan täysin käytönkuvaimista ei kuitenkaan voida luopua. Sovelluksen koostamiseen liittyvän tiedon määrittelyyn käytetään yhä suurilta osin XML-tiedostoja. Sovellustasolla ajatellen tarvittavien XML-tiedostojen määrä laskee silti murto-osaan aiemmasta. Ohjelmoija voi edelleen halutessaan käyttää annotaatioiden sijaan käytönkuvaimia. Jos komponentin määrittelyssä on käytetty yhtäaikaisesti sekä annotaatioita että käytönkuvaimia, viimeksi mainittu jää voimaan.

5.4 Java Persistence API

EJB 3.0:n myötä entiteettikomponentit korvataan Java Persistence API -komponenteilla. Kyseisen tekniikan avulla voidaan konvertoida olioparadigman mukaisia rakenteita suoraan tietokantaan pysyväissäilytystä varten. Täten liiketoimintalogiikan ohjelmoija välttyy tietokantakohtaisten kyselykielten käytöltä ja siirtää vastuun datan noutamisesta ja tallentamisesta EJB-säiliölle. JPA-komponentit käyttävät EJB-komponenttien tavoin annotaatioita, joten komponentteja voidaan ajaa myös EJB-säiliön ulkopuolella.

Useimmiten yksittäinen JPA-komponentti vastaa yksittäistä tietokannan taulua, ja edelleen yksittäinen JPA-olio yksittäistä tietokantataulun riviä. Komponentti on tavallinen Java-luokka, joka on annotoitu entiteetiksi `@Entity` -komponentilla. Luokan tietojäsenet vastaavat tietokantataulun sarakkeita. Luokalle voidaan annotaatioiden avulla määrittää perusavaimena toimiva tietojäsen sekä yksittäisten tietojäsenten relaatiot tietokannan muihin tauluihin. Hakujen suorittamisessa käytetään EJB 2.0:n tapaan EJB QL-kieltä, mutta kieltä on päivitetty uusilla ominaisuuksilla. Uusi on versio on silti taaksepäin yhteensopiva vanhan version kanssa.

5.5 Komponenttien muuttaminen EJB 2.0 -versiosta EJB 3.0 -versioon

Komponenttien muuttaminen EJB 2.0 -syntaksista EJB 3.0 -syntaksiin vaihtelee komponenttityypeittäin. Sessiokomponenttien ja sanomaohjattujen komponenttien muuttaminen on melko suoraviivaista, mutta entiteettikomponenttien siirto JPA-komponenteiksi on huomattavasti työläämpi operaatio.

Panda, Rahman & Lane (2007:506) määrittelevät sessiokomponentin muuttamisen muuttamiseen tarvittavat askeleet seuraavasti:

1. EJB-komponentista tehdään POJO:ja karsimalla natiivien EJB-rajapintojen toteutukset

2. Määritellään komponentille ainakin yksi liiketoimintarajapinta

3. Käytönkuvaimien XML-koodi korvataan annotaatioilla

Natiivien EJB-rajapintojen implementoinnit poistetaan sekä liiketoimintarajapinnasta että varsinaisista komponenttiluokista. Komponentin liiketoimintarajapinta määritellään annotoimalla rajapinta `@Local` tai `@Remote` -annotaatiolla. Olemassa olevien käytönkuvainten koodi tulee käydä läpi ja siirtää määritykset Java-koodiin annotaatioiden avulla. Komponenttien elinkaareen liittyvät metodit voidaan korvata tavallisilla, elinkaariannotaatioilla varustuteilla metodeilla.

Sanomaohjattujen komponenttien muuttaminen on myös hyvin suoraviivaista. Natiivien EJB-rajapintojen implementoinnit poistetaan ja komponenttiluokat annotoidaan `@MessageDriven` -annotaatiolla.

EJB 2.0 entiteettikomponenttien muuttaminen Java Persistence API:n mukaiseksi on sen sijaan vaikeampi tehtävä. Panda, Rahman & Lane (2007:514) varoittavat, että arkkitehtuuriltaan huonosti suunnitellun entiteettikerroksen muuttaminen voi vaatia koko kerroksen uudelleensuunnittelua. Jos kerros on kuitenkin suunniteltu huolellisesti asianmukaisia suunnittelumalleja käyttäen, prosessi helpottuu huomattavasti. Kirjoittajat mainitsevat tässä yhteydessä erityisesti istuntofasadien ja tiedonsiirto-olioiden suotuisan vaikutuksen. Molempia suunnittelumalleja käytetään ParLetissa.

