Suunnitteludokumentti

Ohjelmistotuotantoprojektin tietojärjestelmä — OhtuTie

Helsinki 16.7.2004

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Janne Nevalainen
Jyrki Kankaanpää
Sinikka Loikkanen
Esa-Matti Miettinen
Petteri Nurmi
Max Österman

Asiakas

Turjo Tuohiniemi

Vastuuhenkilö

Juha Taina

Kotisivu

`http://www.cs.helsinki.fi/group/otie/`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	24.6.2004	Tarkastettu ja korjauksin hyväksytty versio
1.1	28.6.2004	Korjattu versio
1.2	15.7.2004	Poistettu luokat InputComponent ja MetricComponent Lisätty TableComponent ja sen perivät Bean-luokat Päivitetty Tomcat-ympäristön kuvausta Päivitetty tietokantakaavio

Sisältö

1 Johdanto

Tämä suunnitteludokumentti kuvaa toteutettavan Ohjelmistotuotantoprojektien tietojärjestelmän – OhtuTie – teknisen toteutuksen näkökulmasta. Suunnitteludokumentti toimii järjestelmätoteutuksen ohjeena ja siinä kuvataan järjestelmän arkkitehtuuri, komponentit, tietosisältö, käyttöliittymä sekä laadunvalvontaan ja testaukseen kuuluvia tekijöitä. Suunnitteludokumentin perustana käytetään OhtuTie-projektin määrittelydokumenttia 1.0.

1.1 Tuotteen tausta ja tarkoitus

Helsingin yliopiston tietojenkäsittelytieteen aineopintoihin kuuluu Ohjelmistotuotantoprojekti-kurssi, jonka tavoitteena on antaa opiskelijoille käytännön tuntumaa ohjelmistotuotannon välineisiin ja menetelmiin projektimuotoisen ryhmätyöskentelyn puitteissa.

Kurssille osallistuvat opiskelijat jaetaan aluksi projektiryhmiin ja he aloittavat työskentelynsä laatimalla projektisuunnitelman. Suunnitelman tavoitteena on toimia projektin aikana ryhmän työskentelyä ohjaavana dokumenttina ja siinä arvioidaan mm. ohjelmiston kokoa, tarvittavaa työmäärää ja projektin aikataulutusta. Ongelmana on kuitenkin ollut, että aikaisemmista ja vastaavista projekteista ei ole saatavissa historia- ja rakennetietoja. Näin opiskelijat eivät saa suunnittelunsa pohjaksi vertailukelpoista aineistoa. Lisäksi laitoksen kokeelliselta ohjelmistotutkimukselta puuttuu helposti saatavilla olevaa aineistoa, koska ohjelmistotuotantoprojekteihin liittyvä dokumentaatio on työlästä läpikäytävää.

Kurssia uudistetaan vuoden 2004 aikana siten, että projekteista aletaan kerätä laajempia mittaustietoja. Aikaisemmista projektitöistä on kerätty mittaustietoina lähinnä projektivaiheisiin liittyvät työtuntimäärät. Kerättävät mittaustiedot eivät kuitenkaan olleet kesällä 2004 projektiryhmän työskentelyaikana täsmällisesti määriteltyjä ja niihin onkin odotettavissa muutoksia lähitulevaisuudessa. Siksi mittaustietoja ei voi tässä vaiheessa täysin kiinnittää eli ne halutaan pitää muutettavina. Muuttuvista mittaustiedoista käytetään dokumentissa nimitystä metriikka ja loogisesti yhteensopivat metriikat muodostavat metriikkasarjan. Tulevien projektien osalta mittaustulosten keruu annetaan projektiin osallistuvan opiskelijan huolehdittavaksi ja saatavia mittaustuloksia tullaan käyttämään laitoksen kokeellisen ohjelmistotutkimuksen aineistona.

Mittaustietojen tallennukseen ei Tietojenkäsittelytieteen laitoksella kuitenkaan löydy olemassa olevaa järjestelmää. Laitoksella on siksi todettu tarpeelliseksi kehittää sovellus, jota voidaan käyttää ohjelmistotuotantoprojekti-kursseihin liittyvien mitattavien historia- ja rakennetietojen tallennukseen. Työväline ratkaisisi tai toimisi alustavana ratkaisurunkona edellä mainituissa tutkimukseen ja opiskeluun liittyvissä ongelmissa.

Tavoitteena on myös mahdollistaa entistä tarkempi projektinhallinta. Se on mahdollista siten, että järjestelmän avulla kunkin projektin ohjaaja ja kurssin vastuuhenkilö voivat seurata projektityön edistymistä tallennettujen mittaustietojen perusteella.

Tämän OhtuTie-ohjelmistotuotantoprojektiryhmän tarkoituksena on suunnitella ja toteuttaa tulevien ohjelmistotuotantoprojektiryhmien mittaustietojen tallentamiseen soveltuva järjestelmä. Sen tulee sisältää mittaustietojen tallentamiseen ja käyttöön soveltuva tieto-

kanta sekä tietokannan käyttöliittymä. Sen tulee lisäksi tarjota mahdollisuus projektitietojen seurantaan projektiryhmän ohjaajalle ja kurssin vastuuhenkilöille. Lisäksi järjestelmän tulee tarjota rajapinta mittaustietojen visualisointia varten.

Järjestelmän suunnittelussa huomioidaan järjestelmään kohdistuvat tulevat muutostarpeet tietosisällön muuttumisen osalta. Siksi järjestelmästä tehdään mahdollisimman dynaaminen, jotta muuttuvat metriikat ja metriikkasarjat ovat sen avulla esitettävissä.

1.2 Terminologiaa

HTTP: *Hypertext Transfer Protocol*, World Wide Webissä asiakas- ja palvelintietokoneiden (client/server) kommunikointikieli.

JDBC-rajapinta: on *Java Database Connectivity* -rajapinta

Metadata: on tietoa tiedosta.

Metriikka: on mitattava tieto. Metriikalla on nimi, kuvaus ja tyyppi (metriikkamalli). Kuvaus on samalla ohje käyttäjälle. Esimerkiksi valmiista koodista laskettu koodirivien lukumäärä on metriikka.

Metriikkamalli: on metriikan tyyppi, joka määrittää miten metriikka tulostetaan ja tallennetaan. Metriikkamalleja ovat totuusarvo, numeerinen arvo, taulukkonumeerinen arvo, päivämäärä ja vapaa tekstisyöte.

Metriikkasarja: on kokoelma yhteenliittyviä metriikoita. Esimerkkinä toteutusvaiheessa kerättävät tiedot (metriikat) sekä testausvaiheesta kerättävät tiedot.

Näkymä: on käyttöliittymän selkeästi rakenteellisesti erottuva osa. Esimerkiksi yksi välilehtisivu on näkymä ja kukin välilehtisivu muodostaa oman näkymänsä, jos se

Projekti: on Helsingin yliopiston tietojenkäsittelytieteen laitoksen kurssin Ohjelmistotuotantoprojekti puitteissa toteutettava projektimuotoinen ryhmätyö, jonka tavoitteena on ohjelmistosovelluksen toteuttaminen.

Projektin tiedot: ovat ne metriikkatiedot, joiden avulla yksilöidään kukin ohjelmistotuotantoprojekti.

Servlet: on palvelinsovelma. Suoritettava Java-luokka, joka toteuttaa rajapinnan javax.servlet.Servlet ja joka ajetaan Web-palvelimella. Vastaanottaa ja vastaa Web-clientien palvelupyyntöihin.

Tunnistenumero: on vastuuhenkilön antama ohjelmistotuotantoprojektit lukukauden sisällä yksilöivä numero. Esimerkkinä kesä 2004 projekti numero 2 (OhtuTie).

W3C: World Wide Web Consortiumin (W3C) tavoitteena on kehittää WWW-kulttuurin yhteisiä ja yhteensopivia pelisääntöjä ja teknologioita. W3C pyrkii suosituksillaan luomaan yleiskäyttöisiä ohjeistuksia mm. yhteensopivuuden takaamiseksi, näin esimerkiksi WWW-sivut näkyvät eri selaimissa oikein. Ks. `http://www.w3c.org/`.

XML: *eXtensible Markup Language* on metakieli, jonka avulla kuvataan tiedon rakennetta. XML on väline, jonka voidaan esittää tiedon hierarkia sekä kapseloida data.

XSL: *eXtensible Style Sheet Language* on kieli, joka on tarkoitettu käytettäväksi ensisijaisesti asiakaslaitteiden (client) XML-datan esittämiseen ja muotoiluun.

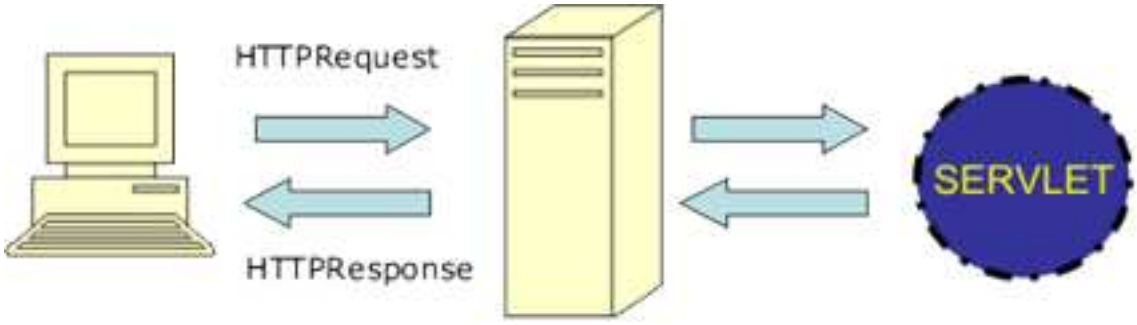
XSLT: *eXtensible Style Sheet Language Transfer* on kieli, joka eroaa XSL:stä lähinnä siten, että se on tarkoitettu käytettäväksi palvelimella (server).

1.3 Dokumentin rakenne

Luvussa 2 annetaan kokonaiskuva tuotettavasta ohjelmistosta. Luvussa 3 kuvataan ohjelmiston arkkitehtuuri. Luvuissa 4, 5 ja 6 kuvataan järjestelmän nelikerrosrakenne eli asiakas-, sovelluslogiikka-, komponenttilogiikka- ja tietokantakerros. Luvussa **??** kuvataan järjestelmään tietokantaoperaatioiden helpottamiseksi toteutettava apukirjasto. Luvussa 8 kuvataan XML-tiedostojen käytön helpottamiseksi laadittava XML-apukirjasto. Luvussa 9 kuvataan istuntojen hallinnasta vastaava komponentti. Luvussa 10 kuvataan ohjelmiston datan visualisointiin tarkoitettu rajapinta. Luvussa 12 kuvataan järjestelmän tietosisältö. Luvussa 13 kuvataan järjestelmän metadatarakenne. Luvussa 14 kuvataan järjestelmän käyttöliittymä yleisellä tasolla. Luvussa 15 kerrotaan lyhyesti sovelluksen testaukselle asetettavat tavoitteet. Luvussa 16 on sekvenssikaaviot käyttötapauksista.

2 Järjestelmän yleiskuvaus

Ohjelmistotuotantoprojektien tietojärjestelmä toteutetaan *Java Servlet* -teknologiaa hyödyntäen. Servlet-teknologiassa HTTP-protokollan välityksellä tehdyt sivupyynnöt muunnetaan Java-luokaksi (*Servlet Request*), joka välitetään viiteparametrina servlet-objektille. Servlet-objekti vastaa sivun sisällön generoimisesta sekä palauttaa tuotetun sivun palvelimelle *Servlet Response*-oliona, josta palvelin palauttaa käyttäjälle tuotetun sivun. Palvelimen rooli tämänkaltaisessa mallissa on toimia eräänlaisena puhelinvaihteena, joka välittää pyynnöt oikeille komponenteille. Sivupyynnön prosessoinnin vaiheita on havainnollistettu kuvassa 1.



Kuva 1: Sivupyynnön käsittelyprosessi Servlet-teknologiaa käytettäessä.

Servlet-sovellukset toimivat yleensä Tomcat-palvelimen välityksellä. Tomcat palvelimessa sovelluksen juurihakemistossa on XML-tiedosto jokaista palvelimella tarjolla olevaa sovellusta kohden. Lisäksi jokaisen sovelluksen pääkansiossa on *web.xml* niminen tiedosto, joka määrittää, miten sivupyynnöt ohjataan eri servleteille. OhtuTie-järjestelmässä tätä ajattelutapaa hyödynnetään myös servlettien sisällä — järjestelmässä on yksi kontrolleriservlet, joka vastaa siitä, että sivupyyntöjen määrittämä ns. näkymä tuotetaan oikealla komponentilla. Eli sen sijaan, että toteutettaisiin erillinen servlet-objekti jokaista sivua varten, luodaan yksi servlet-objekti, joka hyödyntää erillisiä sivukomponentteja, jotka vastaavat pyydetyn näkymän generoimisesta. Näkymällä tarkoitetaan tässä yhteydessä esim. hallintatietosivua. Tällä tavalla järjestelmän rakenteesta saadaan helpommin laajennettava, ohjelmakoodi on paremmin uudelleenkäytettävää ja turhaa "copy–paste-ohjelmointia voidaan välttää. Vastaavasti kuten Tomcat-palvelimessa, myös tämänkaltaisessa ratkaisussa tarvitaan XML-tiedostoa (metatietoa), joka kertoo, mille komponentille näkymäpyynnöt ohjataan.

Sisällön generointi "tyhjästä"jokaisen sivupyynnön yhteydessä ei ole kovinkaan tehokasta saati järkevää. Standardi ratkaisu moderneissa Web-sovelluksissa on hyödyntää sivupohjia (template), jotka ovat eräänlaisia staattisia muotteja ja jotka muodostavat yksittäisen sivun rungon. Tietojen joukkoon on upotettu erityisiä prosessointikäskyjä, jotka korvataan dynaamisella sisällöllä sivupyynnön käsittelyn yhteydessä. Tällä tavoin sivujen ulkoasua voidaan helposti muokata jälkikäteen ilman, että itse sovelluskoodiin tarvitsee tehdä muutoksia ja lisäksi sivun generointi on nopeampaa, koska osa sivusta saadaan luettua suoraan merkkijonovirtana. Jotta sivupohjien käyttäminen olisi mahdollisimman joustavaa, joudutaan turvautumaan metadatan käyttöön. Tässä tapauksessa järjestelmässä tehdään rajoitus siten, että ns. pääkonfiguraatiotiedostossa määritetään hakemistopolku, jossa sivupohjat sijaitsevat ja lisäksi sivupohjatiedostojen nimet sidotaan kiinteästi sovelluksen tarjoamiin näkymiin. Esimerkiksi etusivun sivupohja tiedoston nimenä voisi tällöin olla esim. *frontpage.xml*. Koska järjestelmän tulisi tukea myös monikielisyyttä, otetaan käyttöön ylimääräinen *_li*-lisäosa, jonka avulla osoitetaan haluttu kieli. Järjestelmän oletuskieli on suomi, joten vain englanninkielisiä sivuja varten tarvitsee käyttää loppuosaa.

Määrittelydokumentin vaatimusten mukaisesti sovelluksen eri osien autentikointi on toteutettava mahdollisimman joustavasti. Tätä tarkoitusta varten jokaiselle sivulle ja komponentille voidaan määrittää erillinen turvallisuustunniste. Tietokannan käyttäjätietoihin tallennetaan erillinen oikeustunniste, joka määrittää käyttäjälle tarjotut oikeudet. Kun sivukomponentteja generoidaan, täytyy käyttäjän oikeudet tarkistaa. Tehokkuussyistä tämä toiminto integroidaan suoraan komponenttikoodin sisään. Integrointi tapahtuu määrittämällä rajapinta, joka mahdollistaa turvatietojen välittämisen komponenteille sekä huolehtimalla kehitysvaiheessa siitä, että turvallisuustunnisteet on hyvin määritelty.

Jotta järjestelmän ulkoasu ei olisi sidottu liikaa yhteen muottiin, ja hyvän Web-suunnittelun periaatteen mukaisesti, ulkoasun esittäminen erotetaan erilliseksi osaksi järjestelmää. Koska sivupohjat kuvataan XML-tiedostona, on luontevaa käyttää XSLT-kieltä toteuttamaan ulkoasukuvaukset. Lisäksi Javassa on hyvä sisäänrakennettu XML-tuki, joka mahdollistaa XML ja XSLT -tekniikkoja hyödyntävän sovelluksen laatimisen suhteellisen helposti. Kaiken lisäksi eriyttämällä ulkoasujen esityskerros omaksi komponentikseen, on mahdollista laatia sovellus, jossa käyttäjän selain huomioidaan ulkoasua generoitaessa. Tämäkin

osio vaatii toimiakseen metadataa, joten järjestelmässä on erillinen metadatatiedosto, josta selviää, mitä XSL-tiedostoa muunnokseen tulisi kulloinkin käyttää.

Määrittelydokumentissa sovellus kuvataan nelikerroksisena järjestelmänä, jossa alin kerros on tietokantakerros. Tietokantakerroksen päätehtävänä on kommunikoida itse tietokannan kanssa. Tämän lisäksi toteutetaan kokoelma erillisiä kirjastorutiineja, jotka helpottavat tietokannan käsittelyä. Koska JDBC-rajapinnan käyttö vaatii aina tietokantaajurin lataamista sekä yhteydenottoa tietokantaan, on järjestelmän ulkopuolelle kuuluvat parametrit (ajurin tiedostopolku, tietokannan osoite jne.) eristetty erilliseen metadatatiedostoon. Tämän (ja muiden) komponenttien metadatakuvaukset esitetään suunnitteludokumentin luvussa 13.

Järjestelmässä on lisäksi erillisiä kirjastoja, joiden tehtävänä on helpottaa tiettyjen toimintojen suorittamista. Ensinnäkin järjestelmä hyödyntää XML-kieltä runsaasti, joten järjestelmässä on erillinen XML-kirjasto XML:n käsittelyn helpottamiseksi. Toisekseen, koska asiakkaan vaatimuksena oli, että järjestelmän avulla voidaan toteuttaa graafeja, toteutetaan erillinen graafien piirtokirjasto. Lisäksi HTTP-istuntojen hallintaa sekä salaustoimintoja varten toteutetaan erilliset kirjastoluokat.

Sovelluksen pääkomponentit lyhyine kuvauksineen on esitetty taulukossa 1. Lisäksi taulukkoon on merkitty, mihin kerrokseen (asiakas, sovelluslogiikka, komponenttilogiikka vai tietokantakerros) komponentti kuuluu vai onko kyseessä kirjastokomponentti. Taulukossa 2 puolestaan on esitetty lyhyt kuvaus järjestelmän käyttämästä metadatasta sekä luku 13, josta tarkempi kuvaus on löydettävissä. Taulukon 1 lyhenteiden vastaavuudet esitetään alla ja kuvassa 2 esitetään UML-muotoinen järjestelmän yleiskuvaus.

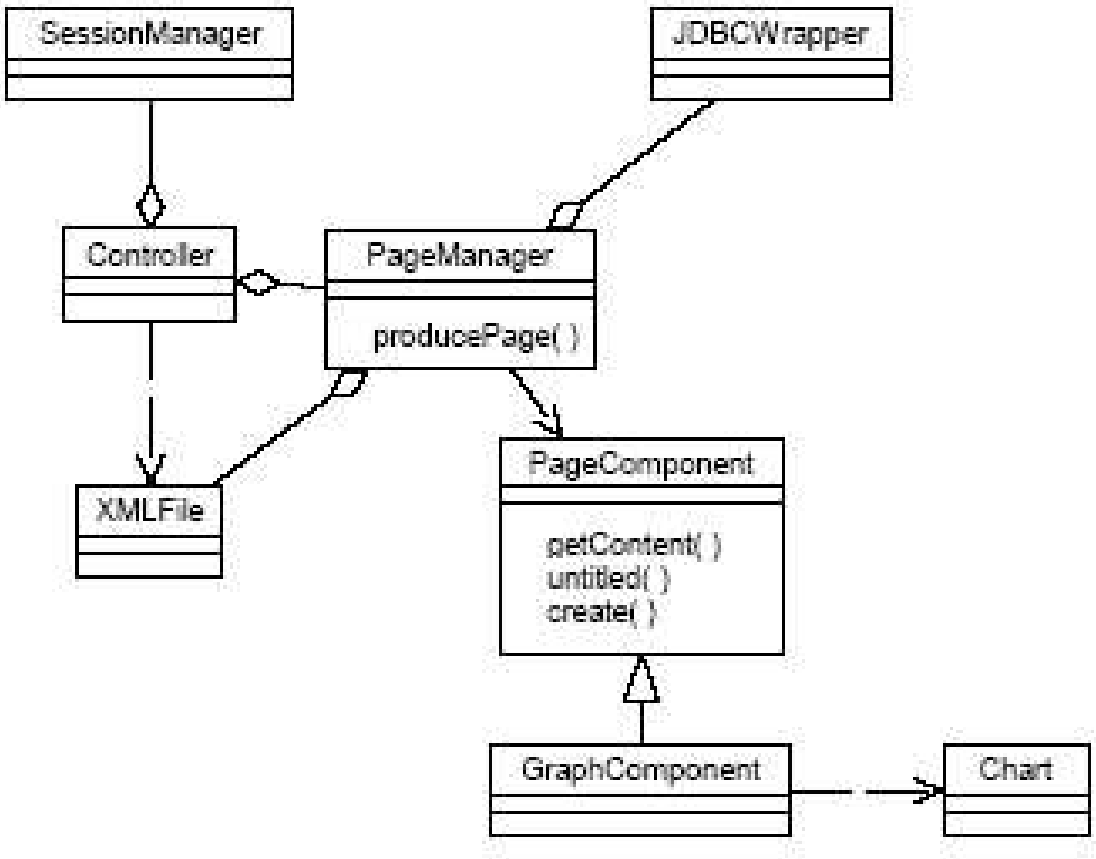
- **AK** = Asiakaskerros
- **SL** = Sovelluslogiikkakerros
- **KL** = Komponenttilogiikkakerros
- **TK** = Tietokantakerros
- **Ki** = Kirjastoluokka

2.1 Toteutus- ja toimintaympäristö

Ohjelmisto toteutetaan Tietojenkäsittelytieteen laitoksen Java-ympäristössä käyttäen Java-ohjelmointikielen versiota 1.4.2.