5.6 EJB 2 ja EJB 3 samassa sovelluksessa

EJB 3.0 -spesifikaatio vaatii täyttä tukea myös EJB 2.x -pohjaisille komponenteille. Näin ollen 3.0 -versiota noudattavat komponentit voisivat kutsua 2.x -version komponentteja ja päin vastoin. Panda, Rahman & Lane (2007: 500) varoittavat kuitenkin, että vanhempia spesifikaatioita noudattavia sovelluksia ei todennäköisesti voida siirtää täysin ilman muutostoimenpiteitä uudelle EJB 3.0 -sovelluspalvelimelle.

Eri versiota edustavien komponenttien toimimista samassa sovelluksessa on pidetty siinä määrin tärkeänä, että komponenttien yhteensopivuudesta on julkaistu oma lukunsa EJB 3.0 -spesifikaatiossa. EJB 2.x- ja EJB 3.0-versioiset komponentit voivat kutsua toisiaan ongelmitta.

Käytännössä hyvä yhteensopivuus mahdollistaa sen, että EJB 2.x -spesifikaatiota noudattava sovellus voitaisiin päivittää vain osittain, tiettyjen komponenttien osalta EJB 3.0 -version mukaiseksi. Tällä tavoin voidaan priorisoida päivitystarvetta ja harkita erilaisia etenemistapoja sovelluksen migraatiossa. Osittaisissa päivityksissä on kuitenkin huonot puolensa. Tiwari (2006) toteaa artikkelissaan ”Migrating EJB 2.x applications to EJB 3.0” seuraavasti:

> *Koska painopiste on tällä hetkellä yhteensopivuudessa ja päivityksen helppoudessa, nyt saattaa olla paras aika päivittää sovellukset kokonaan EJB 3.0 -versioon. Tulevaisuudessa EJB -spesifikaation jatkaessa kehittymistään, vanhempia versioita (2.1. ja aiemmat) noudattavien sovellusten muuttaminen uudempien spesifikaatioiden mukaiseksi saattaa muodostua yhä haasteellisemmaksi.*

Versioiden keskinäinen yhteensopivuus luo kuitenkin paljon uusia mahdollisuuksia. ParLetin osalta näihin mahdollisuuksiin palataan seuraavan luvun loppupuolella.

6 EJB 2.0 –version vaihto EJB 3.0 –versioon ParLetissa

Jotta voitaisiin tarkemmin arvioida komponenttikerroksen migraatiota ParLetin osalta, tarvitaan lisää tietoa päivitysprosessin luonteesta ja mahdollisista etenemistavoista.

Aluksi kartoitettiin eri keinoin liiketoimintakerroksen tilaa. Muutostarpeen laajuuden selvittämiseksi ja kohteen rajaamiseksi laadittiin lista päivitystä vaativista EJB -komponenteista. Seuraavaksi käytiin läpi mahdollisen päivitystyön vaiheita, etenemistapoja, vaikutuksia sekä mahdollisia ongelmia. Osana opinnäytetyötä toteutin koeympäristön, jossa muutin erään olemassaolevan ParLetin EJB 2.0 -komponentin EJB 3.0 -versiota ja Java Persistence API:a käyttävään muotoon. Luvun loppupuoliskossa esitellään koeympäristön rakennusprosessin tavoitteet, eteneminen ja tulokset.

6.1 Yleistä

ParLetin ajoympäristönä toimii BEA WebLogic Server 8.1 -sovelluspalvelin, joka ei tue EJB 3.0 -versiota. BEA lopettaa kuitenkin kyseisen palvelinversion teknisen tuen vuoden 2009 loppussa, johon mennessä ParLetin kehityksessä ja tuotannossa tulisi ottaa käyttöön uudempi versio. Sovelluspalvelimen uudemmat versiot (9.0:sta ylöspäin) tukevat EJB 3.0 -versiota, mikä mahdollistaisi komponenttitekniikan uusien ominaisuuksien hyödyntämisen. Tätä kirjoittaessa uusin WebLogic Server -versio on 10.2.

Päivitystä vaativat komponentit

ParLetin EJB-komponenttien ja näiden sisältämien Java-luokkien määrä laskettiin Windows XP:n komentokehotteen avulla. Taulukossa 1 komponentit on jaoteltu pakkauksiin ja mahdollisiin alipakkauksiin.

Taulukko 1: EJB-komponenttien määrä ParLetissa pakkauksittain (tilanne 1.10.2008)

Pakkaus	**Komponentteja / kpl**
parlet_ejbs	
parlet.control	126
parlet.data	75
parlet.message	82
parlet.printing	22
parlet.stat	6
parlet_cleaner	
parlet.cleaner	13
parlet_common	
parlet.common	4
Yhteensä	**328**

Taulukko 2 erittelee pakkauksittain komponenttien sisältämien Java-luokkien määrän.

Taulukko 2: EJB-komponentteja sisältävien pakkauksien Java-luokkien määrä (tilanne 1.10.2008)

Pakkaus	**Käsintehtyjä**	**Generoituja**
parlet_ejbs	493	1264
parlet_cleaner	40	66
parlet_common	123	165
Yhteensä	**656**	**1495**

Käsintehdyt luokat ovat sovelluskehittäjien luomia, kun taas generoidut luokat luodaan automaattisesti EJBGen -työkalulla sovelluksen kääntövaiheessa.

6.2 Mahdollisia migraatiotapoja

EJB 3.0-version käyttöönotossa on useita eri mahdollisuuksia. Tässä kappaleessa käsitellään mahdollisia migraatiotapoja hyvien ja huonojen puolten osalta, sekä muutosten vaikutuksia sovelluksen toimintaan ja kehitykseen tulevaisuudessa.

Eri versiota edustavien EJB -komponenttien keskinäinen yhteensopivuus mahdollistaa useita erilaisia vaihtoehtoja migraation suhteen. Seuraavassa esitellään neljä mahdollista migraatiotapaa ja eritellään niiden vaikutuksia ParLetin toiminnan kannalta.

6.2.1 Täysi muutos

Täydessä muutoksessa sekä EJB-komponenttiluokat muutetaan EJB 3.0-versiolle ja entiteettikerros Java Persistence API:lle. Tämä olisi tekninen ihannetilanne, jossa kaikki uuden version tuomat edut saataisiin käyttöön. Muutostyön ongelmaksi nousisi kuitenkin todennäköisesti hyvin suuri vaadittavien työresurssien määrä.

Hyvät puolet

- Komponenttikerroksen tekninen yhtenäisyys
- Mahdollinen suorituskyvyn paraneminen
- Sovelluksen fyysisen koon pienentyminen
- Ohjelmakoodin ylläpidon yksinkertaistuminen
- Mahdollisesti sovelluksen käyttöiän pidentyminen

Ongelmakohdat

- Erittäin suuri työmäärä
- Tarvitaan paljon asiantuntijaresursseja
- Mahdollisesti alhainen kustannustehokkuus

6.2.2 Priorisoitu migraatio

Priorisoimalla päivittäminen tarkoittaa vain tärkeimpien komponenttien päivittämistä. Työn määrittelemiseksi komponentit tulisi tulisi asettaa ensijaisuusjärjestykseen tai prioriteettiryhmiin esimerkiksi komponentin toiminnan kriittisyyden, suorituskykyfaktorin, komponenttityypin tai päivityksen helppouden perusteella. Täten päivitettäväksi voitaisiin valita resurssien sallima määrä prioriteetiltaan korkeita komponentteja.

Tämän tavan suurin etu täyteen muutokseen verrattuna on kustannustehokkuus. Huomattavasti pienemmällä työmäärällä on mahdollista saada suhteessa verrattain suuri etu. Osittaisten muutosten suurimpana vaarana on mahdollisuus EJB 2.x –versioiden tuen loppuminen tulevaisuuden sovelluspalvelimilla. Tämän riskin todennäköisyyttä on kuitenkin vaikea arvioida.