Sovelluspalvelimena käytetään Apache Tomcatin versiota 4.0.

Kehitysvaiheessa sovellus asennetaan tietojenkäsittelytieteenlaitoksen palvelimelle alkokrunni.cs.helsinki.fi. Tämän palvelimen kohdalla tomcatille luodaan hakemisto tomcat/ ,jonka alle sovellus sijoitetaan. Lähdekoodit tulevat hakemiston ./tomcat/webapps/OTIE/WEB-INF/src/ alle, ja käännetyt class-tiedostot hakemiston ./tomcat/webapps/OTIE/WEB-INF/classes/ alle. Tomcatille luodaan myös asetustiedosto web.xml.



Kuva 2: Yleiskaavio järjestelmän rakenteesta.

Tomcatille täytyy asettaa ympäristömuuttujat, ja tomcat käynnistetään yllä mainitussa hakemistosta tomcat.

Järjestelmän tietokantaympäristönä toimii Oraclen versio 9.0 ja se on asennettu Helsingin yliopiston Tietojenkäsittelytieteen laitoksen Bodbacka-nimiseen palvelintietokoneeseen.

Dokumentointikielenä käytetään suomea ja kommentointikielenä englantia. Java-luokkien nimet ovat englanniksi, kuten myös metodien ja attribuuttien nimet. Tietokannan attribuutteja vastaavat muuttujat nimetään sekaannusten välttämiseksi samoin kuin tietokannassa. Muuttujat kirjoitetaan pienellä alkukirjaimella ja kaksi- tai useampiosaisissa muuttujissa sanan alkukirjain kirjoitetaan isolla kirjaimella, kuten esimerkiksi muuttujan nimi *xmlFile*. Tietokannan attribuutit nimetään siten, että kaksiosaiset nimet kirjoitetaan alaviivan kanssa, kuten *xml_file*.

2.2 Rajaukset

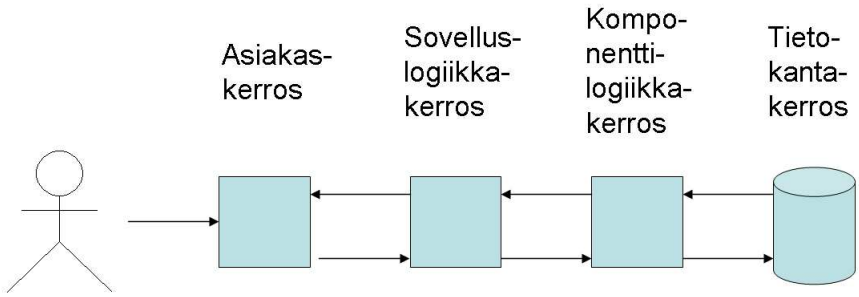
Järjestelmän päävaatimukset ovat graafinen käyttöliittymä sekä tietokantaliittymä.

Järjestelmä tulee toimimaan ainakin Internet Explorer 6.0, Konqueror 3.2.1 ja Mozilla 1.0 -selaimilla. Koska järjestelmää käytetään selaimella, se toimii sekä Windows että Linux

-ympäristöissä ja myös kaikissa muissa käyttöjärjestelmissä, joissa kyseiset selaimet ovat asennettuina.

3 Arkkitehtuurikuvaus

Järjestelmän keskeinen rakenne on esitetty kuvassa 3. Järjestelmän esitys- eli asiakaskerros toteutetaan WWW-selainpohjaisena käyttöliittymänä. Järjestelmän loppukäyttäjälle näkymätön sisäinen tila toteutetaan kahtena erillisenä kerroksena, joista sovelluslogiikkakerros hoitaa käyttöliittymän kautta tulevat syötteet ja kommunikoi kolmannen kerroksen eli komponenttilogiikkakerroksen kanssa. Kolmas kerros vastaa järjestelmän tietokantaoperaatioista.



Kuva 3: Järjestelmän rakennekuvaus

Järjestelmän arkkitehtuuriratkaisuna on 4-tier-MVC -malli, jossa hyödynnetään refleksiivisen ohjelmoinnin periaatteita. MVC eli model-view-controller on arkkitehtuuriratkaisu, jossa järjestelmän mallinnus-, näkymä- ja kontrollointitoiminnot on kukin eriytetty erillisiksi komponenteiksi.

Asiakaskerros eristetään muusta sovelluksesta, mutta kaikki tiedon esittämiseen liittyvät toiminnot toteutetaan asiakaskerroksen tasolla. Asiakaskerroksen sivut tuotetaan XML-muotoisina dokumentteina. XML-dokumentit käännetään ajonaikaisesti XHTML-muotoon erityisten XSLT-muunnostiedostojen avulla. Tällä tavalla pystytään erottelemaan toisistaan ulkoasun sisältö ja esitystapa.

Sovelluslogiikkakerros huolehtii asiakas- ja komponenttilogiikkakerroksien välisen koordinoinnin. Sovelluslogiikkakerrokseen kuuluvat myös ”sivukomponentit”, joiden avulla esityskerroksen sivujen rakentaminen tapahtuu. Sovelluslogiikkakerros koostuu *Controller* ja *PageManager* -komponenteista. Komponentit kuvataan luvussa 5.

Komponenttilogiikkakerros huolehtii sivujen dynaamisen sisällön luomisesta ajonaikaisesti. Komponenttilogiikkakerroksen pääluokka on abstrakti *PageComponent*-luokka. Se on kuvattu aliluokkineen tarkemmin luvussa 6.

JDBCWrapper-luokka eli tietokantakirjasto on sovelluksen keskeisimpiä osia. Se koostuu käytännössä erilaisista tietokannan käsittelyyn tarvittavista metodeista. *JDBCWrapper* tehdään tietokantarajapinnaksi, joka helpottaa Javan JDBC-rajapinnan käyttöä.

XML-apukirjasto laaditaan helpottamaan XML-tiedostojen käsittelyä. Kirjastoluokkien ensisijainen käyttötarkoitus on XML-dokumenttien ajonaikainen manipulointi. Kirjasto on kuvattu tarkemmin luvussa 8.

Taulukko 1: Sovelluksen pääkomponenttien kuvaukset.

Controller	**SL**	Sovelluksen runko. Huolehtii siitä, että oikeita komponentteja kutsutaan oikeassa järjestyksessä ja että komponenteille välitetään oikeat parametrit. Vastaa siis tavallisen Java-sovelluksen main-metodia, paitsi että tällä kertaa pääkoodi toteutetaan Servlet-objektin *doPost()*-kutsuna. Lisäksi kontrolleri on vastuussa pääkonfiguraatiotiedoston lukemisesta.
JDBCWrapper	**TK**	Objektin päätehtävänä on huolehtia automaattisesti tietokantayhteyden ottamisesta sekä tietokanta-ajurin lataamisesta. Lisäksi luokka tarjoaa joukon tietokantarutiineja, joiden avulla mm. SQL-kyselyjen suorittaminen helpottuu.
PageManager	**SL**	Komponentti, joka vastaa oikean sivupohjan lataamisesta, sen pitämisestä muistissa, oikeiden komponenttien kutsumisesta, oikean käyttäjän selaimen mukaan valittavan XSL-ulkoasutiedoston lataamisesta sekä lopullisen XHTML-ulkoasun tuottamisesta.
SessionManager	**Ki**	Luokka, joka tarjoaa rutiineja istuntojen hallitsemiseksi.
XMLFile	**Ki**	Luokka, joka vastaa yhtä XML-tiedostoa. Luokan metodit helpottavat XML-dokumenttien ajonaikaista käsittelyä. Luokat *XMLAttribute* ja *XMLObject* ovat tämän pääluokan apuluokkia.
XMLAttribute	**Ki**	kts. *XMLFile*
XMLObject	**Ki**	kts. *XMLFile*
Chart	**Ki**	Rajapinta yksittäisen graafin esittämistä varten.
SecurityManager	**Ki**	Tietoturvakirjasto, joka tarjoaa salauspalveluja muun sovelluksen käyttöön.
PageComponent	**KL**	Abstrakti luokka, josta kaikki sivukomponentit periytetään. Määrittää yhteisen "rajapinnan", joka muiden luokkien on toteutettava. Täten komponenttien käyttäminen voidaan toteuttaa suoraviivaisesti.
Sivukomponentit	**KL**	Sivukomponentit ovat Java-luokkia, jotka toteuttavat *PageComponent*-luokan määrittämän rajapinnan. Jokainen komponentti vastaa yksittäisen sivun generoinnista tai pienemmästä sivun osasta.

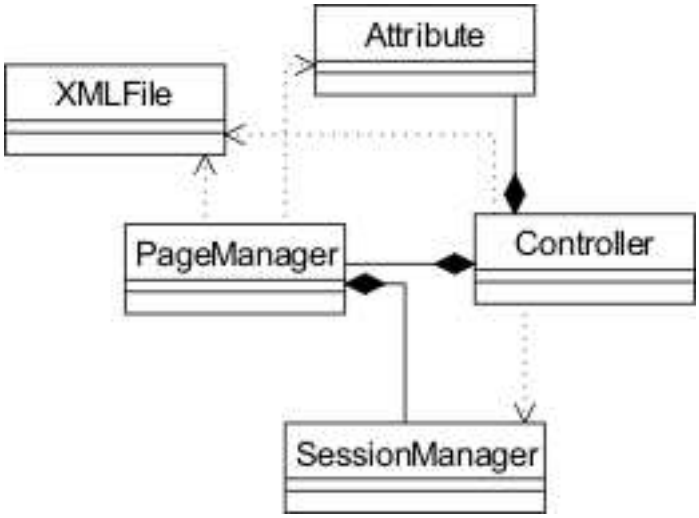
Taulukko 2: Järjestelmän metadatatietojen yleiskuvaus.

otieDBDD.xml	Tiedosto sisältää tietokannan hallinnointiin tarvittavaa tietoa. Tällaista tietoa ovat tietokanta-ajurin täydellinen luokkanimi, tietokannan sijainti, tietokannan käyttäjätunnus ja salasana sekä käytettävän katalogin nimi.
otieLayoutDesc.xml	Tämä tiedosto kuvaa, mikä selain (user agent) liitetään mihinkin XSL-dokumenttiin. Järjestelmään toteutetaan vain yksi XSL-muunnos, mutta jotta järjestelmän laajentaminen olisi myöhemmin helppoa, toteutetaan järjestelmään tuki nopeille ulkoasumuutoksille.
otieLanguages.xml	Tässä tiedostossa määritetään kielille annettavat loppulyhenteet eli esim. *en = English*, jonka jälkeen valitsemalla sovelluksessa kieleksi *English*, järjestelmä osaa hakea oikean sivupohjan lisäämällä *_en* loppuosan.
otieMeta.xml	Sovelluksen pääkonfiguraatiotiedosto. Sisältää mm. sovelluksen käyttämien hakemistopolkujen nimet (ja osoitteet) sekä tietokanta-asennustiedoston nimen ja polun.
otieRightsDesc.xml	Tätä tiedostoa ei sovelluksen (tässä vaiheessa) hyödynnetä mihinkään, mutta kaikki sovelluksen tietoturvatunnisteen kerätään yhteen tiedostoon, joka palautetaan muun sovelluksen yhteydessä.
otieSecurityDesc.xml	Järjestelmän tietoturvakuvaustiedosto. Tiedosto määrittää sovelluksessa symmetriseen salaukseen käytettävät avainainekset sekä käytettävät salausalgoritmit.
[page].xml	Jokainen sivupohja (yksi sivunäkymä) toteutetaan erillisenä XML-tiedostona. XML-tiedostossa on erityisiä <*dynamic*> tageja, joiden välissä oleva alue korvataan dynaamisella sisällöllä.
language.xml	Merkkijonoja, joita voi käyttää sellaisenaan.
templateConfig.xml	Kuvaa miten sivupohjat liittyvät näkymiin eli näkymien ja XML-tiedostojen vastaavuudet.

4 Asiakaskerros

Asiakaskerroksen luokat vastaavat välittömästä kommunikoinnista sovelluksen ja asiakkaan välillä. OhtuTie-sovelluksessa asiakas kerrostuu yhdestä Servlet-luokasta, *Controller*, sekä kahdesta kirjastoluokasta: *Attribute* ja *SessionManager*. Kuva 4 kuvaa näiden luokkien keskinäisiä suhteita sekä asiakaskerroksen suhdetta sovelluslogiikkakerrokseen, jota *PageManager* luokka edustaa.

Asiakaskerroksen päätehtävänä on näyttää asiakkaalle järjestelmän tuottama sivu ja tämä oikeastaan on kaikki, mitä *Controller*-oliot tekevät. Luokan *SessionManager* tehtävänä on puolestaan huolehtia istuntotietojen hallinnasta ja tehdä istuntojen käyttämisestä mahdollisimman helppoa ja vaivatonta.



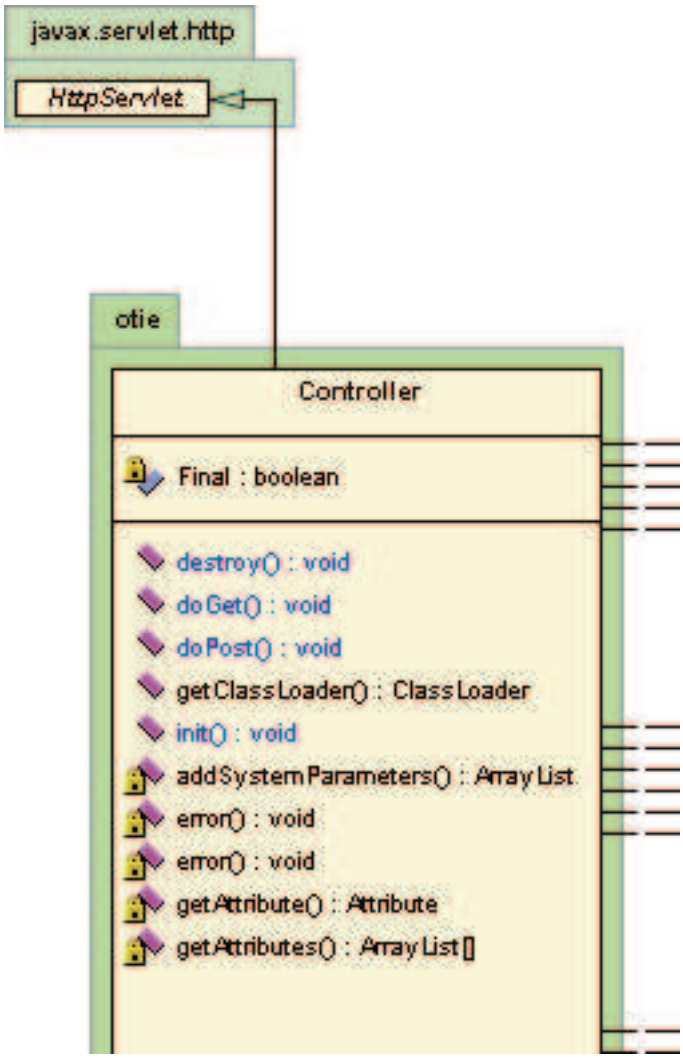
Kuva 4: Asiakaskerroksen pääluokkien rakenne UML-muodossa.

4.1 Controller

Controller-luokka toimii järjestelmässä ainoastaan ns. container-tyyppisenä objektina. Perinteisesti MVC-malleissa Servlettien tehtävänä on huolehtia käskyjen välittämisestä oikeille mallikerroksen komponenteille, mutta OhtuTie-järjestelmässä tämä tehtävä on delegoitu *PageManager*-luokalle. Tällä tavoin saadaan aikaan arkkitehtuurirakenne, jossa testaus ja ylläpito on helpompaa: palvelimen voidaan antaa olla koko ajan päällä ja muita luokkia voidaan testata pääosin kuten tavallisia Java-luokkia.

5 Sovelluslogiikkakerros

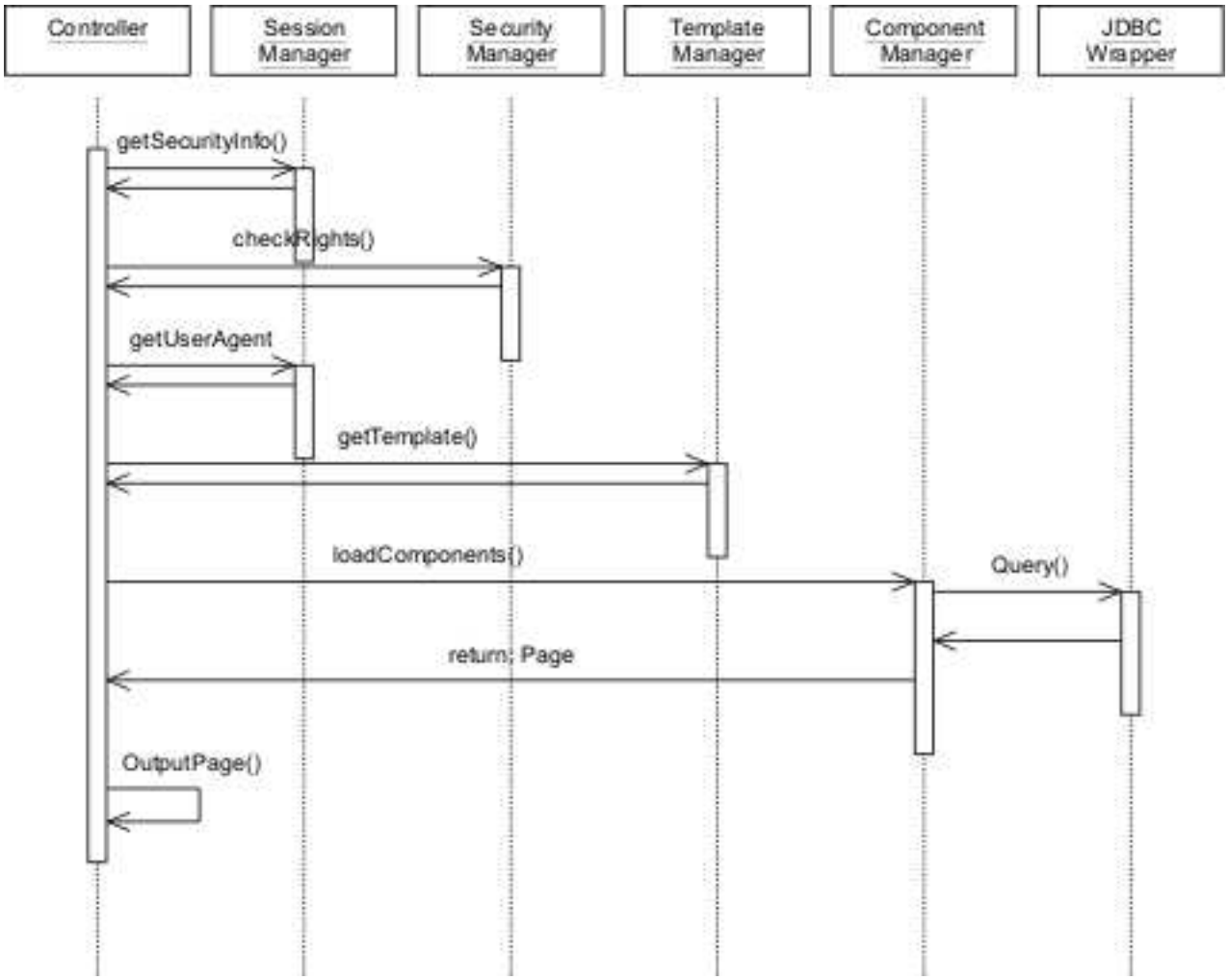
Sovelluslogiikkakerroksen pääasiallisena tehtävänä on huolehtia koordinoinnista asiakas- ja komponenttilogiikkakerroksien välillä. Lisäksi sovelluslogiikkakerrokseen voidaan katsoa kuuluvaksi ns. *sivukomponentit*, jotka ovat valmiita "peruspalikoita", joiden avulla si-



Kuva 5: Controller

vujen rakentaminen voidaan toteuttaa kivuttomasti. Kuvassa 6 on esitetty sovelluksen pääasiallinen tietovirta. Komponentit *SessionManager* ja *JDBCWrapper* ovat komponenttilogiikkakerroksen toimintoja. SessionManager-komponentti kuvataan tarkemmin luvussa 9 ja JDBCWrapper-komponentti kuvataan luvussa **??**. *PageManager*-komponentti kuvataan myöhemmin tässä luvussa.