Hyvät puolet

- Hyötysuhde resurssien käytöllä ja saavutetuilla eduilla huomattavasti parempi täyteen muutokseen verrattuna
- Nykyisellään hyvin toimiviin osa-alueisiin ei tarvitse kajota

Ongelmakohdat

- Komponenttikerros epäyhtenäistyy teknisesti
- Pirstoutuminen eri versiota edustaviin komponentteihin voi vaikeuttaa ohjelmakoodin ylläpitoa
- Sovellukseen jätetyt vanhaa versiota edustavat komponentit voivat tuottaa yhteensopivuusongelmia tulevaisuudessa (esimerkiksi sovelluspalvelimen versionnostoissa)

6.2.3 Osittaiset migraatiot kerroksittain

Erotuksena edellä mainittuun priorisoituun migraatioon osittaisella migraatiomenetelmällä tavoitellaan loogisten kerrosten teknistä yhtenäisyyttä. Tässä tapauksessa esimeriksi pelkkä entiteettikerros muutettaisiin käyttämään Java Persistence API:a, EJB-kerroksen pysyessä 2.0 -version mukaisena. Toisena mahdollisuutena olisi

päinvastainen tilanne, eli pelkän EJB-kerroksen muuttaminen entiteettikerroksen pysyessä ennallaan.

Hyvät puolet

- Loogisten kerrosten tekninen yhtenäisyys

Ongelmakohdat

- Mahdollisesti alhainen kustannustehokkuus

6.2.4 Jatkokehitys EJB 3.0 -versiolla

Tässä vaihtoehdossa pelkästään sovelluksen jatkokehitys tapahtuisi uudella versiolla. Käytännössä vain sovellukseen lisättävät uudet komponentit toteuttaisivat EJB 3.0 -versiota. Tämä edellyttäisi entiteettikerroksen pitämistä ennallaan, joten Java Persistence API:n käyttöönotto ei olisi mahdollista.

Hyvät puolet

- Korkea kustannustehokkuus
- Vaivaton toteuttaa

Huonot puolet

- Saavutettu hyöty varsin pieni edellämainittuihin migraatiomenetelmiin verrattuna
- EJB-kerroksen tekninen yhtenäisyys heikkenee

6.3 ParLet-koeympäristön toteuttaminen

EJB 3.0 -version käyttöönottoon ParLetin kehitystyössä liittyviä toimenpiteitä haluttiin tutkia käytännössä. Osana opinnäytetyötä luotiin kokeellinen palvelinympäristö, johon tuotiin erikseen toimeksiantajan valitsema EJB-komponentti käytössä olevasta ParLetin kehitysympäristöstä. Komponenttiin tehtiin tarvittavia muutoksia, jotta se toimisi uudessa ajoympäristössä.

Seuraavassa käydään läpi työn tavoitteet, määrittely, suunnittelu, toteutus ja tulokset.

6.3.1 Koeympäristön vaatimusmäärittely

Migraatiota koskevan tausta-aineiston tuottamisesta neuvoteltiin yhdessä toimeksiantajan kanssa. Toimeksiantaja halusi käytännön esimerkin ParLetin komponenttien muuttamisesta EJB 3.0 -version mukaiseksi. Neuvottelujen pohjalta päädyttiin seuraaviin vaatimuksiin (Palaveri 13.6.2008):

1. *Pyritään osoittamaan, kuinka EJB -tekniikan versionvaihto vaikuttaa yksittäisen komponentin ohjelmakoodin syntaksiin ja luokkarakenteeseen*

2. *Tuotetaan EJB 3.0- koodiesimerkki kehittäjäryhmälle*

3. *Testataan käytännössä ParLetin EJB 2.0 -entiteettikomponentin muuttaminen Java Persistence API -komponentiksi*

4. *Listataan toteutuksessa esiintyvät ongelmakohdat ja muut huomionarvoiset seikat*

5. *Tuotetaan taustatietoa migraation työmäärän arvioimiseksi*

Vaatimuslistan kohdat 1 – 4 pyrkivät hyödyttämään etupäässä ParLetin kehitystyössä toimivia ohjelmoijia. Kohdan 5 tuottamia tuloksia pyritään käyttämään tulevaisuudessa mahdollisen migraation työmäärää arvioidessa.

Tavoitteiden saavuttamiseksi työn sisältöä määriteltiin yhdessä toimeksiantajan kanssa. Päätettiin, että luodaan EJB 3.0 -versiota tukevalle sovelluspalvelimelle koeympäristö, johon tuodaan yksittäinen komponentti ParLetin nykyisestä kehitysversiosta. Tämä komponentti muutetaan EJB 2.0 -syntaksista EJB 3.0-version sekä Java Persistence

APIn mukaiseksi. Muutettavaksi komponentiksi toimeksiantaja valitsi Vaihtokurssi-komponentin.

Todettiin, että koeympäristön pystyttäminen vaatii uuden sovelluspalvelimen asentamisen, sillä ParLetin tämänhetkinen kehityspalvelin ei tue EJB 3.0 -versiota. Sovelluspalvelimeksi valittiin BEA WebLogic 10.2.

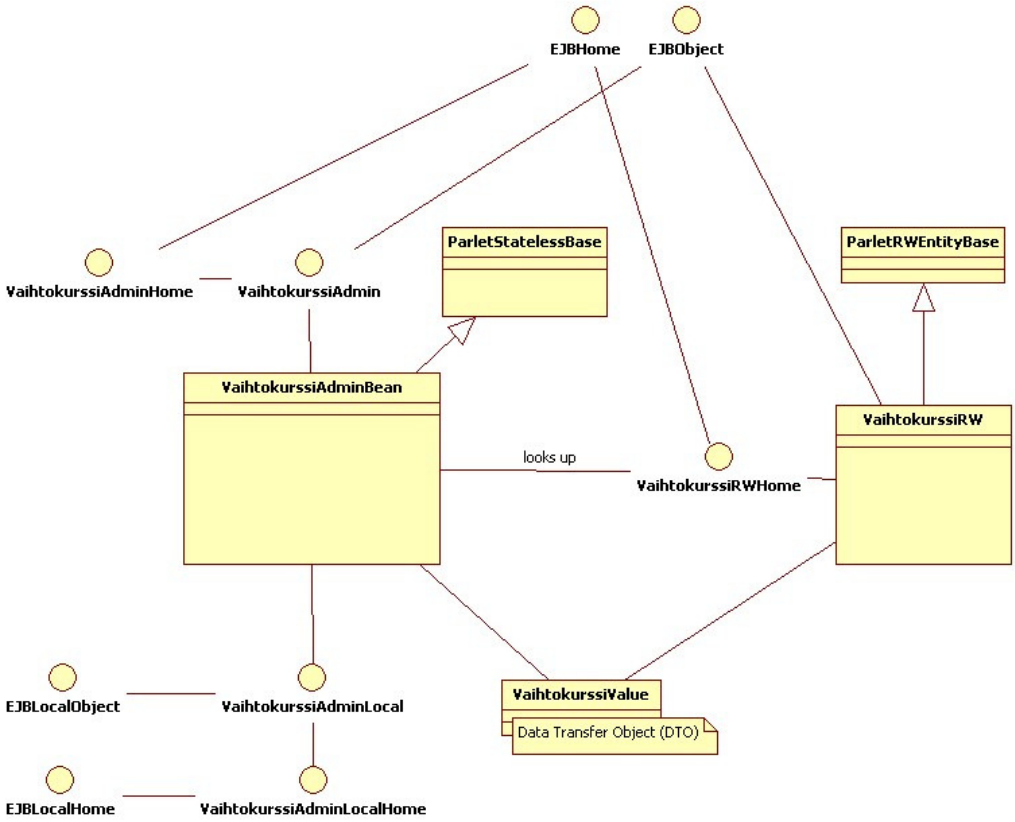
Prosessin työvaiheet muotoiltiin seuraavasti:

1. *Sovelluksen suunnittelu*
2. *Koeympäristön asennus ja konfigurointi*
3. *Komponentin siirto ja muokkaus*
4. *Testaus ja tulosten poiminta*

Laadin työvaiheiden pohjalta aikatauluarvion ja toteutussuunnitelman.