Kuvan 6 tietovirta etenee seuraavasti. Käyttäjän pyytäessä uutta sivua, pyyntö välitetään *Controller*-objektille, joka huolehtii tarvittavista toimenpiteistä. Ensiksi *SessionManager*-objektilta haetaan tietoturvainformaatiokuvaus, joka on käyttäjän oikeudet kertova bittimaski. Tämä parametri salataan, joten istuntoja ei pysty muokkaamaan siten, että tietoturva vaarantuisi. Tietoturvainformaatiokuvauksen perusteella SessionManager-komponentti tarkistaa, onko käyttäjällä oikeuksia pyydettyyn sivuun. Mikäli on, selvitetään käyttäjän selain ja pyydetään *PageManager*-komponenttia hakemaan tarvittava sivupohja. Sivupohjat ovat valmiita XML-dokumentteja, joissa osa tageista sisältää tekstin sijasta erityisiä prosessointikomentoja. Kun sivupohja annetaan PageManager-komponentille, se

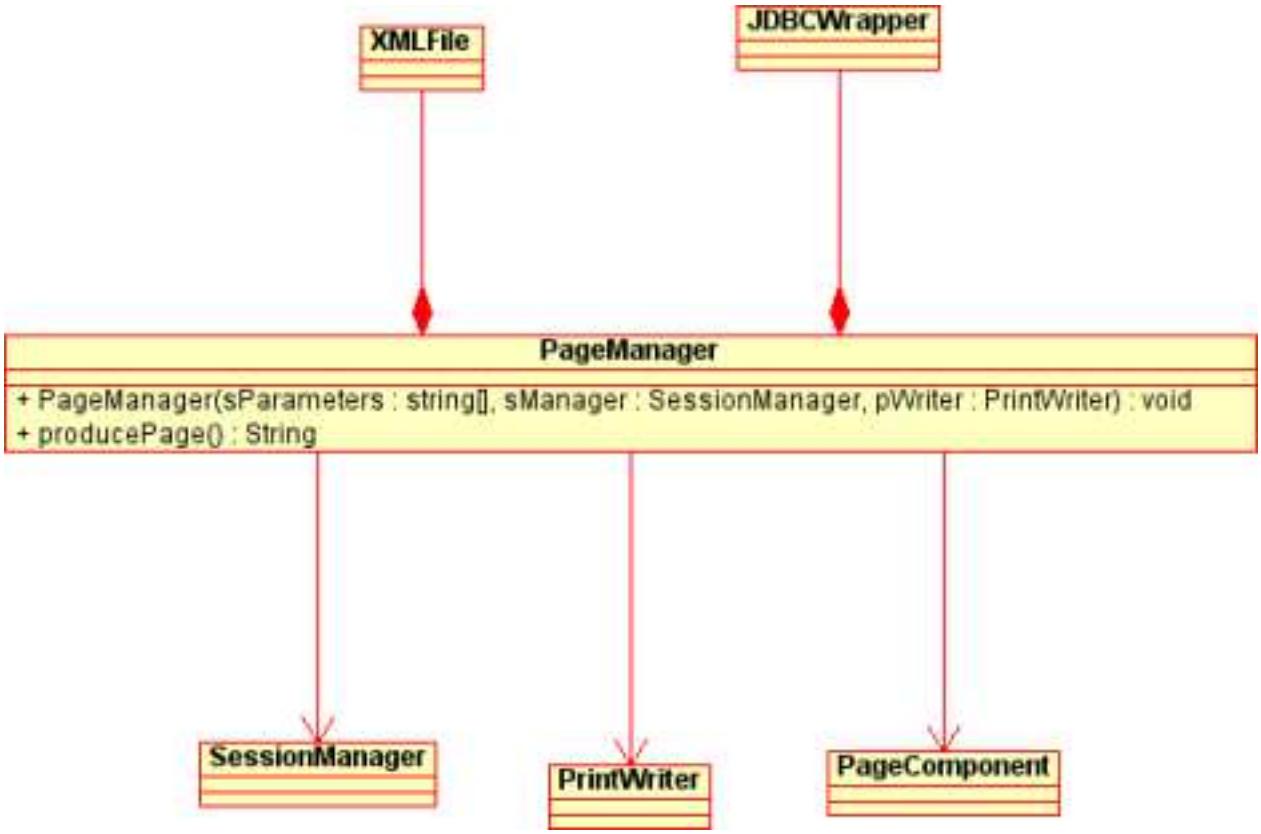


Kuva 6: Sovelluksen päätietovirta.

osaa tuottaa sivun dynaamiset osat lataamalla tarvittavat komponentit prosessointikomentojen perusteella. Sisällön tuottaminen vaatii usein tietokannan käyttöä, jonka takia *JDBCWrapper*-komponenttia saatetaan tarvita. PageManager-komponentti palauttaa Controllerille valmiin XML-muotoisen dokumentin, joka muunnetaan XHTML-muotoon kuvassa 6 esitetyllä tavalla.

5.1 PageManager

PageManager-komponentin tehtävänä on vastata ulkoasun tuottamisesta. Komponentti tuottaa sekä XML-dokumentin että lopullisen XHTML-muotoisen dokumentin, joka näytetään käyttäjälle. Käytännössä tämä siis tarkoittaa oikean sivupohjan lataamista, oikeiden komponenttien kutsumista, ulkoasutiedoston lataamista sekä ulkoasun tuottamista. PageManager-luokan UML-kuvaus on esitetty kuvassa 7.



Kuva 7: PageManager-luokan UML-kuvaus.

PageManager(Arraylist Parameters, SessionManager sManager, PrintWriter printWriter)

Metodi on luokan ainut konstruktori. Parametri *sParameters* sisältää taulukon kaikista sovelluksen käytössä olevista sovellusparametreista. Taulukossa on sekä tarvittavat

HTTP-parametrit että sovelluksen pääkonfiguraatiotiedostossa määritellyt sovellusparametrit. Parametri *sManager* on viite aikaisemmin luotuun *SessionManager*-olioon. Viite tarvitaan, jotta istuntoparametreihin päästäisiin käsiksi PageManager-luokan sisältä. Viimeinen parametri, *printWriter*, sisältää viittauksen *printWriter*-olioon, jonka avulla valmiin sivun tulostaminen onnistuu.

getParameterByName(String string):

getJDBCWrapper():

producePage():String

On luokan ainoa julkinen metodi konstruktorin lisäksi. Metodi tuottaa valmiin XHTML-sivun näytettäväksi ruudulla. Itse toiminnallisuus refaktoroidaan siten, että metodit muodostavat järkevän kokonaisuuden. Käytännössä metodi lataa XML-sivupohjan sekä tarvittavat ulkoasutiedostot, muodostaa sivun dynaamisen sisällön kutsumalla oikeita komponentteja ja lopuksi jäsentää XML- ja XSL-tiedostoista valmiin XHTML- muotoisen sivun. Lisäksi tietokantayhteys otetaan tämän metodin kutsun yhteydessä.

6 Komponenttilogiikkakerros

Komponenttilogiikkakerroksen tehtävänä on luoda sivujen dynaaminen sisältö ajon aikana. Kaikki toiminnallisuus perustuu sivupohjien joustavaan käyttöön sekä Javan *reflect*-rajapinnan ominaisuuksien tehokkaaseen hyödyntämiseen. Yksittäisessä sivupohjassa kuvataan sivun sisältö XML-muodossa. Tagien joukossa on erityisiä prosessointikäskyjä, jotka ovat muotoa <*dynamic*> olevia tageja. Tagien välissä on annettuna sen komponentin täydellinen luokkanimi (esim. *otie.component.ExampleComponent*), jolla kyseinen sisältöalue halutaan tuottaa. Komponentista luodaan uusi ilmentymä *reflect* rajapinnan avulla. Tämä tapahtuu kutsulla *Class.forName(<className>).newInstance*, joka palauttaa *Object*-luokan ilmentymän. Tämä olio voidaan suoraan muuntaa (cast) *PageComponent*-olioksi sekä kutsua *PageComponent.getContent(param1)*-metodia. Jotta tämänkaltainen lähestymistapa toimisi, seuraavia periaatteita on noudatettava: jokaisella sivukomponentilla on oltava oletuskonstruktori (ei parametrejä). Lisäksi jokaisen sivukomponentin tulee olla *pageComponent*-luokan aliluokka.

6.1 PageComponent

Komponenttilogiikkakerroksen pääluokka on abstrakti *PageComponent*-luokka, joka määrittelee yhden abstraktin *getContent*-metodin. Käytännössä PageComponent on kuitenkin erittäin tärkeä sillä kaikkien sivukomponenttien täytyy olla *PageComponent*-luokan aliluokkia. Tällä tavoin komponentit saadaan luotua ajonaikaisesti yhtenäistä rajapintaa käyttäen. PageComponent-luokan UML-kaavio on esitetty kuvassa 8.

getContent(PageManager pMan):XMLFile

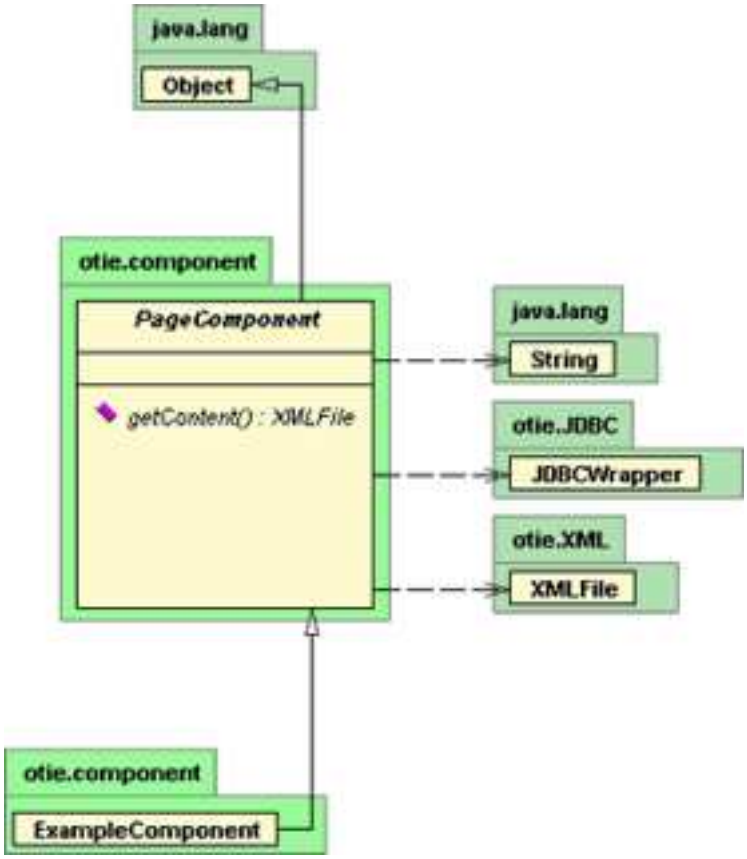
Metodi tulostaa komponentin tuottaman osan sivusta. Parametri *accessString* on viite String-taulukkoon, jossa jokainen elementti on yksi käyttäjälle määrätty turvallisuustunniste. Jos turvallisuustunniste täyttää komponentille kehitysvaiheessa määritellyt ehdot, komponentin sisältö voidaan luoda ja palauttaa merkkijonona. Koska komponentit saattavat tarvita tietokannan käyttöä, välitetään komponenteille parametrina *JDBCWrapper* luokan ilmentymä. Metodi saa parametrina myös viitteen XMLFile-tyyppiseen olioon johon se lisää omat tulosteensa.

6.1.1 LoginComponent

LoginComponent käsittelee käyttäjän sisäänkirjautumisen.

6.1.2 LogoutComponent

LogoutComponent käsittelee käyttäjän uloskirjautumisen.



Kuva 8: PageComponent-luokan UML-kaavio.

6.1.3 SearchEngineComponent

SearchComponent hoitaa käyttäjien kyselyt järjestelmän tietoihin. Komponentti tarjoaa käyttäjälle listat mahdollisista hakuehdoista SELECT ja WHERE ehtoihin valittavaksi, suorittaa käyttäjän valintojen edellyttämät kyselyt ja näyttää haun tuloksen käyttäjälle.

6.1.4 AdminComponent

AdminComponent tarjoaa liittymän järjestelmän hallintaan niille käyttäjille joille on annettu tähän oikeudet.

AdminComponentin alta käyttöliittymässä löytyvät kohdat: metriikoiden hallinta, metriikkamallien hallinta, metriikkasarjojen hallinta, henkilöiden hallinta, vaiheiden hallinta, projektien hallinta, projektimallien hallinta, projektityyppien hallinta ja ajanjaksojen hallinta. Nämä alikohdat on toteutettu kukin Beaninä.

6.2 TableComponent (Bean-luokat)

Beanit ovat tässä järjestelmässä luokkia jotka toimivat tietokannan ja käyttöliittymän välissä, hoitaen yleensä yhden tietokantataulun päivityksen.

Beanit tulostavat XML:ää, joka muunnetaan HTML:ksi ja näytetään käyttäjälle. Bean tulostaa lomakkeen lähetysnappeineen, jonka sisään se tulostaa tietokannasta haettuja tietoja, sekä syötekenttiä käyttäjälle. Bean nimeää syöttökentät niin että tunnistaa ne itse kun käyttäjä lähettää ne selaimestaan takaisin.

Esimerkiksi PersonBean hakee tietokannasta Person-taulusta mm. tiedot $kayttajatunnus$, $salasana$, $etunimi$ ja $sukunimi$, ja tulostaa ne käyttäjälle HTML:n textfield-kenttinä. Kun käyttäjä muuttaa tietoja ja lähettää lomakkeen selaimestaan, PersonBean ottaa paluuarvot vastaan ja päivittää ne tietokannan Person-tauluun.

Bean saa parametrina valmiin XMLFile-olion, johon se tulostaa sisällön. Tietokantayhteyksiin bean käyttää JDBCWrapperia, jonka ilmentymän bean saa PageManagerilta metodilla getJDBCWrapper().

Kaikkien Bean-luokkien yliluokka on TableComponent Seuraavassa kuvataan tärkeimmät metodit.

setXMLFile(XMLFile xFile)

public void setXMLFile(XMLFile xFile)

Tällä metodilla PageManager antaa Beanille XMLFile-olion johon tämä voi tulostaa sisältönsä.

createForm(String params[])

protected abstract void createForm(String[] params)

Kutsuva luokka - PageManager, on avannut XMLFile-olioon lomakkeen avaustageilla ennen kutsumista, ja pyytää tällä metodilla Beania tulostamaan sisällön lomakkeeseen. Metodissa lomakkeen kentille annetaan nimet kuten $form\_firstname$ ja $form\_lastname$.

Jotta Bean voi näyttää käyttäjälle vanhat tiedot tietokannasta nähtäväksi ja muutettavaksi, Bean saa PageManagerilta parametrina tarvitsemansa tiedot String[] params -taulukossa. Params-taulukko sisältää tiedot kyseisen Beanin käyttämästä taulusta, esim. PersonBeanille Person-taulu, ja kyseiselle id:lle, esim PersonBeanille sen käyttäjän id:llä jota muokataan.

Params[] -Taulukko sisältää tietokantataulun arvot kyseessä olevalle id:lle. Params[0] on tietokantataulun 1. attribuutin arvo, params[1] tietokantataulun 2. attribuutin arvo jne.

validateContent

public abstract void validateContent()

Tätä metodia kutsutaan kun käyttäjä lähettää tiedot selaimestaan. Metodi tarkistaa että syötteet ovat kelvolliset, ja tallettaa muutokset tietokantaan, tehden tarvittaessa INSERT, DELETE ja/tai UPDATE lauseita.

Metodi pyytää käyttäjän lähettämät tiedot PageManagerilta this.pMan.getParameterByName(String) -metodilla. Parametrin nimi on HTML-kentän nimi,jonka Bean createForm -metodissa antoi HTML-kentälle, esimerkiksi $this.pMan.getParameterByName(\text{"}form\_firstname\text{"})$.

Jos syötteet ovat virheellisiä, lisää Bean virheen errorString-muuttujaan, joka Bean-luokan ulkopuolelta näytetään käyttäjälle.

6.2.1 MetricBean

MetricBean toteuttaa metriikoiden hallinnan. MetricBeanin avulla voi luoda ja poistaa metriikoita, sekä muokata olemassa olevia. Kukin metriikka käyttää jotakin kuudesta metriikkamallista, jonka tiedon MetricBean saa HTML-lomakkeen kentästä nimeltä $form\_metricmodel$. Tämän MetricBean saa kutsumalla metodia $PageComponent.getParameterByName(\text{"}form\_metricmodel\text{"})$.

Metriikoiden yhteisiä tietoja ovat nimi, kuvaus, luomisaika ja muokkausaika. Tämän lisäksi metriikkamallia 1,2 ja 3 käyttävillä metriikoilla on muuttuja maksimipituus tai mak-

simiarvo. Metriikkamallilla 4 on nimetyt alkiot valittavaksi, esim. 1 = "vesiputous", 2="iteratiivinen"jne. Metriikkamallia 4 käyttävien metriikoiden kohdalla MetricBean tarjoaa myös näiden muokkauksen, sekä tietokantataulun MetricItemNames käsittelyn Metrictaulun lisäksi.

MetricBean luokalla on 3 eri toimintaa kutsusta riippuen. 1) käyttäjä on luomassa uutta metriikkaa: annetaan käyttäjälle lista olemassa olevista metriikkamalleista joista hän valitsee yhden. 2) käyttäjä on valinnut kohdassa 1 metriikkamallin ja lähettää sen, käyttäjälle tulostetaan kyseistä metriikkamallia käyttävä tyhjä metriikka. 3) käyttäjä muokkaa olemassa olevaa metriikkaa. Käyttäjälle tulostetaan kyseinen metriikka sen oman metriikkamallin mukaan.

6.2.2 PersonBean

PersonBeanin kautta voi lisätä ja poistaa käyttäjiä ja muokata näiden tietoja ja oikeuksia.

6.2.3 ProjectTypeBean

ProjectTypeBeanin kautta voi lisätä, poistaa ja muokata projektityyppejä, kuten ’informaatiojärjestelmät’.

6.2.4 PhaseBean

PhaseBeanin kautta muokataan järjestelmän tuntemia projektin vaiheita, kuten ‘suunnittelu’ ja ’määrittely’.

6.2.5 SemesterBean

6.2.6 ProjectBean

ProjectBeanin kautta luodaan ja muokataan projekteja, sekä liitetään niihin opiskelijoita, ohjaajia ja vastuuhenkilöitä.

6.2.7 MetricSerieBean

MetricSerieBeanin kautta luodaan ja muokataan metriikkasarjoja, sekä liitetään niihin metriikoita.

6.2.8 ProjectModelBean

ProjectModelBeanin kautta lisätään järjestelmään projektimalleja, kuten ’vesiputous’ ja ’iteratiivinen’, sekä muokataan niitä.