6.3.2 Koeympäristön tekninen suunnittelu

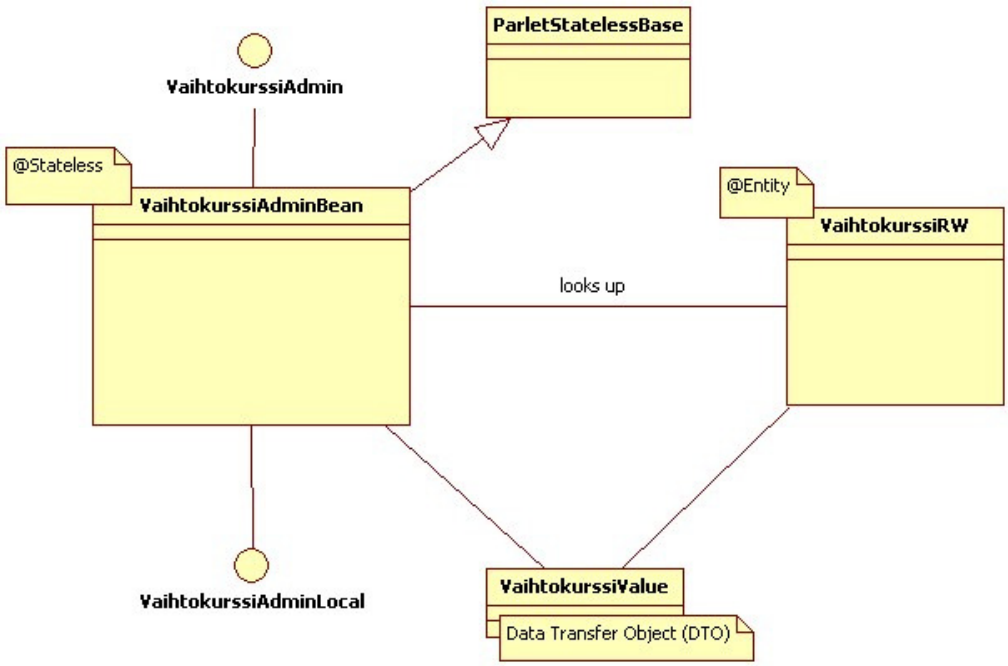
`VaihtokurssiAdminBean` on istuntofasadina toimiva tilaton komponentti. Se sisältää metodit `VaihtokurssiRW`-nimisen entiteettikomponentin hallinnointiin. Kyseessä on kokonaisuudessaan melko pieni komponentti ParLetin keskimääräiseen komponenttikokoon verrattuna. Komponentin luokkarakennetta esitellään kuviossa 11.



Kuvio 11 Lähtötilanne: Vaihtokurssi-komponentin luokkakaavio EJB 2.0 -versiolla.

Kuten kaaviosta näemme, komponentti perii tyypillisen EJB 2.0 -komponentin tavoin useita EJB-kirjaston rajapintoja. Nimeltään ”Vaihtokurssi” -alkuiset rajapinnat generoidaan kehitysympäristössä EJBGen -työkalun avulla.

Suunnitteluvaiheessa laadin komponentille uuden, EJB 3.0 -version mukaisen rakenteen. Tässä mallissa EJB 2.0 entiteetti korvautuu Java Persistence API -komponentilla. EJB-kirjaston rajapinnat korvautuvat annotaatioilla, joten perittävien rajapintojen määrä on vähentynyt. Tätä rakennetta esitellään kuviossa 12.



Kuvio 12 Tavoitetilanne: Vaihtokurssi-komponentin luokkakaavio EJB 3.0 -versiolla

Toteutettujen luokkakaavioiden avulla hahmottelin koeympäristössä tavoiteltavan luokkarakenteen ja kuvion 12 avulla valitsin uuteen ympäristöön siirrettävät luokat.

6.3.3 Koeympäristön toteutus

Koeympäristön pystyttäminen alkoi BEA WebLogic 10.2 -sovelluspalvelimen asennuksella. Kehitystyökaluksi kokeilin aluksi BEA:n Workspace Studio 1.1 -ohjelmistoa, mutta EJB 3.0-työkalujen puuttellisuuden takia päädyin lopulta Eclipse IDE:n 3.4.1-versioon.

Kehitysympäristön pystyttämiseen jälkeen toin Vaihtokurssi-komponentin ohjelmakoodit ParLetin kehitysversiosta uuteen ympäristöön. Generoin komponentin tarvitsemat rajapinnat manuaalisesti EJBGen-työkalulla. Konfiguroin koeympäristön ratkaisemalla komponentin vaatimat kirjastoriippuvuudet.