7 Tietokantakerros

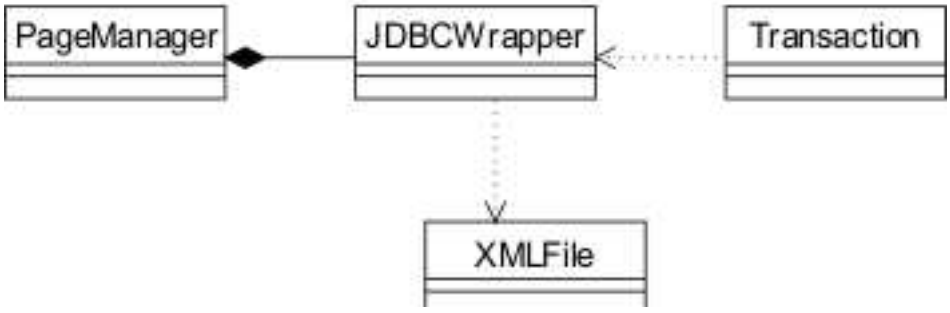
Tietokantakerroksen tehtävät ovat kahdenlaisia. Monet kerroksen luokista toimivat yleisluontoisena kirjastona tietokantaoperaatioita varten ja pyrkivät helpottamaan siirtymistä tietokantajärjestelmien välillä. Toisaalta tietokantakerros tarjoaa myös "kuvaustason", jossa tietokannan rakenne kuvataan olio-kielelle (Object-relational mapping). Tämän kaltainen toiminnallisuus toteutetaan yleensä (Java-kielessä) papujen (Bean) avulla, joten ryhmämme päätti nimetä luokkansa analogisesti vallitsevan käytännön kanssa. Koska papujen ajaminen vaatii Java Enterprise palvelinta, ei sovelluksessa käytetä varsinaisia Java papuja vaan samankaltainen toiminnallisuus on saavutettu ohjelmoimalla tavallisia Java-luokkia. Haittapuolena tässä on se, että luokat eivät ole yhtä hyvin siirrettävissä kuin pavut.

Kerros koostuu kolmesta eri rajapinnasta sekä niiden alaluokista. Kaikista keskeisin luokka on *JDBCWrapper*, joka vastaa mm. tietokantayhteyden muodostamisesta. Lisäksi *JDBCWrapper* tarjoaa joukon apumetodeja, joiden avulla kyselyjä voidaan muodostaa suoraan taulukkomuotoisista parametreistä.

Toinen rajapinnoista on *Transaction* rajapinta/kirjasto, joka tarjoaa metodeja transaktioiden, eli jakamattomien operaatiosarjojen, suorittamiseen. Transaktiorajapinta on suunniteltu siten, että siirtyminen tietokannan tarjoajasta toiseen on suhteellisen kivutonta. Järjestelmän projektin tuloksena tukemat tietokannanhallintajärjestelmät ovat MySQL ja Oracle. Tuetut tietokantaoperaatiot ovat INSERT, UPDATE, DELETE sekä erityinen INSERT_KEY, jossa suoritetaan INSERT-operaatio siten, että halutaan tuloksena syntyvä uusi auto_increment / SEQUENCE sarakkeen arvo talteen.

Kolmas tietokantakerroksen tukema rajapinta on ns. Bean-rajapinta. Kaikkia staattisia (pysyviä) tietoja sisältäviä tauluja kohden on toteutettu yksi Bean-luokka, joka periytyy yhteisestä TableComponent luokasta. Jokainen Bean-luokka vastaa täten yhden tietokannan taulun hallinnasta.

Yksinkertainen UML-kuvaus tietokantakerroksen, ja erityisesti JDBCWrapper-luokan, roolista järjestelmässä on esitetty kuvassa 9.



Kuva 9: Tietokantakerroksen luokkien suhde muihin järjestelmän komponentteihin

7.1 JDBCWrapper

JDBCWrapper on tietokantakerroksen keskeisin osa. Käytännössä kyse on erillisestä rajapinnasta, joka on rakennettu Javan JDBC API:n päälle ja joka tarjoaa sovelluksen tarkoituksiin räätälöityjä apumetodeja ja näiltä osin JDBCWrapper voidaan tulkita kirjastoluokaksi. JDBCWrapper on kuitenkin myös luokka, joka edustaa (avointa) tietokantayhteyttä: jokaisen sivupyynnön alussa luodaan uusi JDBCWrapper olio, joka avaa tietokantayhteyden tietokantaan. JDBCWrapper olion kautta on sitten mahdollista suorittaa tietokantakyselyjä kirjastofunktioluokan tavoin.

JDBCWrapper luokka hyödyntää tietokantakuvaustiedostoa (database deployment descriptor), joka mahdollistaa luokan helpon käyttämisen uusissa sovelluksissa. Tietokantakuvaustiedosto voidaan tulkita vastaavan komponenttiteknologiassa käytettyjä asennus metadatatiedostoja (deployment descriptor), jotka perinteisesti ovat XML muotoisia. Jotta luokan käyttäminen olisi mahdollista myös laajemmissa business sovelluksissa, tietokantakuvaus on XML-muotoinen.

JDBCWrapper luokka hyödyntää kahta eri kirjastoa. Kaikki tietokantatoiminnot rakentuvat suoraan Javan JDBC API:n päälle. Tietokantakuvaustiedoston käsittelyssä hyödynnetään otie.XML paketin luokkia *XMLFile* ja *XMLObject*.

JDBCWrapper(String configurationFile) throws InvalidDeploymentDescriptorError, ClassNotFoundException, InstantiationException, IllegalAccessException

Oletuskonstruktori: lukee konfiguraatiotiedoston sekä muodostaa tietokantayhteyden. Jos konfiguraatiotiedosto ei ole oikeanlainen, metodi aiheuttaa *InvalidDeploymentDescriptorError* virheen. Jos konfiguraatiotiedostossa määriteltyä ajuritiedostoa ei voida ladata, metodi aiheuttaa *InstantiationExceptino* poikkeuksen. Jos ajuritiedostoa ei löydy, metodi aiheuttaa *ClassNotFoundException* poikkeuksen. Jos nykyinen turvallisuuspolitiikka ei salli ajuriluokan ilmentymien luomista, metodi aiheuttaa *IllegalAccessException* poikkeuksen. Kolme viimeksi mainittua ovat standardit Java Reflect API:n poikkeustilanteet.

getConnection():Connection

Metodi palauttaa viitteen java.sql.Connection olioon. Jos yhteys on avoin, viite on validi yhteysobjekti. Muuten metodi palauttaa **null** arvon.

alterQuery(String query):String

Metodi poistaa kyselylausekkeista luvattomia merkkejä sekä muuntaa erikoismerkit oikeaan muotoonsa (lisää merkin niiden eteen). Metodia kutsutaan automaattisesti muiden paitsi execute() kyselymetodien yhteydessä.

query(String query): ArrayList throws ConnectionException

Metodi tekee tietokantakyselyn ja palauttaa tulokset ArrayList- oliossa. Jokainen ArrayList-olion alkio on muotoa String[], missä String[] taulukko vastaa yhtä tulosriviä ja taulukon jokainen alkio vastaa yhtä tulosrivin saraketta. Eli esim. kyselyn *SELECT id, name FROM sometable* tulokset olisi esitetty String[2]-olioina, joissa indeksin 0 parametri vastaa id arvoa ja indeksin 1 määrittämä parameteri name arvoa. Jos tietokantayhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen.

execute(String query): int throws ConnectionException, SQLException

Metodi suorittaa INSERT, UPDATE, DELETE kyselyn tai vaihtoehtoisesti metodin avulla voidaan suorittaa ns. DDL kyselyjä, joita ovat mm. CREATE TABLE, CREATE INDEX, ALTER TABLE jne. DDL tyyppisiä kyselyjä ei voi suorittaa muiden metodien avulla. Metodi palauttaa rivien määrän, johon kysely vaikutti. Mikäli tietokanta yhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen. Mikäli kysely *qString* ei ole validi tietokantalause/kysely, metodi aiheuttaa *SQLException* poikkeuksen.

getQueryIterator(String query):Iterator throws ConnectionException

Metodi suorittaa tietokantakyselyn ja palauttaa java.util.Iterator olion tulosriveihin. Jos yhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen.

insert(String tableName, String[] Names, String[] Values):int throws ConnectionException, InvalidSQLParameterException

Metodi suorittaa SQL INSERT operaation, joka on muotoa INSERT into *tableName* (*Names*) VALUES *Values*. Metodi palauttaa lisättyjen rivien määrän. Jos tietokanta yhteydessä on vikaa, metodi aiheuttaa *ConnectionException* poikkeuksen. Jos *Names* ja *Values* taulukot ovat erikokoiset, metodi aiheuttaa *InvalidSQLParameterException* poikkeuksen.

insert(String tableName, String[] Names, String[] Values, boolean[] useQuotes):int throws ConnectionException, InvalidSQLParameterException

Metodi, joka suorittaa insert kyselyn. Metodi on muuten sama kuin edellinen insert metodi, paitsi että nyt käyttäjä voi määrittää boolean[] taulukon, joka kertoo metodille, tuleeko ’-merkkiä käyttää parametrien yhteydessä.

delete(String tableName, String[] Names, String[] Values):int throws ConnectionException, InvalidSQLParameterException

Metodi suorittaa SQL DELETE -operaation, joka on muotoa DELETE FROM *tableName* WHERE *Names[0] = Values[0]* $\cdots$ AND *Names[Names.length-1 = Values[Values.length-*

1]. Metodi palauttaa poistettujen rivien lukumäärän. Jos tietokantayhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen. Jos *Names* ja *Values* taulukot ovat eri pituiset, metodi aiheuttaa *InvalidSQLParameterException* poikkeuksen.

private getStatement():Statement

Metodi palauttaa uuden Statement-olion, jonka avulla kyselyjä voidaan suorittaa. Koska jokainen JDBC API:n kautta tehtävä kysely vaatii Statement oliota, oli järkevää eristää tämä toiminnallisuus erilliseksi metodiksi.

private updateQuery(String qString):int

Metodi on varsinainen UPDATE, INSERT ja DELETE -kyselyjen suorittaja. Muut metodit muodostavat kyselylausekkeen ja kutsuvat tätä metodia, joka palauttaa vastaavan integer arvon. Eli UPDATE-kyselyissä metodi palauttaa muutettujen rivien lukumäärän, INSERT-kyselyissä palautetaan lisättyjen rivien lkm. ja DELETE-kyselyjen yhteydessä palautetaan poistettujen rivien lkm.

delete(String tableName, String cond):int throws ConnectionException

Metodi suorittaa SQL DELETE kyselyn, joka on muotoa DELETE FROM *tableName* WHERE *cond*. Metodi palauttaa integer arvon, josta käy ilmi poistettujen rivien lukumäärä. Jos yhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen.

update(String tableName, String[] Names, String[] Values,String[] whereNames, String[] whereCond):int throws ConnectionException, InvalidSQLParameterException)

Metodi suorittaa UPDATE sql-kyselyn tauluun *tableName*. Taulun parametrien *Names* uusiksi arvoiksi asetetaan parametrin *Values* määrittämät arvot. Päivitys kohdistetaan niihin riveihin, joiden *whereNames* sarakkeet täyttävät ehdon *whereCond*. Metodi palauttaa muutettujen rivien lukumäärän. Jos yhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen. Jos *Names* ja *Values* taulukot ovat eri kokoiset, metodi aiheuttaa *InvalidSQLParameterException* virheen.

update(String tableName, String[] Names, String[] Values,String cond): int throws ConnectionException, InvalidSQLParameterException

Metodi suorittaa UPDATE sql-kyselyn tauluun *tableName*. Taulun parametrien *Names* uusiksi arvoiksi asetetaan parametrin *Values* määrittämät arvot. Päivitys kohdistetaan niihin riveihin, jotka täyttävät WHERE SQL ehdon *cond*. Metodi palauttaa muutettujen rivien lukumäärän. Jos yhteys ei ole avoin, metodi aiheuttaa *ConnectionException* poikkeuksen. Jos *Names* ja *Values* taulukot ovat eri kokoiset, metodi aiheuttaa *InvalidSQLParameterException* virheen.

close()

Metodi sulkee tietokantayhteyden. Tätä metodi kutsutaan jokaisen sivupyynnön käsittelyn lopuksi tai siinä vaiheessa, kun ollaan varmoja, että tietokantayhteyttä ei enää tarvita.

newTransaction(TransactionType[] transactionElements) throws SQLException:Transaction

Metodi luo uuden *Transaction* luokan ilmentymän ja liittää siihen *transactionElements* parameterissa määritellyt operaatiot.

getVendor():String

Metodi palauttaa tietokannan hallintajärjestelmän tarjoojan/myyjän nimen. Esim. käytettäessä MySQL-tietokantaa 4.0.20, metodi palauttaa *MySQL*. Vastaavasti Oraclen version 9.0.2i thin-ajuria käytettäessä metodi palauttaa *Oracle*. Näitä merkkijonoja ei ole mitenkään käsitelty, vaan kyseessä on valmistajien itse määrittelemät merkkijonot.

querytoXML(String query):XMLFile throws ConnectionException

Metodi suorittaa tietokantakyselyn ja palauttaa tulokset XML-muodossa. XML-tagit ovat toteutettavassa versiossa sidottu ulkoasutiedostossa määritettäviin käännössääntöihin, joten metodin tuottama XML-sisältö on seuraavassa määriteltyä muotoa:

```
<stdtable>
<newrow>
<newcol>COL_1</newcol>
<newcol>COL_2</newcol>
...
<newcol>COL_N</newcol>
</newrow>
</stdtable>
```

Metodi palauttaa siis XML-dokumentin esiintymän, jossa elementtipuu on rakennettu edellä kuvatulla tavalla. Tagi <stdtable> on siis palautettavan XML-dokumenttiolion juurielementti.

7.2 Transaction

Kuvassa 10 esitetään Transaction rajapinnan rakenne UML muodossa. Keskeiset luokat ovat Transaction ja TransactionType. Transaction luokan ilmentymät (oliot) ovat joukkoja SQL operaatioita, jotka halutaan suorittaa atomisesti eli jakamattomasti. Transaction

rajapinnat operaatiot esitetään TransactionType luokan avulla. TransactionType määrittelee yhteisen yläluokan, jonka muut operaatiot perivät. Koska osa operaatioista toteutetaan eri tavalla eri tietokannoissa, tarvitaan erillisiä adaptereita, jotka määrittävät yhteisen rajapinnan eri tietokannalle. OhtuTie-projektin yhteydessä toteutetaan riisuttu version transaction rajapinnasta, joka tukee operaatioita INSERT, DELETE ja UPDATE-MySQL ja Oracle-tietokannalle. Lisäksi toteutetaan erillinen InsertKey operaatio, joka vastaa INSERT-operaatiota, joka palauttaa avan kentän viimeisimmän arvon. Varsinainen Transact-luokka toteutaan Java-rajapintana (interface).

public void execute() throws SQLException;

Metodi transaktion suorittamiseksi.

public void commit() throws SQLException;

Metodi transaktion vahvistamiseksi. Operaatioiden tulokset ovat näkyviä vasta, kun transaktio on vahvistettu.

public void rollback() throws SQLException;

Metodi transaktion peruuttamiseksi. Peruttaa akaikki tehdyt operaation. JDBC kirjasto vaatii tukea transaktioille, joten kaikkien JDBC yhteensopivien tietokantojen tulisi tukea myös rollback metodia. Kuitenkin MySQL ei perusversiossa tue transaktioita, joten MySQL kutsu ei varsinaisesti tee mitään.

public TransactionMetadataSet getMetadata();

Metodi metadatakuvauksen saamiseksi transaktiosta. Luokka *TransactionMetadataSet* määrittää tietuemaisen Java-luokan, johon tallennetaan tietoa mmm. lisättyjen rivien lukumäärästä. Metadatan avulla voidaan tarkistaa, että transaktio onnistui.

7.3 MySQLTransaction:implements Transaction

7.4 OracleTransaction:implements Transaction

public MySQLTransaction(TransactionType[] transactType, JDBCWrapper jWrap) throws SQLException

public OracleTransaction(TransactionType[] transactType, JDBCWrapper jWrap) throws SQLException

Molemmat rajapinnan toteuttavat luokat ovat toiminnoiltaan identtiset. Tämän takia molemmat luokat kuvataan peräkkäin. Konstruktorin tehtävänä on määrittää yksittäiset kyselyt, joita transaktioissa käytetään. Esim. INSERT_KEY koostuu yhdestä SELECT ja yhdestä INSERT operaatiosta, joten nämä operatiot selvitetään ja valmistellaan. Valmistelussa käytetään JDBC:n *PreparedStatement* rakennetta.

7.5 TransactionFactory

TransactionFactory on tehdasluokka, jonka tehtävän on luoda oikean tyyppinen Transaction rajapinnan toteuttava luokka tietokannanhallintajärjestelmän tarjoajan nimen perusteella. OhtuTie-projektin yhteydessä toteutetaan sekä MySQL että Oracle rajapinnat.

static getInstance(JDBCWrapper jWrap, TransactionType[] transactType): Transaction throws SQLException

Metodi luo uuden Transaction rajapinnan toteuttavan luokan ilmentymän annetun tietokantatoimittajan nimen perusteella. Metodi kutsuu uuden luokan konstruktoria, jossa liitetään *transactType* parametrin määrittämät transaktioelementit transaktioon. Jos transaktioelementtejä ei pystytä liittämään, metodi aiheuttaa *SQLException* poikkeuksen.

7.6 abstract: TransactionType

TransactionType luokka on abstrakti yläluokka, jonka muut transaktiotyypit perivät. Luokassa määritetään parametri *String[] qString*, jota muiden luokkien oletetaan käyttävän kyselylausekkeen muodostukseen. Lisäksi java.lang.Object metodin toString ylikuormitetaan ja täten tarjotaan yhteinen rajapinta merkkijonoesitysten tuottamiselle kyselyistä.

toString():String

Muodostaa hyvin muotoillun merkkijonoesityksen taulukon qString parametrien arvoista.

String[] getQuery()

Palauttaa String[] taulukon, joka sisältää qString[] parametreihin määritetyt arvot.

7.7 Insert

Insert-luokka edustaa abstraktia INSERT SQL-kyselyä, joka on muotoa INSERT *attributename* FROM *sometable* WHERE *clauses*; Insert luokka on luokan *TransactionType* alaluokka ja voi siten toimia yhtenä kyselyn osana transaktiossa.

Insert-luokassa on kolme attribuuttikenttää. String tableName muuttuja pitää sisällään sen taulun nimen, johon tämä Insert operaation ilmentymä kohdistuu. String[] taulukossa *names* on lisättävien attribuuttien nimet ja taulukossa String[] *values* on tallennettuna *names* taulukossa määritellyille attribuuteille annettavat uudet arvot.

Insert(String tablename, String[] names, String[] values)

Insert-luokan oletuskonstruktori. Ensimmäinen parametri määrittää taulun, johon lisäysoperaatio kohdistuu. Toinen parametri, names, määrittää niiden attribuuttien nimet, joiden arvot asetetaan lisäys operaatiossa. Viimeinen parametri, values, kertoo arvot, jotka names taulukossa määritellyille attribuuteille annetaan.

getTableName():String

Metodi palauttaa taulun nimen, joka on määritetty tälle Insert-operaation ilmentymälle.

getNames(): String[]

Metodi palauttaa Insert-operaatiossa asetettavien attribuuttien nimet String[] taulukossa.

getValues(): String[]

Metodi palauttaa Insert-operaatiossa names taulukon määräämille attribuuteille operaatiossa asetettavat arvot.

buildQuery()

Metodi muodostaa Insert-luokan parametreistä hakulausekkeen merkkijonoesityksen. Metodi asettaa merkkijonon arvonkentälle *qString[]*, jonka Insert-luokka perii TransactionType-luokalta.

7.8 Delete

Aivan kuten luokka Insert, myös Delete edustaa abstraktia DELETE SQL-kyselyä. SQL-kyselynä DELETE on muotoa DELETE FROM *tablename* where *wherecond*. Delete luokka on luokan TransactionType alaluokka ja siten Delete-luokan ilmentyvät voivat olla operaationa *Transaction*-luokan edustamissa transaktioissa.

Delete-luokalla on kolme attribuuttia: attribuutin *tableName* arvona on sen taulun nimi, johon kyseinen operaatio kohdistuu. Attribuutin *names[]* arvona on niiden parametrien nimet, joita käytetään kyselyn WHERE osassa. Vastaavasti *values[]* taulukon arvot edustavat *names[]* taulukossa määriteltyjen attribuuttien arvoja.

Delete(String tableName, String[] Names, String[] Values)

Luokan Delete oletuskonstruktori. Luo uuden Delete operaation, joka kohdistuu tauluun *tableName*. DELETE-Lausekkeen WHERE osassa käytettään ehtona Names[] ja Values[] taulukkojen määräämiä arvoja.

getTableName():String

Palauttaa sen taulun nimen, johon tämä delete-operaation kohdistuu.

getNames():String[]

Palauttaa kyselyn WHERE osassa käytettävien attribuuttien nimet.

getValues():String[]

Palauttaa kyselyn WHERE osassa taulukon *names[]* attribuuttien nimiä vastaavat arvot delete kyselyssä.

buildQuery()

Metodi muodostaa varsinaisen kyselylausekkeen. Hakulausekkeen arvo asetetaann kentän *String[] qString arvoksi*. Delete perii kyseisen kentän luokalta TableComponent.

7.9 SimpleDelete

Luokka SimpleDelete edustaa abstraktio SQL-DELETE operaatiota. SimpleDelete on hieman yksinkertaisempi muodostaa kuin Delete, joten SimpleDelete-luokkaa kannattaa käyttää esim. silloin, kun delete operaatio kohdistuu suoraan yhteen riviin yksikäsitteisen avainarvon perusteella. SimpleDelete-kysely on muotoa DELETE FROM *tablename* WHERE *cond*. Eli erotukseksi Delete metodista, nyt koko WHERE avainsanan jälkeinen

osa annetaan yhtenä parametrinä. SimpleDelete on TransactionType-luokan aliluokka, joten luokkaa voi käyttää Transaction luokalle määritettävien operaatioiden joukossa.

SimpleDelete(String tableName, String cond)

Metodi luo uuden DELETE kyselyn, joka kohdistuu tauluun *tableName* ja jossa tuhotaan kaikki ehdon *cond* täyttävät rivit.

getTableName():String

Metodi palauttaa operaatiolle määritetyn taulun nimen.

getCondition():String

Metodi palauttaa WHERE avainsanan jälkeisen *cond* ehdon.

protected: buildQuery()

Metodi rakentaa varsinaisen kyselylausekkeen, joka on siis muotoa *DELETE FROM tablename WHERE cond*.

7.10 Update

Update luokka edustaaa abstraktia SQL-UPDATE-operaatiota. Update- luokka on luokan *TransactionType* aliluokka, joten sitä voidaan käyttää transaktion *Transaction* operaationa. Update-kysely on muotoa UPDATE *tablename* SET *names = params* WHERE *wherecond*, joten luokalle on välttämättä enemmän parametrejä kuin muilla Transaktiotyypeillä.

Update-luokalla on yhteensä neljä parametriä. Parametri *tableName* kertoo, mihin tauluun kyseinen UPDATE operaatio kohdistuu. Taulukot *Names* ja *Values* määrittävät SQL-kyselyn SET ja WHERE avainsanojen välisen osan. Parametri *cond* puolestaan määrittää kyselyn loppuehdon, joka tulee SQL-kyselyssä heti WHERE avainsanan jälkeen.

Update(String tableName, String[] Names, String[] Values, String cond)

Luokan Update oletuskonstruktori. Luo uuden Update-operaation, joka kohdistuu tauluu *tableName* ja jonka SET WHERE osiot määräytyvät taulukkojen *Names[]* ja *Values[]* perusteella siten, että Names[] taulukon attribuuttinimiä vastaavat arvot ovat taulukon Values[] vastaavassa kohdassa. Metodin viimeinen parametri, cond, puolestaan kertoo ehdon, jolla päivitysoperaatio suoritetaan.

getTableName():String

Metodi palautaa sen taulun nimen, johon UPDATE-operaatio kohdistuu.

getNames():String[]

Metodi palauttaa taulukon, jossa on kyselyn SET WHERE -avainsanojen välissä olevien kenttien/sarakkeiden nimet.

getValues():String[]

Metodi palauttaa taulukon, jossa on kyselyn SET WHERE -avainsanojen välissä olevien kenttien/sarakkeiden nimiä vastaavat arvot.

buildQuery()

Metodi muodostaa varsinaisen kyselylausekkeen. Hakulausekkeen arvo asetetaann kentän *String[] qString arvoksi*. Delete perii kyseisen kentän luokalta TableComponent.

7.11 InsertKey

Luokka InsertKey edustaa SQL-kyselyä, jossa hyödynnetään automaattista avainkentän arvoa. Koska tämänkaltaisten kyselyjen syntaksi eroaa tietokannanhallintajärjestelmän perusteella, tarvitaan yhteinen rajapinta, jonka kautta InsertKey-objektit määritetään. Tällä hetkellä InsertKey on vain tyhjä rajapinta, mutta se on luotu mahdollisia tulevaisuuden laajennuksia silmälläpitäen. Luokalla on yksi ainoa metodi: Tyhjä konstruktori.

7.12 MySQLInsertKey

7.13 OracleInsertKey

OracleInsertKey(Insert[] insert)

Sekä OracleInsertKey että MySQLInseryKey määrittävät vain samanlaisen konstruktorin, jota kutsutaan *InsertKeyFactory.getInstance()* metodista. Konstruktorin tarkoituksena on muodostaa tarvittavat INSERT ja SELECT kyselyt, jotta *insert* parametrin lausekkeet voidaan lisätä tietokantaan ja jotta niissä määritetyt avainarvot saataisiin talteen. Toiminnallisuus on siis molemmissa luokissa sama, mutta metodien yksityiskohdat poikkeavat tietokantajärjestelmien erojen takia.

7.14 InsertKeyFactory

InsertKeyFactory on luokka, jonka avulla saadaan luotua oikean tyyppinen InsertKey kysely sen mukaan, minkä valmistajan tietokannanhallintajärjestelmä on käytössä. Valmistajan nimen saa selville java.sql.Connection kautta, joten Ohtutie sovelluksessa valmistajan tunnus saadaan selville *JDBCWrapper* luokan avulla.

static getInstance(Insert[] insert, String vendorName):InsertKey

Tehdasmetodi, joka palauttaa InsertKey tyyppisen Transaktio-operaation. Toteutetussa versiossa tuetaan kahta tietokannan hallintajärjestelmää: MySQL ja Oracle.

7.15 TransactionMetadataSet

TransactionMetadataSet luokkaa on transaktioiden yhdeyssä suoritettavien operaatioiden "tunnuslukujen"säilytyspaikka, jonka avulla transaktion tuloksia voidaan hyödyntää jälkikäteen. Käytännössä TransactionMetadataSet luokkaa käyttävät Bean ja Component oliot, jotka tarkistavat, lisättiinkö tietokantaan oikeamäärä rivejä jne.

Transaktio luo TransactionMetadataSet olion execute metodin suorituksen yhteydessä kuitenkin ennen commit metodin suoritusta. Täten metadatan avulla voidaan tarkistaa, milloin transaktio tulee perua ja milloin sitoa.

TransactionMetadataSet(int insertedRows, int deletedRows, int updatedRows, ResultSet[] selectedItems, String[] keys)

Oletuskontruktori: luo uuden metadatakokoelman ja liittää siihen parametreinä välitetyt tunnusluvut.

getItems():ResultSet[]

Palauttaa SELECT kyselyjen tuottamat ResultSet oliot. Kaikki SELECT kyselyjen tuottamat tulosrivit ovat saatavilla operaation jälkeen.

getCountUpdatedRows(): int

Metodi palauttaa transaktion päivittämien rivien lukumäärän.

getCountDeletedRows():int

Metodi palauttaa transaktio poistamien rivien määrän.

getCountInsertedRows(): int

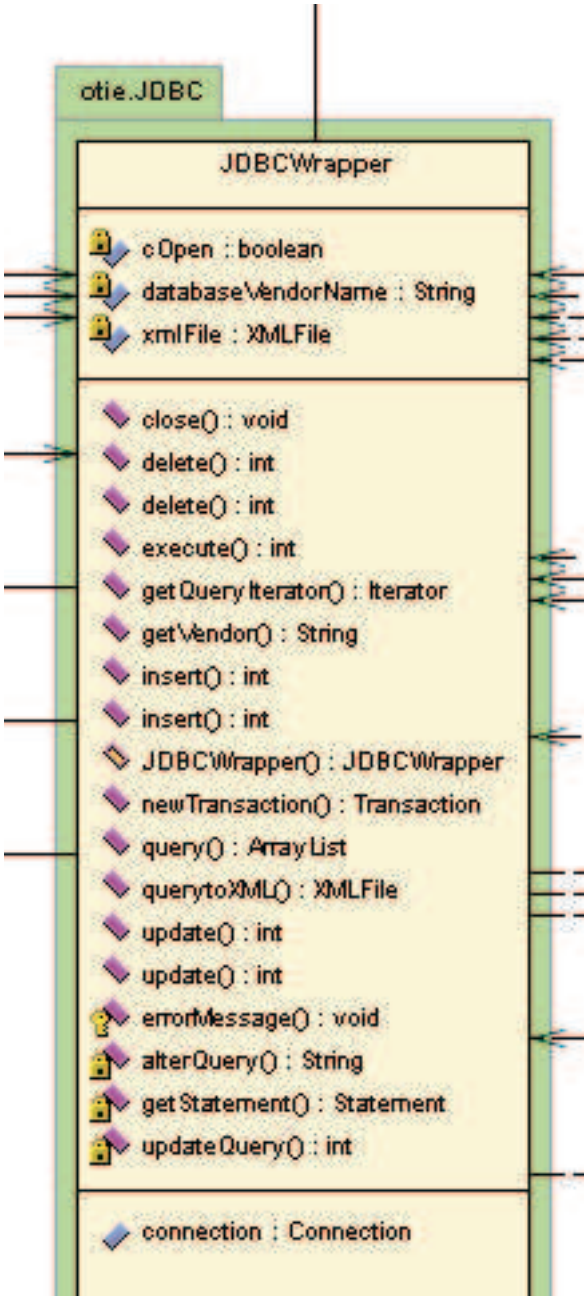
Metodi palauttaa transaktion lisäämien rivien määrän.

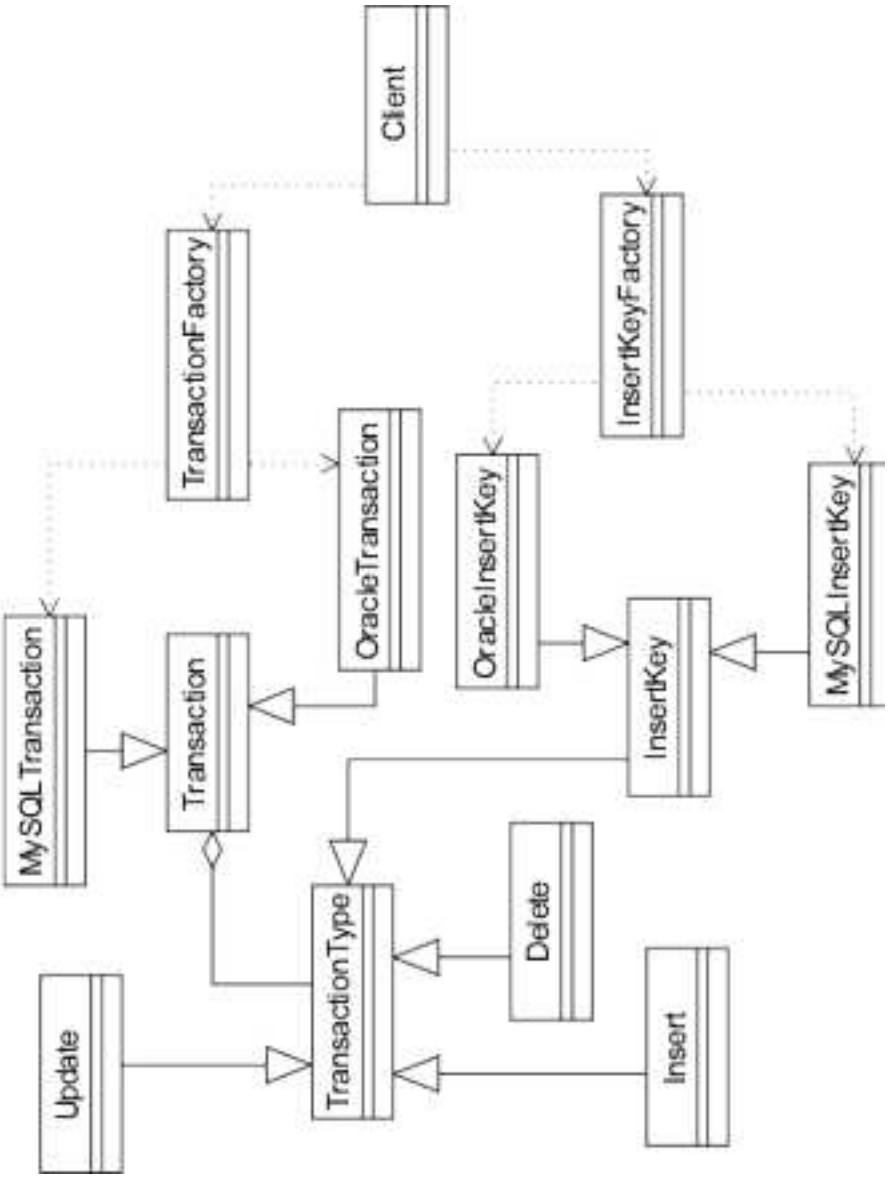
getKeys():String[]

Metodi palauttaa String[] taulukon, jossa on transaktion tuottamien (automaattisten) avain-sarakkeiden arvot. Nämä arvot muodostuvat osana Transaktion INSERT_KEY tyyppisiä kyselyjä.

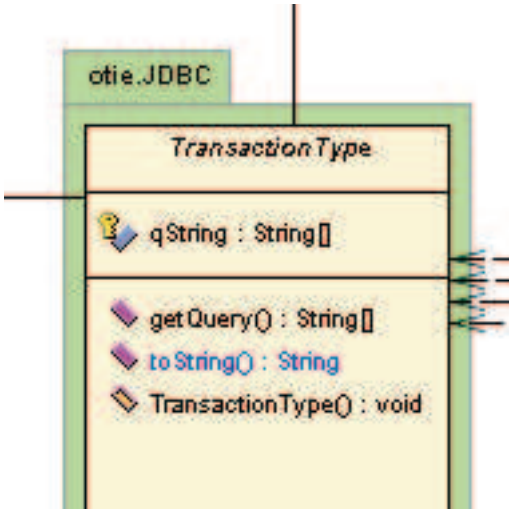
toString():String

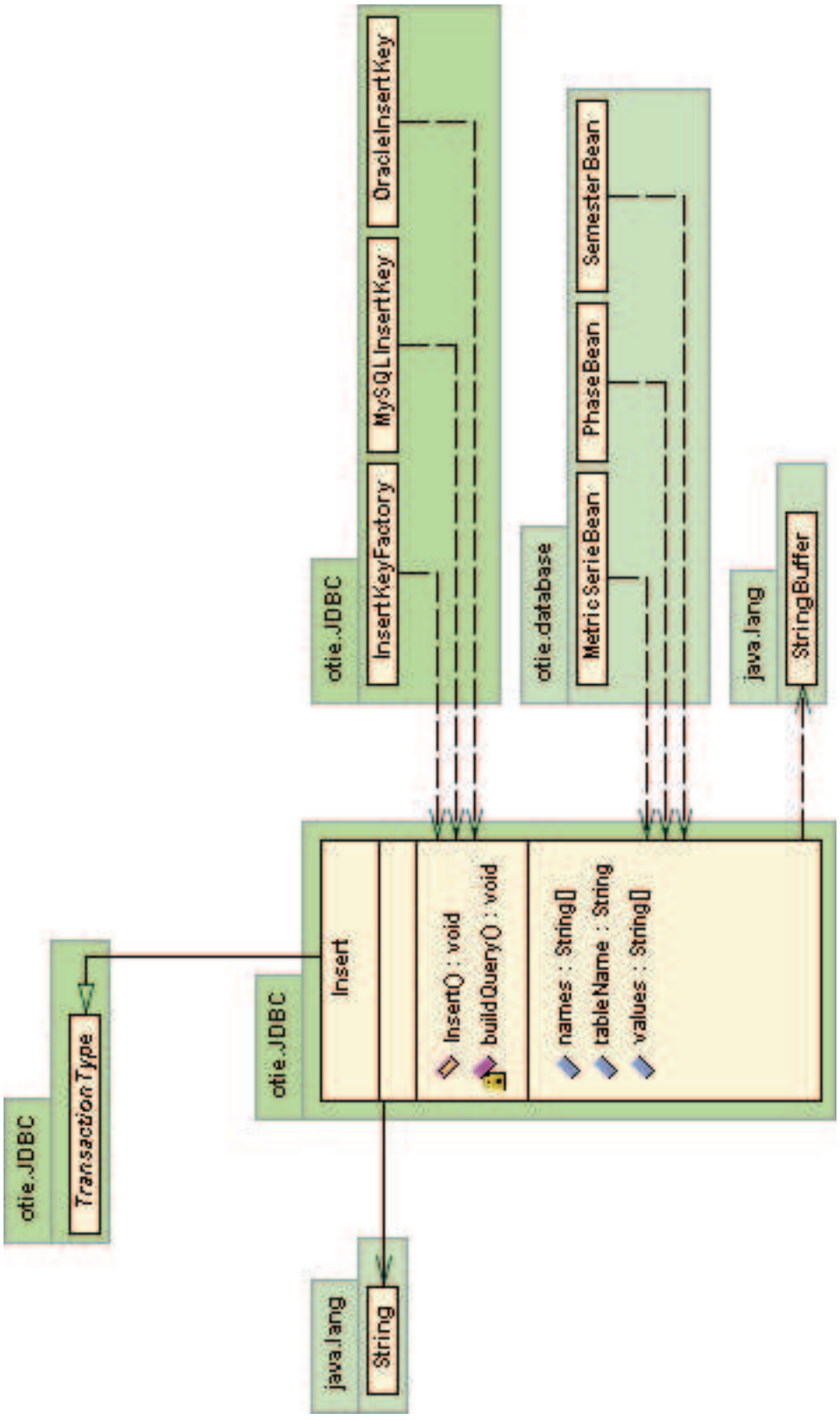
Metodi palauttaa valmiiksi muotoillun merkkijonoesityksen metadatan sisällöstä.

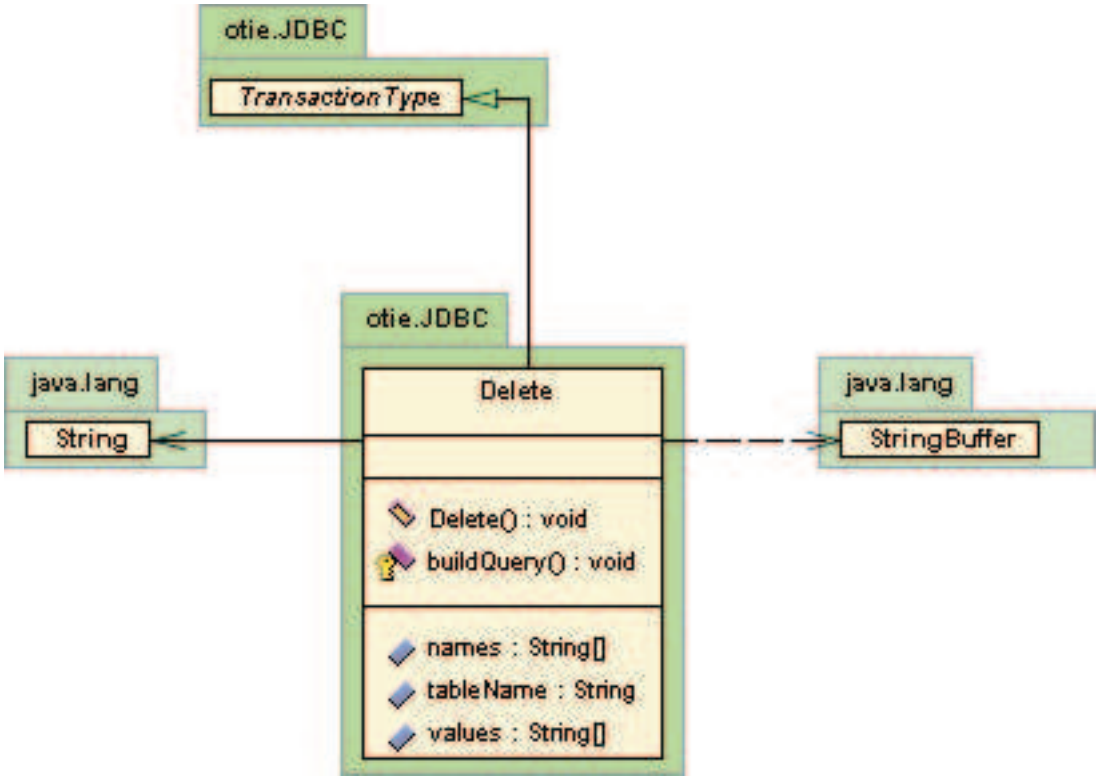

otie.JDBC
JDBCWrapper
cOpen : boolean
databaseVendorName : String
xmlFile : XMLFile
close() : void
delete() : int
delete() : int
execute() : int
getQueryIterator() : Iterator
getVendor() : String
insert() : int
insert() : int
JDBCWrapper() : JDBCWrapper
newTransaction() : Transaction
query() : ArrayList
querytoXML() : XMLFile
update() : int
update() : int
errorMessage() : void
alterQuery() : String
getStatement() : Statement
updateQuery() : int
connection : Connection