Seuraavaksi muokkasin ohjelmakoodin luokittain EJB 3.0 -version mukaiseksi. Tein luokista POJO:ja poistamalla EJB-sidonnaisten rajapintojen implementoinnit. Korvasin käytönkuvaimet annotaatioilla. Aloitin muuttamalla sessiokomponentit, jonka jälkeen siirryin työläämpään osuuteen, eli entiteettikomponentin muokkaamiseen. Selkeästi työläin yksittäinen muutostekijä oli EJB 2.0 -entiteettikomponentin muuttaminen Java Persistence API -komponentiksi.

6.3.4 Koeympäristön rakennusprosessin tulokset

Prosessin toteutuksen jälkeen arvioin saavutettuja tuloksia. Seuraavassa käyn läpi vaatimusmäärittelyssä asetettujen tavoitteiden toteutumisen kohta kohdalta.

1. Tavoite: ***Pyritään osoittamaan kuinka EJB -tekniikan versionvaihto vaikuttaa yksittäisen komponentin ohjelmakoodin syntaksiin ja luokkarakenteeseen***

   Tulos: Tavoite toteutui. Vaihtokurssi-komponentti toteutettiin EJB 3.0 -syntaksin mukaisena. Suunitteluvaiheessa luotu luokkarakenne pystyttiin toteuttamaan käytännössä.

2. Tavoite: ***Tuotetaan EJB 3.0 koodiesimerkki kehittäjäryhmälle***

   Tulos: Tavoite toteutui. Vaihtokurssi-komponentti toimii koodiesimerkkinä.

3. Tavoite: ***Testataan käytännössä ParLetin EJB 2.0 entiteettikomponentin muuttaminen Java Persistence API-komponentiksi***

   Tulos: Tavoite toteutui. Vaihtokurssi-komponentin entiteettiluokka muunnettiin Java Persistence API -komponentiksi.

4. Tavoite: ***Listataan toteutuksessa esiintyvät ongelmakohdat ja muut huomionarvoiset seikat***

   Tulos: Tavoite toteutui. Ongelmakohdat ja huomiot listattiin.

5. Tavoite: ***Tuotetaan taustatietoa migraation työmäärän arvioimiseksi***

   Tulos: Tavoite toteutui osittain. Työmäärät kirjattiin ylös, mutta koska Vaihtokurssi-komponentin on selkeästi keskimääräistä ParLet-komponenttia

pienempi, on sen perusteella vaikea tehdä koko sovelluksen migraatiota koskevia arvioita.

Parannusehdotus: ParLetin komponenttikerroksen migraation työmäärän arvioimiseksi tulisi valita mahdollisimman suuri komponentti, jolloin pystyttäisiin arvioimaan suurin mahdollinen työmäärä joka muutostyöhön voi mennä.

Kaiken kaikkiaan varsinainen koeympäristön rakennus sujui joitain kehitystyökalujen käyttämisessä esiintyineitä ongelmia lukuun ottamatta hyvin. Tuotettu koeympäristö ohjelmakoodeineen on toimeksiantajan käytettävissä jatkotoimenpiteitä varten.

7 Yhteenveto ja johtopäätökset

Opinnäytetön tuloksena tuotin tausta-aineistoa ParLet-sovelluksen liiketoimintakerroksen versiomigraation suunnitteluun. Toivon tuloksien antavan tukea toimeksiantajalle tulevaisuuden jatkotoimenpiteitä suunnitellessa. Opinnäytetyöprosessi oli varsin pitkäkestoinen, jonka johdosta myös työn sisältö eli jonkin verran prosessin aikana.

Mielestäni päätavoite saavutettiin. Tämä työ antaa lukijalleen kuvan EJB-päivitysprosessin luonteesta ParLetin osalta, niin teoria- kuin käytäntötasolla. Suurin ongelma työssä oli työmäärän arviointi, sillä alkuperäinen työn rajaus oli liian laaja opinnäytetyöhön mitoitettuun työmäärään nähden. Työ nosti kuitenkin esille joitain lisäselvitystä vaativia tekijöitä tulevaisuutta ajatellen.