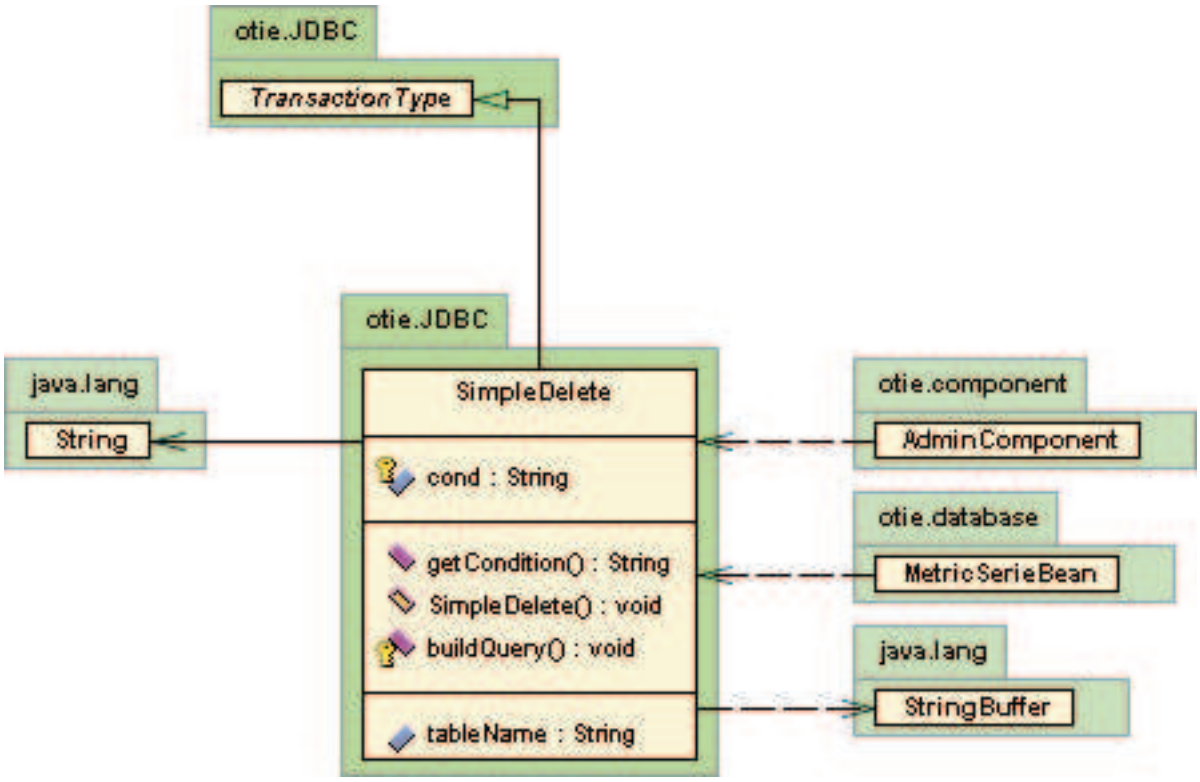


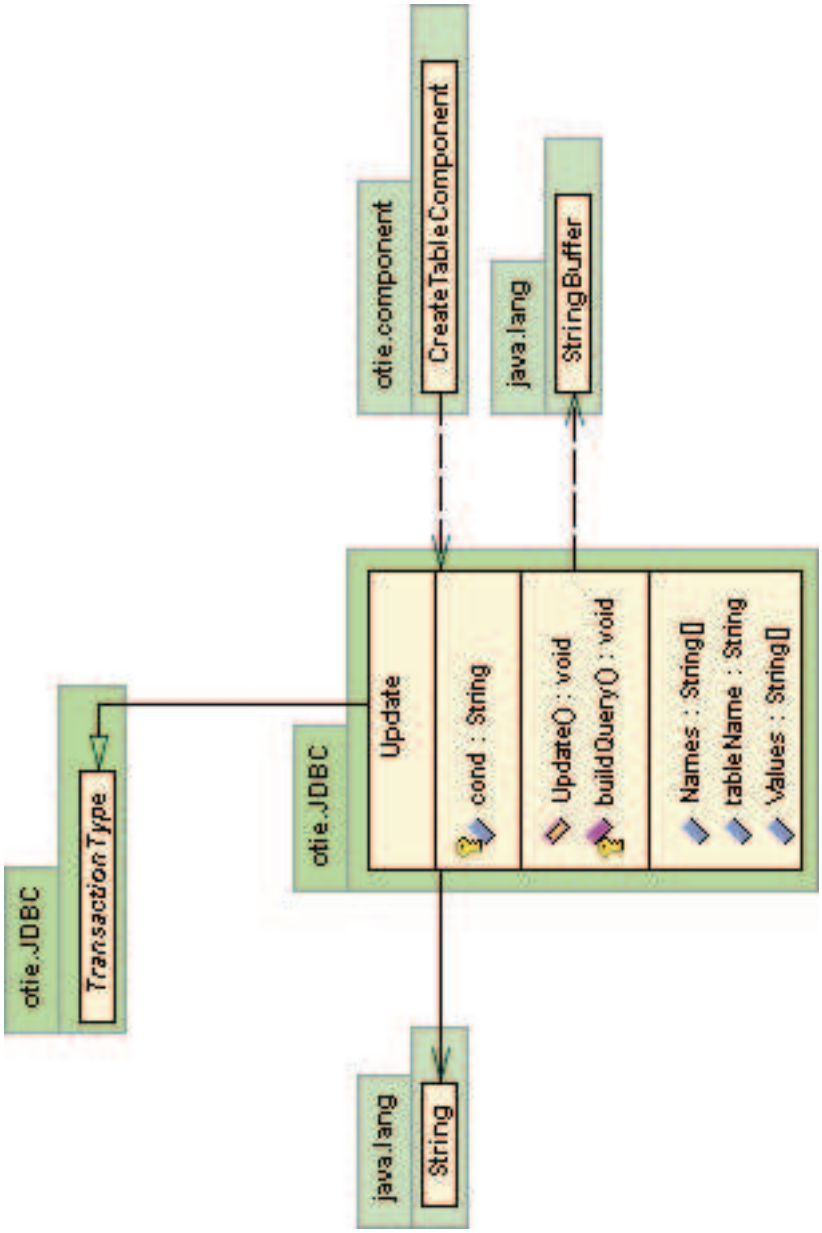
Kuva 10: Transaction rajapinnan UML-kuvaus

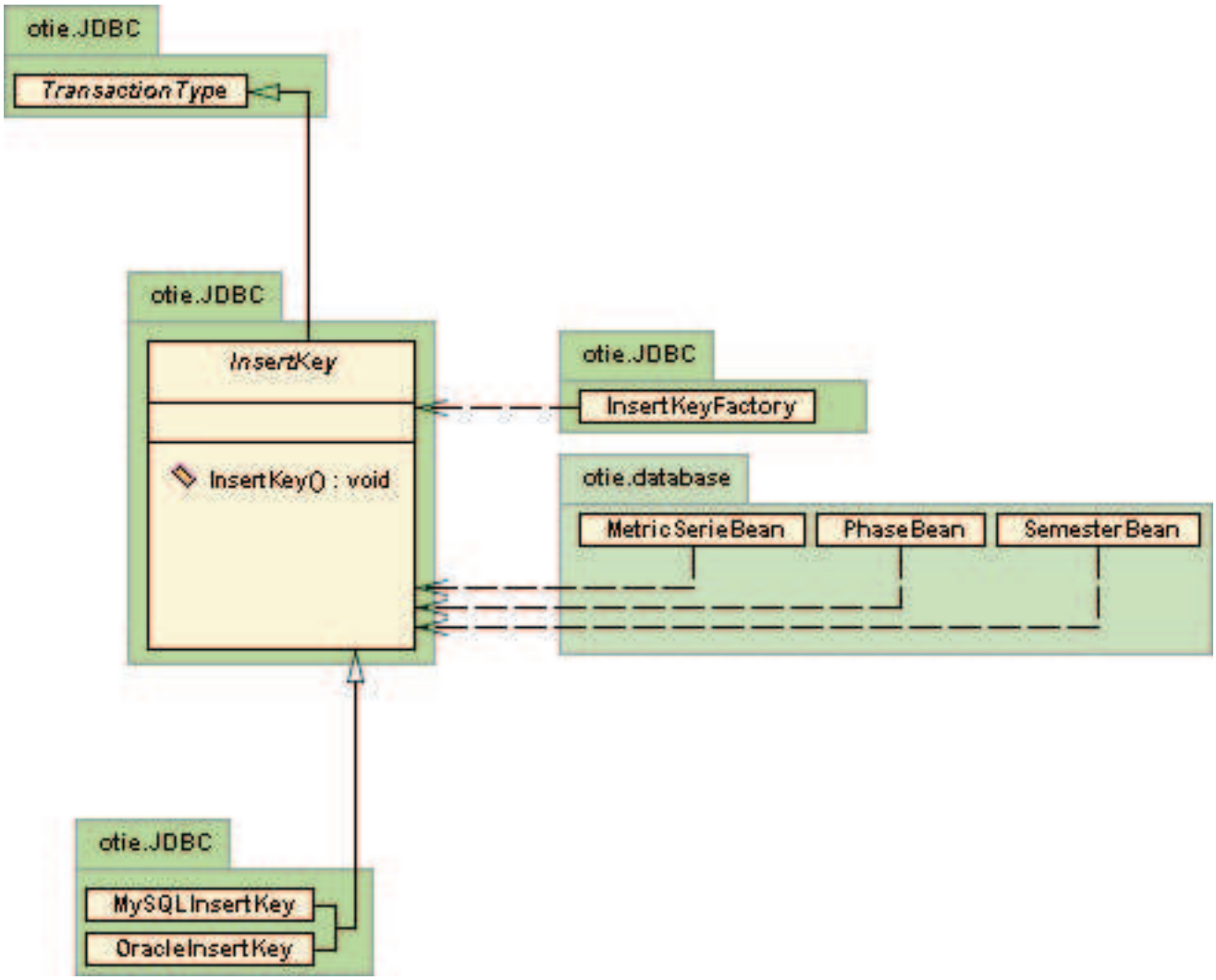

otie.JDBC
TransactionType
qString : String[]
getQuery() : String[]
toString() : String
TransactionType() : void

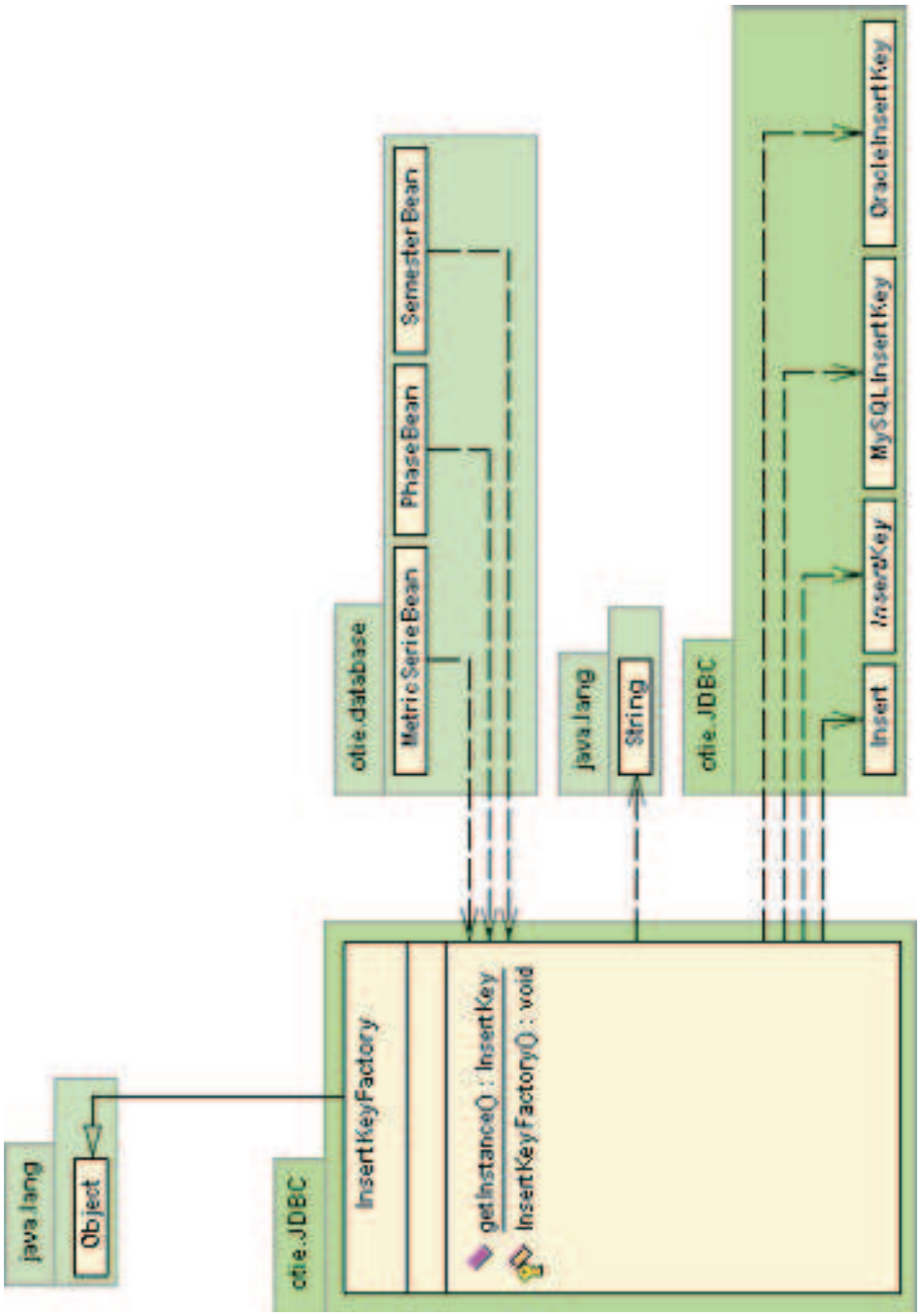

otie.JDBC
InsertKeyFactory
MySQLInsertKey
OracleInsertKey
otie.database
MetricSerieBean
PhaseBean
SemesterBean
java.lang
StringBuffer
otie.JDBC
Insert
insertQ : void
buildQuery() : void
names : String[]
tableName : String
values : String[]
otie.JDBC
TransactionType
java.lang
String

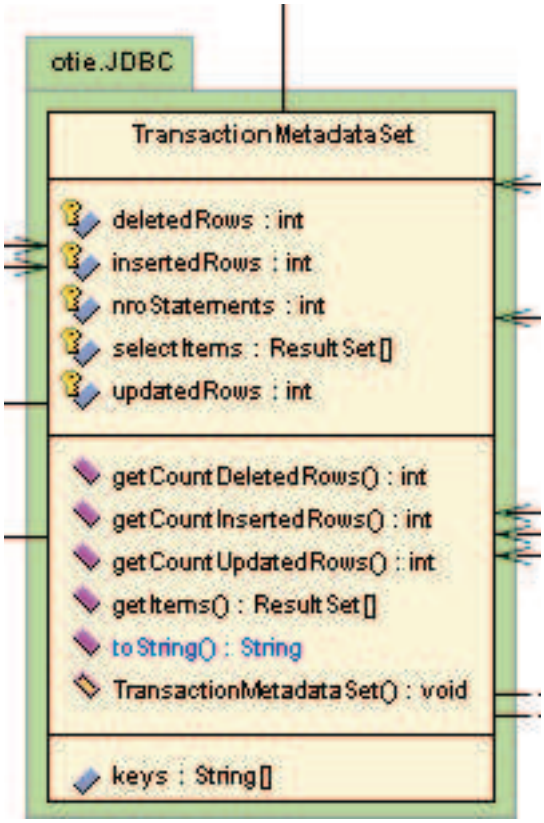

otie.JDBC
TransactionType
otie.JDBC
Delete
java.lang
String
java.lang
StringBuffer
Delete() : void
buildQuery() : void
names : String[]
tableName : String
values : String[]


otie.JDBC
TransactionType
otie.JDBC
SimpleDelete
java.lang
String
otie.component
AdminComponent
otie.database
MetricSerieBean
java.lang
StringBuffer
cond : String
getCondition() : String
SimpleDelete() : void
buildQuery() : void
tableName : String


otie.component
CreateTableComponent
java.lang
StringBuffer
otie.JDBC
Update
cond : String
UpdateQ : void
buildQueryQ : void
Names : String[]
tableName : String
Values : String[]
otie.JDBC
TransactionType
java.lang
String


otie.JDBC
TransactionType
otie.JDBC
InsertKey
InsertKey() : void
otie.JDBC
InsertKeyFactory
otie.database
MetricSerieBean
PhaseBean
SemesterBean
otie.JDBC
MySQLInsertKey
OracleInsertKey


java.lang
Object
otie.JDBC
InsertKeyFactory
getInstance() : InsertKey
InsertKeyFactory() : void
otie.database
MetricSerieBean
PhaseBean
SemesterBean
java.lang
String
otie.JDBC
Insert
InsertKey
MySQLInsertKey
OracleInsertKey


otie.JDBC
TransactionMetadataSet
deletedRows : int
insertedRows : int
nroStatements : int
selectItems : ResultSet[]
updatedRows : int
getCountDeletedRows() : int
getCountInsertedRows() : int
getCountUpdatedRows() : int
getItems() : ResultSet[]
toString() : String
TransactionMetadataSet() : void
keys : String[]

8 XML-apukirjasto

XML-apukirjasto laaditaan XML-tiedostojen käsittelyn helpottamiseksi. Kirjastoluokkien ensisijainen käyttötarkoitus on XML-dokumenttien manipulointi muistissa eli että XML-puun rakenteen muokkaaminen "lennossa"on mahdollista. Jotta sovelluksesta saataisiin mahdollisimman muokattava, on jotakin metakuvauskieltä käytettävä runsaasti. OhtuTie-projektin yhteydessä metakuvauskieleksi valitaan XML-kieli, koska se on saavuttanut standardiaseman modernissa Web-suunnittelussa ja koska XML mahdollistaa myös hajautetut Web-palvelut esim. SOAP-teknologian välityksellä.

XML-kirjasto koostuu neljästä luokasta: XMLFile, XMLObject, XMLAttribute ja XMLTransformer. XMLFile edustaa yksittäistä XML-dokumenttia ja luokan avulla voidaan luoda virtuaalisia XML-dokumentteja, jotka ovat valideja (ja tarvittaessa hyvinmuodostettuja, eli niille on määritetty DTD) XML-dokumentteja ja joita manipuloidaan muistiavaruudessa kirjoittamatta tiedostoa koskaan levylle. XMLObject-luokka edustaa yhtä solmua XML-puussa. Solmulla tarkoitetaan tässä yhteydessä, hieman standardista XML-kielestä suppeammin, yksittäisestä tagista alkavaa XML-puun osaa. Solmujen attribuuttien kuvaamiseksi laaditaan XMLAttribute-luokka, joka siis edustaa yhtä attribuuttia, jolla on nimi ja arvo. Yksinkertaisuuden vuoksi kaikki attribuutit määritetään merkkijonomuotoiseksi. XMLTransformer-luokan tehtävänä on mahdollistaa ajonaikainen XML- ja XSL-dokumenttien avulla tehtävä käännös XHTML-muotoon. Luokka toisin sanoen piilottaa sisälleen erilliset jäsenninkutsut ja täten yksinkertaistaa muunnosten käyttöä.

XMLFile-luokan kuvaus esitetään kuvassa 11, XMLObject-luokan kuvaus esitetään kuvassa 12 ja XMLAttribute-luokan kuvaus esitetään kuvassa 13.

8.1 XMLFile

XMLFile

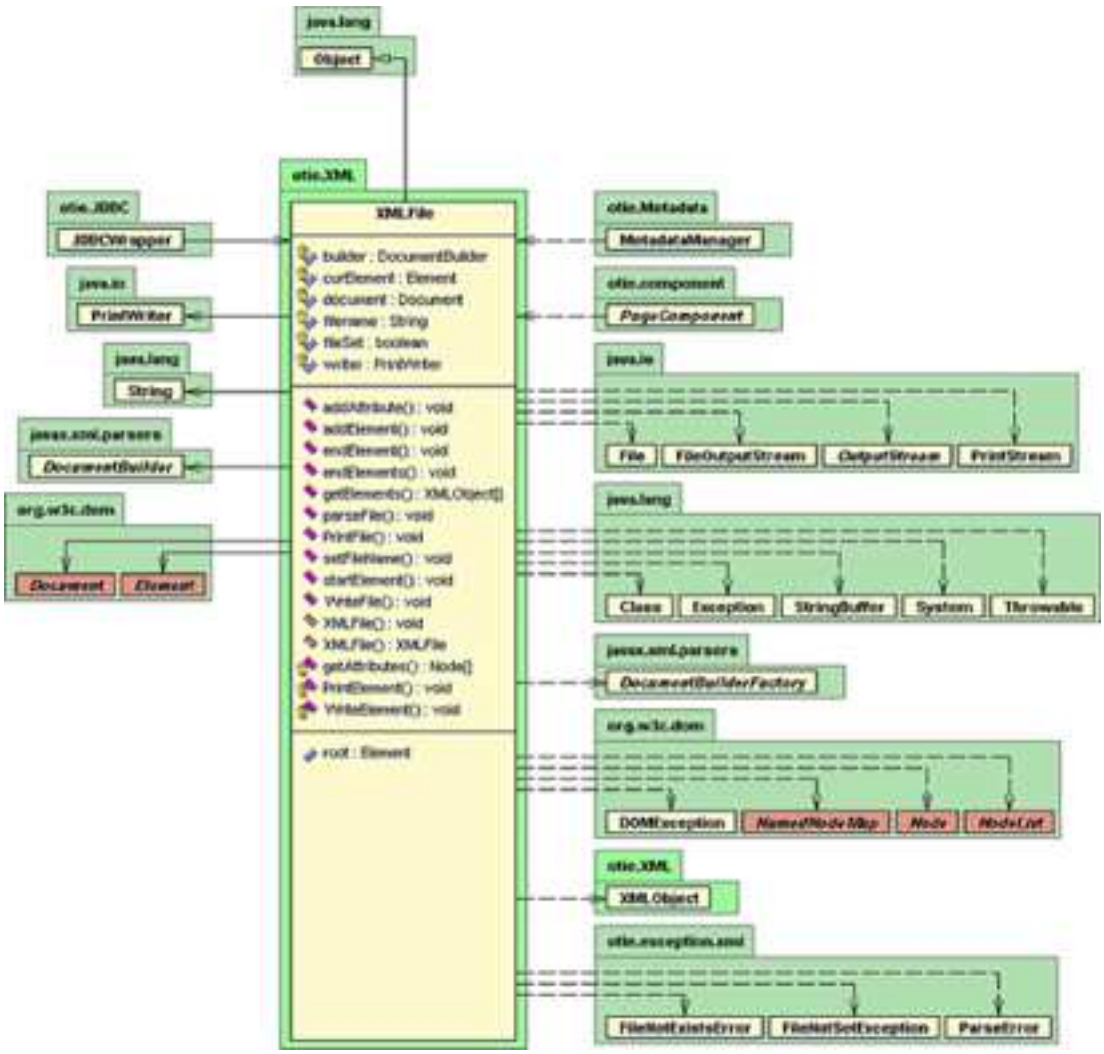
XMLFile-luokan oletuskonstruktori luo uuden XMLFile-olion muistiin. Tämä olio edustaa tyhjää XML-tiedostoa, jolle ei ole määritetty tiedostonimeä ja jossa ei ole yhtään XML-elementtiä.

XMLFile(String fileName)

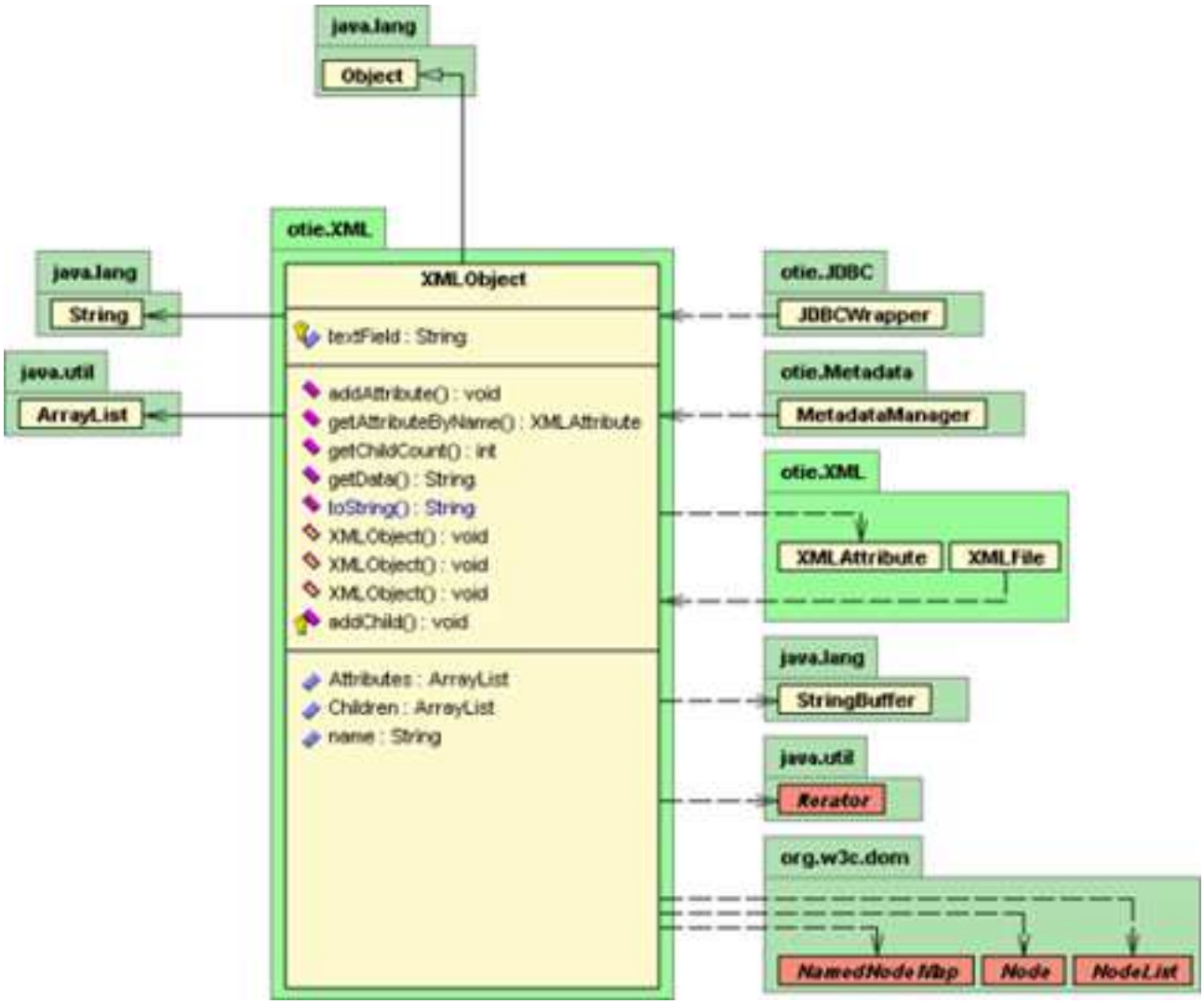
Tämä konstruktori luo uuden XMLFile-olion ja liittää sen *fileName*-parametrin määrittämään tiedostoon. Samalla tarkastetaan, onko parametrina määritetty tiedosto olemassa. Jos tiedosto on olemassa, asetetaan erillinen *fileSet* attribuutti arvoon *true* ja muuten arvoon *false*. Tämän tarkoituksena on helpottaa virheentarkistusta jäsennysvaiheessa.

setFileName(String fileName)

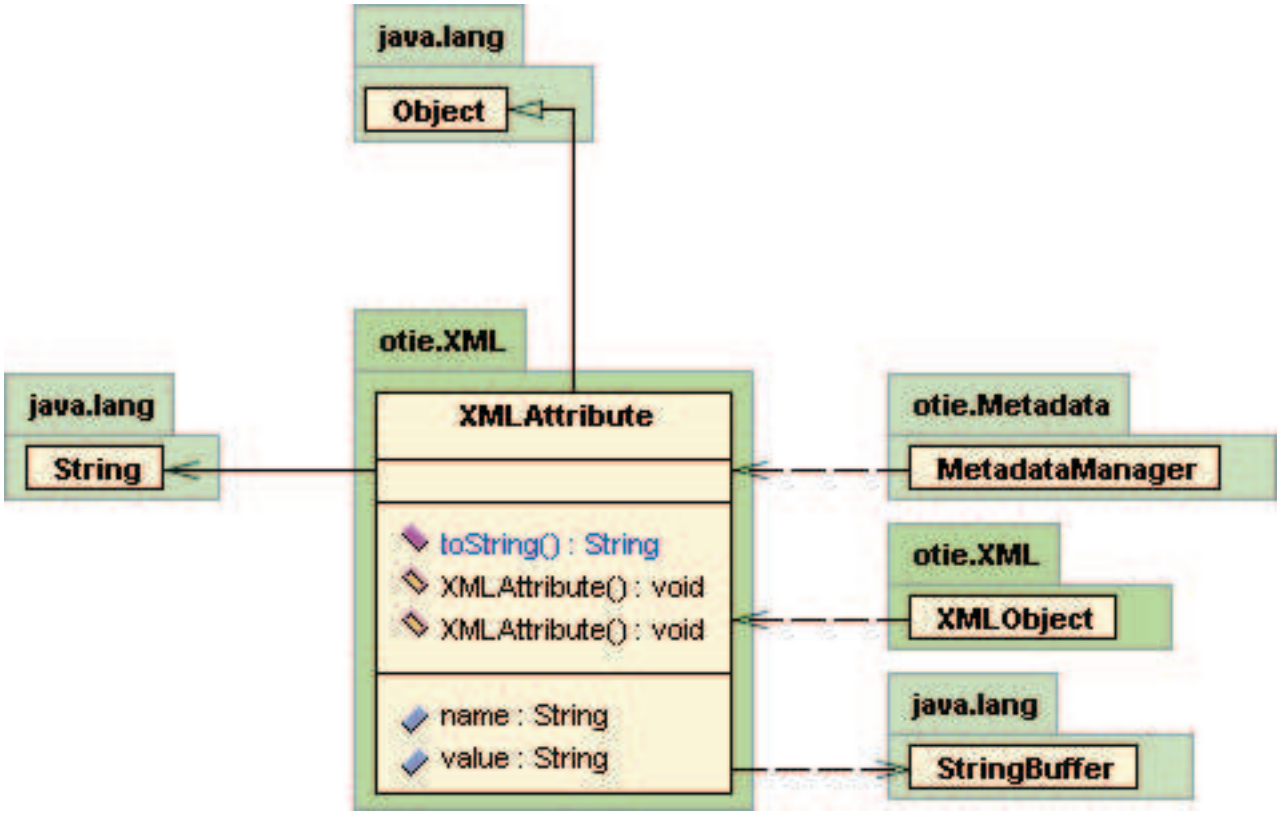
Metodi liittää XMLFile-olion *fileName*-parametrin määräämään tiedostoon. Jos tiedosto on olemassa, asetetaan erillinen *fileSet*-attribuutti arvoon *true* ja muuten arvoon *false*.



Kuva 11: XMLFile-luokan kuvaus.



Kuva 12: XMLObject-luokan UML-kuvaus.



Kuva 13: XMLAttribute-luokan UML-kuvaus.

Tämän tarkoituksena on helpottaa virheentarkistusta jäsennysvaiheessa.

static parseFile(XMLFile xmlFile)

Metodi jäsentää XML-tiedoston eli muodostaa muistiin puumaisen esitysmuodon levyllä olevasta tiedostosta. Jos *fileSet*-attribuutin arvo on *false*, jäsennystä ei yritetä vaan metodi aiheuttaa keskeytyksen. Muussa tapauksessa tiedosto jäsennetään. Jos tiedosto ei ole validi ja hyvinmuodostettu XML-tiedosto, metodi aiheuttaa keskeytyksen.

addElement(String elementName)

Metodi lisää muistissa olevaan XML-puuhun uuden elementin, joka on muotoa <*elementName/*>.

endElements()

Metodi sulkee kaikki avoimet XML-tagit ja siirtää XML-dokumentin sisäisen solmuosoittimen osoittamaan juurisolmuun.

startElement(String elementName)

Metodi aloittaa uuden elementin, joka on muotoa <*elementName*. Toisin kuin metodin *addElement* tapauksessa, *startElement*-kutsu mahdollistaa attribuuttien ja lapsisolmujen lisäämisen XML-puuhun.

endElement()

Metodi sulkee viimeisimmän *startElement*-kutsulla avatun tagin.

WriteFile()

Metodi kirjoittaa XML-dokumentin kirjastoon. Tiedostonimi täytyy olla määritetty eli *fileSet*-parametrin tulee olla *true*-arvoinen.

PrintFile()

Metodi tulostaa XML-dokumentin standarditulostevirtaan (*System.out*).

addAttribute(String attributeName, String value)

Metodi lisää viimeisimmän *startElement*-kutsun avaamaan XML-solmuun attribuutin, jonka nimi on annettu parametrina *attributeName* ja jonka arvo on annettu parametrina *value*.

getRoot():XMLObject

Metodi palauttaa XML-dokumentin juurisolmun, eli dokumentin jäsennyspuun ylimmän solmun (solmu, jolla ei ole isäsolmua).

getElements(String tagName):XMLObject[]

Metodi palauttaa taulukon XML-solmuja, joka pitää sisällään kaikki XML-dokumentissa esiintyvät *tagName*-nimiset solmut.

8.2 XMLObject

XMLObject-luokka edustaa yksittäistä solmua XML-dokumentin jäsennyspuussa. Solmu voi olla tyhjä solmu, jolloin sillä ei ole attribuutteja eikä lapsisolmuja. Tyhjä solmu on XML-muodossa seuraavanlainen: <*solmu* />. Yleensä kuitenkin solmulla voi olla useita attribuutteja ja lapsisolmuja, jolloin solmu on XML-muodossa seuraavanlainen: <*solmu*>...</*solmu*>. Solmulla on nimi, joka toimii solmun tunnisteena. Nimi on sama kuin lopullisessa XML-dokumentissa esiintyvän tagin nimi. Eli esim. <*test*/>-tagi vastaa *test*-nimistä XML-solmua.

XMLObject(String oName)

Konstruktori joka luo uuden *XMLObject*-solmun ja nimeää sen parametrin *oName* mukaan.

getData():String

Metodi, joka palauttaa sen solmun alku- ja lopputagin välissä olevan tekstiaineiston, joka ei kuulu minkään solmun alisolmuun. Esimerkiksi jos solmu on seuraavanlainen:

```
<solmu>
    tekstiä
    <toinen_solmu/>
    <kolmas_solmu/>
</solmu>
```

niin metodi *getData()* palauttaa merkkijonon *tekstiä*.

addAttribute(String name, String value)

Metodi lisää XML-solmulle uuden attribuutin, jonka nimi on määritetty parametrissä *name* ja jonka arvo on annettu parametrissä *value*.

getChildCount():int

Metodi palautaa XML-solmun lapsisolmujen määrän. Esimerkiksi jos solmu on seuraavanlainen:

```
<solmu>
    tekstiä
    <toinen_solmu/>
    <kolmas_solmu/>
</solmu>
```

niin metodi *getChildCount()* palauttaa arvon kaksi.

getChildren():Iterator

Metodi palauttaa *java.util.Iterator*-olion, jonka avulla lapsisolmujen läpikäynti on vaivatonta.

getAttributes():Iterator

Metodi palauttaa *java.util.Iterator*-olion, jonka avulla attribuuttien läpikäynti on vaivatonta.

getAttributeByName(String attrName):XMLAttribute

Metodi palauttaa kaikki XMLAttribuutit, joiden nimi on määritetty parametrissä *attrName*.

getName():String

Metodi palautaa XML-solmun nimen eli solmun taginimen valmiissa XML-dokumentissa.

8.3 XMLAttribute

Jokaisella XML-jäsennyspuun solmun attribuutilla on kaksi parametria: nimi ja arvo. *XMLAttribute*-luokka on yksinkertainen luokka, jolla on vain nämä kaksi parametria ja joiden käsittelyyn luokka tarjoaa *set*- ja *get*-metodit.

XMLAttribute(String name, String value)

Konstruktori, joka luo XMLAttribute-olion, jonka nimi on *name*-parametrin arvo ja jonka edustaman attribuutin arvo on *value*-parametrin määrittämä.

getName():String

Metodi joka palautaa attribuutin nimen.

getValue():String

Metodi joka palauttaa attribuutin arvon.

setValue(String value)

Metodi asettaa attribuutin arvoksi *value*-parametrin määrittämän arvon.

setName(String name)

Metodi asettaa attribuutin nimeksi *name* parametrin määrittämän merkkijonon.

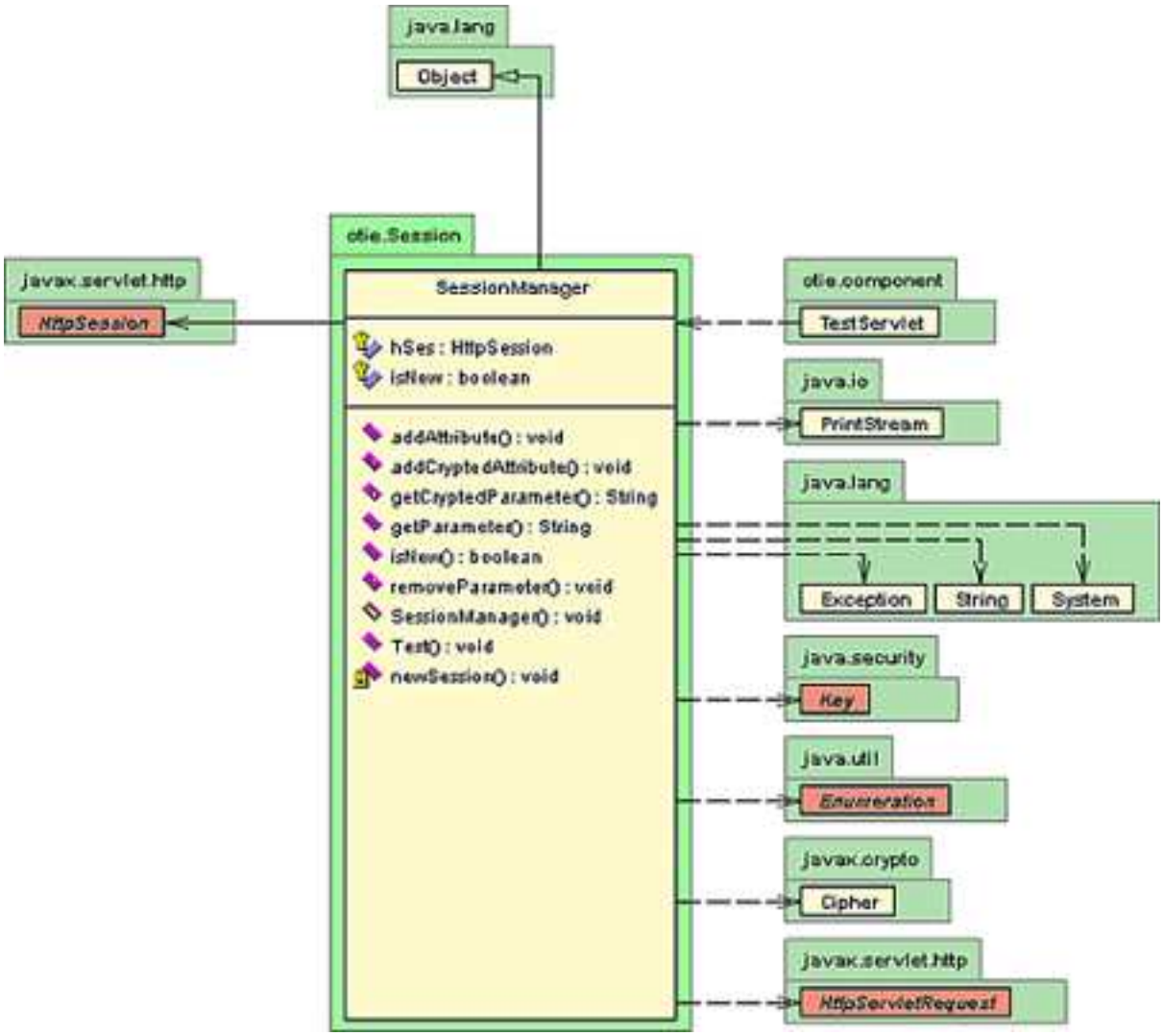
9 Istuntokirjasto

SessionManager-komponentti on vastuussa istuntojen hallinnasta. Eristämällä istuntotietojen hallinta omaksi komponentikseen, on sovelluksen testaaminen sekä kehitys- että varsinaisessa testausvaiheessa helpompaa. Kontrolleriobjekti luo jokaisen sivupyynnön yhteydessä uuden SessionManager-olion, jos käyttäjä on kirjautunut järjestelmään. Konstruktorin parametrina syötetään *HttpRequest*-muuttuja, joka saadaan suoraan kontrollerin *doPost()*-metodin kutsusta. Jos käyttäjälle ei oltu vielä aloitettu istuntoa, luodaan uusi istunto ja asetetaan *boolean*-muuttujan *isNew*-arvoksi **true**. Jos käyttäjälle oli jo aloitettu istunto, konstruktorissa kutsuttava *HttpServletRequest.getSession()*-metodi palauttaa *HttpSession*-objektin, jota voidaan hyödyntää sellaisenaan. Tietoturvasyistä käyttäjän IP-osoite tarkistetaan jokaisen sivupyynnön yhteydessä kutsumalla *newSession()*-metodia. Tämän tarkoituksena on parantaa sovelluksen tietoturvallisuutta pyrkimällä havaitsemaan istuntojen kaappausyritykset. Jos käyttäjän IP-osoite vaihtuu kesken istunnon, oletetaan, että joku ulkopuolinen on saanut sovelluksen käyttöoikeuden haltuunsa. Käyttäjä suljetaan tällöin järjestelmästä tuhoamalla käyttäjän istuntotiedot. Tällöin käyttäjä joutuu kirjautumaan järjestelmään uudelleen.