Jälkeenpäin käytetyistä menetelmistä on helppo löytää kehitettävää tulevaisuuden varalle. Koeympäristön kohdekomponentiksi valittiin tiukan aikataulun takia mahdollisimman yksinkertainen komponentti. Jotta syntaksin muutos olisi tullut paremmin esille, tulisi muutettavaksi valita kaksi entiteettikomponenttia, jotka omaavat relaation toisiinsa.

Jotta päivitykseen kuluvien työmäärien arviointi olisi mahdollista, voitaisiin komponentit luokitella erilaisiin kategorioihin kompleksisuutensa mukaan. Eri kompleksisuusluokille voisi arvioida työmäärät, joiden perusteella erilaisten päivitysvahtoehtojen työmäärien laskenta olisi helppoa. Näiden luokkien pohjalta voitaisiin myös laskea pisin mahdollinen sovelluksen muuttamiseen kuluva aika.

Mielestäni tärkein työn aikana esille tullut yksittäinen tekijä oli eri versioisten komponenttien keskinäinen yhteensopivuus. Työtä aloitettaessa kehittäjäryhmässä ei tiedetty osittaisten migraatioiden olevan mahdollisia. Työn myötä paljastui siis useita täysin uusia mahdollisia päivitysmenetelmiä. Itselleni työstä oli suurta hyötyä, sillä tuntemukseni syventyi sekä kohdejärjestelmän että käytettävien tekniikoiden osalta.

Tätä opinnäytetyötä voidaan hyödyntää myös muissa EJB-migraatiota vaativissa kohteissa. Työn teoreettinen viitekehys sisältää yleishyödyllistä tietoa EJB-tekniikasta ja mahdollisista menetelmistä. Joissain määrin tuloksia voidaan hyödyntää myös muita tekniikoita kuin EJB:tä käyttävissä sovelluksissa.

Kun ParLetin liiketoimintakerroksen tulevaisuudesta tehdään päätöksiä, voidaan tässä työssä esille tulleita menetelmiä harkita toteutettaviksi. Koeympäristö antaa kehittäjille kuvan uuden tekniikan ohjelmakoodin syntaksista ja on käytettävissä tulevaisuuden jatkotoimenpiteitä suoritettaessa. Tulevaisuudessa aiheeseen liittyvien toimenpiteiden painopistettä voitaisiin mielestäni siirtää priorisoidun päivityksen mahdollisuuksien tutkimiseen ja työmäärien arviointiin.

Lähteet

BEA WebLogic Server – Programming WebLogic Enterprise JavaBeans, Version 3.0. 2007.

DeMichiel, Linda G.,Yalcinalp, L. Ümit & Krishnan, Sanjeev. 2001. Enterprise JavaBeans specification, Version 2.0. Sun Microsystems.

EJBGen [online] [viitattu 16.10.2008]. http://beust.com/ejbgen

The Java Community Process(SM) Program – JSRs: Java Specification Requests – List of all JSRs [online] [viitattu 25.9.2008] http://jcp.org/en/jsr/all

Java EE Tutorial [online] [viitattu 16.10.2008]. http://java.sun.com/javaee/5/docs/tutorial/doc/

Panda, Debu, Rahman, Reza & Lane, Derek. 2007. EJB 3 in action. Manning Publications Co.

Panda, Debu. Loosing weight is easier than you think [online] [viitattu 16.10.2008] www4.java.no/presentations/javazone/2006/slides/4472.ppt

Shannon, Bill. 2006. Java™ Platform, Enterprise Edition (Java EE) Specification, v5. Sun Microsystems.

Tiwari, Shashank 2006. Migrating EJB 2.x applications to EJB 3.0 [online] [viitattu 25.9.2008]. www.javaworld.com/javaworld/jw-08-2006/jw-0814-ejb_p.html

The Java EE 5 Tutorial. 2006. Sun Microsystems.

Your First Cup: An Introduction to the Java EE Platform. 2006. Sun Microsystems.

Logia Software Oy, 2004. KVAIMS Toiminnalinnen määrittely.

Logia Software Oy, 2006. ParLet ratkaisukuvaus 3.3.

Tomi Rainio, ohjelmistosuunnittelija, Logia Software Oy, haastattelu 2.5.2008. Tampere.

Merja Järvinen, tulosyksikön johtaja ja Tomi Rainio, ohjelmistosuunnittelija, Logia Software Oy, palaveri 13.6.2008. Tampere.