SessionManager- komponentin UML-kaavio esitetään kuvassa 14. Seuraavassa kuvataan luokan keskeisimmät metodit:

SessionManager(HttpServletRequest)

Konstruktorin tehtävänä on ensiksi tarkistaa, onko istuntoa aloitettu vai ei. Jos istunto on jo aloitettu, muuttujan *hSes*-arvo saadaan suoraan *HttpServletRequest.getSession(false)*-kutsusta. Muussa tapauksessa luodaan uusi istunto *HttpServletRequest.getSession(true)*-kutsulla. Lisäksi tällöin asetetaan *boolean*-muuttujan *isNew* arvoksi **true**.



Kuva 14: SessionManager-komponentin UML-kuvaus.

addAttribute(String name, String value)

Istuntotiedot koostuvat pääasiallisesti merkkijonopareista, jotka ovat muotoa (*Nimi*, *Arvo*). Metodin *addAttribute* avulla voidaan lisätä uusi tämänkaltainen pari istuntotietoihin. Käytännössä metodin toteutukseksi riittää siis *HttpSession.setAttribute(String,String)*-metodin kutsuminen.

addCryptedAttribute(String name, String value, java.security.Key)

Tämän metodin avulla voidaan istuntotietoihin lisätä uusi salattu attribuutti. Attribuutin lisääminen tapahtuu kuten *addAttribute*-metodissa, mutta ennen attribuutin lisäystä lisättävän attribuutin arvo täytyy salata. Tämä tapahtuu luomalla ensiksi uusi *javax.crypto.Cipher*-olio ja alustamalla haluttu salausalgoritmi muotoa *java.security.Key* olevan avaimen avulla. Tietoturvasyistä käytettävä salausalgoritmi sekä sen parametrit on jätetty pois tästä dokumentista. Lisäksi kyseisten metodien kuvaus jätetään pois toteutusdokumentista eikä julkaistavassa koodissa anneta tätä metodia ollenkaan.

getParameter(String name):String

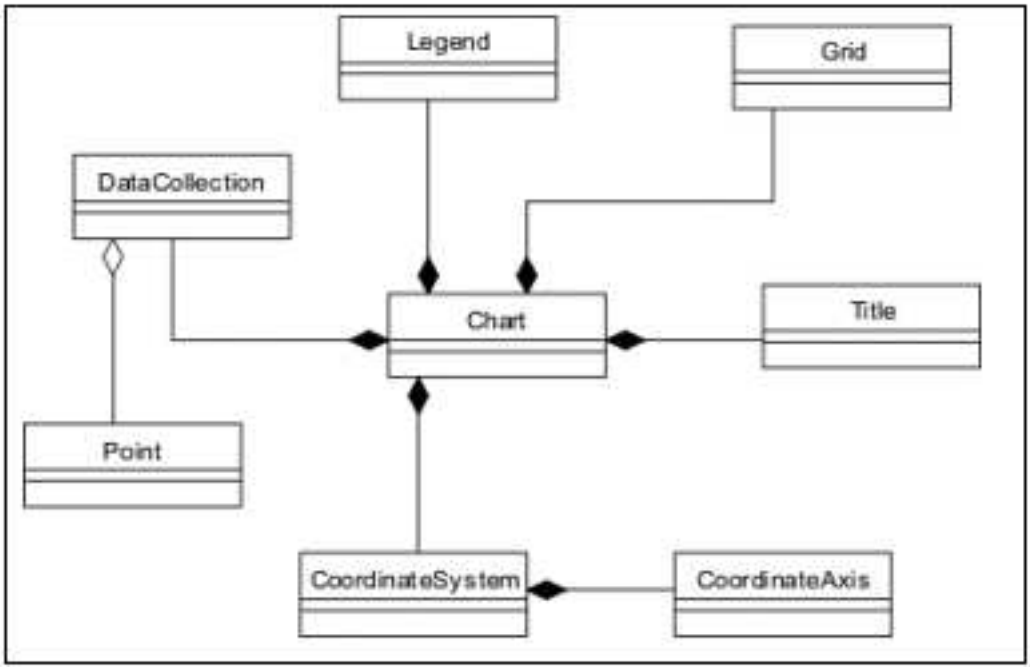
Tämän metodin tarkoituksena on palauttaa sen istuntoattribuutin arvo, jonka nimi annetaan parametrina. Toiminto toteutetaan kutsumalla *HttpSession.getAttribute(String)*-metodia ja muuntamalla ko. metodin palauttama Object-olio String-olioksi. Järjestelmässä ei tallenneta objekteja istuntotietoihin, vaan kaikki tallennettavat oliot muutetaan ensiksi merkkijonomuotoon *Serialize*-kutsujen avulla. Tämän vuoksi sovelluksessa ei tarvita muunlaisia *getParameter*-metodeja.

getCryptedParameter(String name, Key key):String

Metodi *getCryptedParameter* toimii käytännössä samoin kuin *getParameter*-metodi. Ensiksi haetaan se istuntoattribuutin arvo, johon parametri **name** viittaa. Jos ko. metodi palauttaa tyhjästä merkkijonosta poikkeavan arvon, oli kyseiseen istuntomuuttujaan tallennettu arvo. Tämän jälkeen attribuutin arvo puretaan käyttämällä parametrina **key** välitettyä salausavainta.

isNew():boolean

Metodin *isNew()* avulla voidaan tarkistaa, onko istunto juuri aloitettu vai vanhempi. Käytännössä tämä tarkoittaa sitä, että onko käyttäjä juuri kirjautunut järjestelmään, vai onko istunto kestänyt jo kauemmin. Tätä tietoa hyödynnetään tulostettaessa erityistä tervetuloruutua käyttäjälle.



Kuva 15: Graafikirjaston komponenttien rakenteen UML-kuvaus.

removeParameter(String name)

Metodi poistaa istuntoattribuutin, jonka nimi on annettu parametrina **name**.

Test()

Kyseinen metodi on luokassa testitarkoituksia varten. Tällöin luokan metodeja voi testata ilman, että *Tomcat*-palvelinta täytyy välillä sammuttaa ja käynnistää uudelleen. Metodin toiminnallisuus vaihtelee toteutuksen eri vaiheissa ja lopullisesta versiosta tämä metodi jää pois.

newSession()

Metodi on refaktoroinnin tuloksena syntynyt metodi, joka tarkistaa jokaisen sivupyynnön yhteydessä käyttäjän selaimen ja IP-osoitteen. Selaimen tarkistus tarvitaan, jotta käyttäjälle voitaisiin valita oikea näkymä. Tietoa IP-osoitteesta tarvitaan tietoturvasyistä, kuten tämän luvun johdannossa on selostettu.

10 Graafi rajapinnan kuvaus

10.1 Point

Point-luokka on moniulotteisen pisteen abstraktio, joka piilottaa datan ulottuvuuden. Luokkaa käytetään *DataCollection*-luokan tietorakenteena ja tämän luokan käyttö on välttämätöntä, jotta ei tarvitsisi käyttää dynaamisia taulukoita. Monet luokan metodeista ovat OhtuTie-järjestelmän kannalta turhia, mutta koska luokka on alunperin suunniteltu yleisluontoiseen käyttöön, ovat kaikki metodit mukana valmiissa projektissa. Suunnitteludokumentissa kuitenkin kuvataan vain kaksi tärkeintä metodia.

getDimensionality():int

Palauttaa datan ulottuvuuksien määrän.

getCoordinate(int index):double

Palauttaa ulottuvuuden *index* osoittaman koordinaattiarvon.

getCoordinates:double[]

Palauttaa *double* taulukon, jossa taulukon jokainen indeksi vastaa yhtä pisteen yksikkövektorin komponenttia Euklidisessä avaruudessa.

10.2 DataCollection

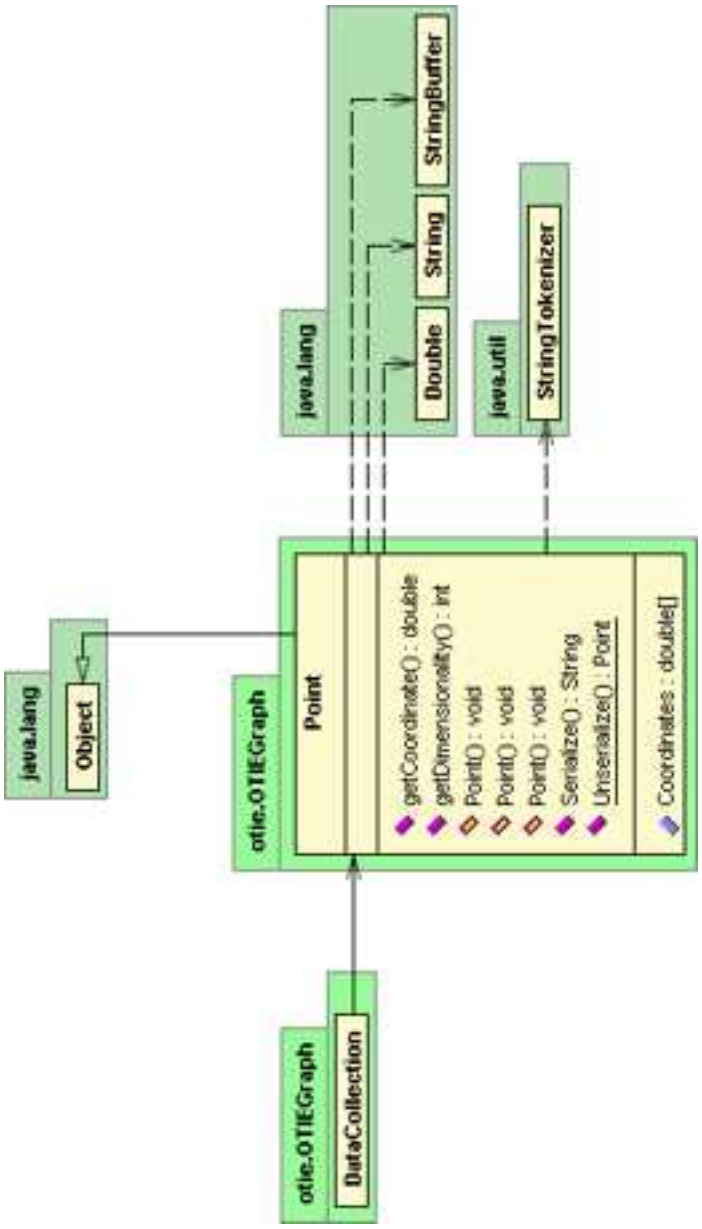
DataColletion-luokka abstrahoi joukon pisteitä. *Point*-luokan avulla abstrahoidaan yksittäisen pisteen ulottuvuuksien määrä. Ennen kaikkea *DataCollection* on abstrakti tietorakenne, joka helpottaa graafien piirtoa huomattavasti. Täydellisyyden (ja uudelleenkäytettävyyden) vuoksi luokkaan on toteutettu mm. ylimääräisiä konstruktoreita, joita ei OhtuTie-järjestelmässä tarvita. Tässä dokumentissa listataan vain ne luokan metodit, joita oikeasti sovelluksessa käytetään.

DataCollection()

Oletuskonstruktori varaa tilaa yksittäisille datapisteille. Oletusarvona on ARRAYSIZE, jonka arvoksi on määritelty 64.

DataCollection(int dataDimensionality)

Konstruktori varaa tilaa *DataCollection.ARRAYSIZE* (=64) -pisteelle ja varaa jokaiselle koordinaatille *dataDimensionality*-parametrin osoittaman määrän komponentteja.



Kuva 16: Point-luokan UML-kuvaus.

DataCollection(double[][] data)

Luo uuden *DataCollection*-olion parametrina *data* annetusta datasta.

AddData(double[] data)

Lisää datakokoelmaan uuden pisteen.

ClearData()

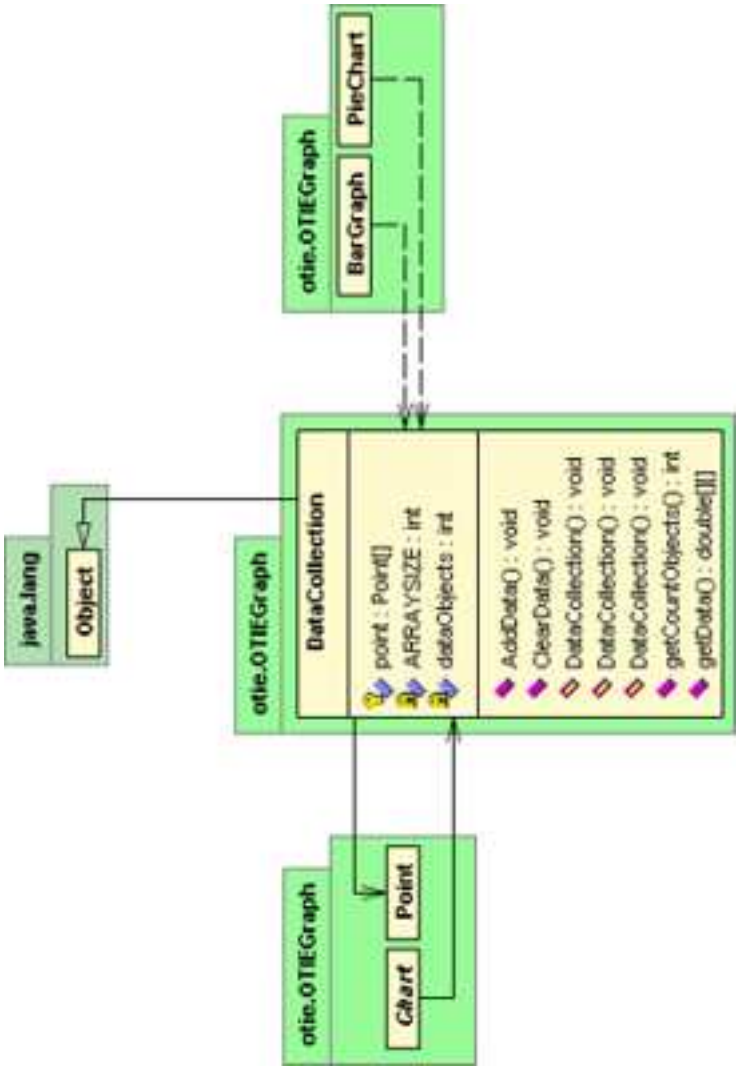
Tyhjentää datakokoelman pisteistä.

getData():double[][]

Palauttaa datakokoelman sisältämät datapisteet *double[]*-taulukossa.

getCountObjects():int

Palauttaa kokoelman pisteobjektien määrän.



Kuva 17: DataCollection-luokan UML-kuvaus.

10.3 Chart

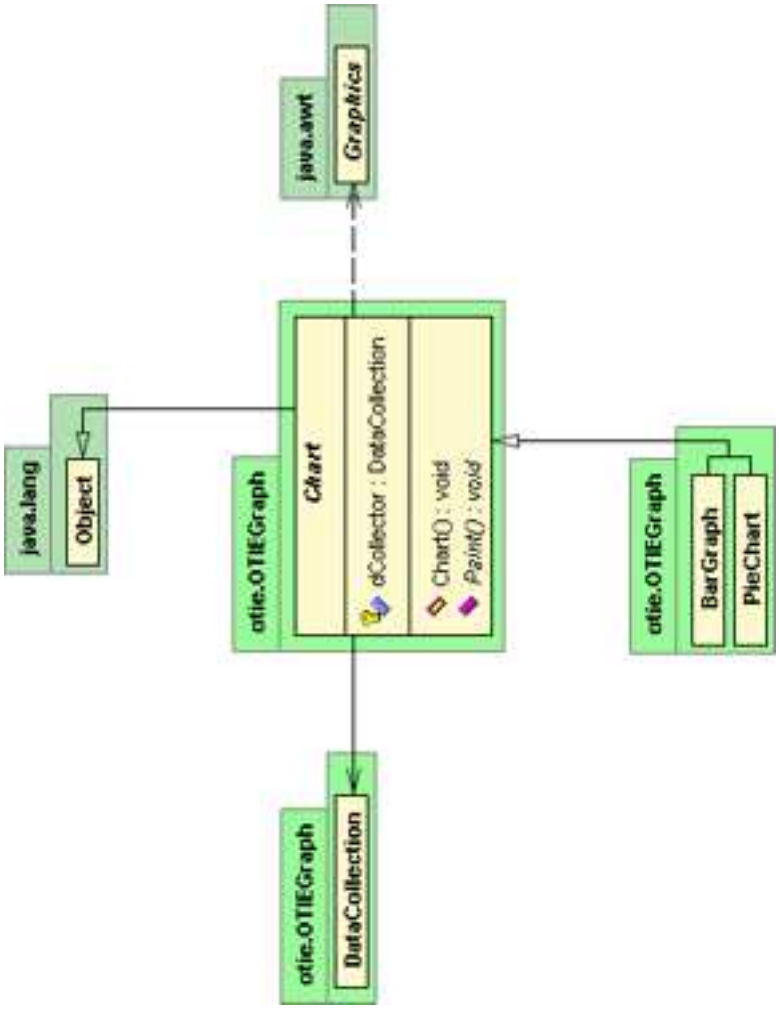
Abstrakti luokka, joka määrittää kaikille kaaviotyypeille yhteisen rajapinnan. Lisäksi *Chart*-luokka sisältää kaikille luokille yhteisen konstruktorin.

Chart(double[][] data)

Konstruktori, joka luo uuden *DataCollection*-olion ja lisää parametrin *data* sisältämät pisteet kokoelmaan.

Paint(Graphics g, int x, int y,int width, int height) {*abstract*}

Abstrakti piirtometodi, joka on toteutettava kaikissa perityissä graafiluokissa.



Kuva 18: Chart-luokan UML-kuvaus.

Kuva 19: Title-luokan UML-kuvaus.

10.4 Legend

10.5 Title

Title-luokka edustaa kaavion otsikkokenttää. Otsikkokentällä on teksti, joka tulostetaan neliönmuotoiseen laatikkoon kaavion yläpuolelle. OhtuTie-projektin puitteissa laadittavan *Title*-luokan ominaisuuksiin kuuluu vain fontin vaihtaminen, tekstin värin vaihtaminen sekä taustavärin vaihtaminen. Kuvassa 19 on esitetty Title-luokan UML-kaavio. Mikäli aikaa riittää, luokkaan toteutetaan lisäominaisuuksia kuten sijainnin valinta, reunaviivan tyylin ja paksuuden valinta jne. Näitä ominaisuuksia ei kuitenkaan huomioida suunnitteluvaiheessa.

Title()

Oletuskonstruktori luo tyhjän otsikkolaatikon. Oletusarvoisesti otsikkolaatikkoa, jolle ei ole asetettu otsikkotekstiä, ei piirretä ruudulle.

Title(String titleText)

Luo uuden *Title*-olion ja asettaa sen otsikkotekstin arvoksi *titleText*-parametrin osoittaman arvon.

setTitle(String titleText): String

Metodi asettaa otsikko-olion otsikkotekstin arvoksi *titleText*-parametrin osoittaman arvon.

getTitleText():String

Metodi palauttaa *Title*-oliolle asetetun otsikkotekstin.

setFont(Font font)

Metodi asettaa otsikko-oliolle *font*-parametrin määrittämän fontin. Fontti annetaan *java.awt.Font*-muodossa.

getFont(): Font

Metodi palauttaa otsikko-oliolle määritellyn fontin *java.awt.Font*-oliona.

setForecolor(Color color)

Metodi asettaa otsikko-oliolle *color*-parametrin osoittaman väriarvon tekstin väriksi. Parametri annetaan *java.awt.Color*-oliona.

setBackcolor(Color color)

Metodi asettaa otsikko-oliolle *color* parametrin osoittaman väriarvon taustaväriksi. Parametri annetaan *java.awt.Color*-oliona.

getForecolor() : Color

Metodi palauttaa tekstille määritellyn väriarvon *java.awt.Color*-oliona.

getBackcolor() : Color

Metodi palauttaa otsikko-oliolle määritellyn taustavärin *java.awt.Color*-oliona.

10.6 CoordinateSystem

10.7 Grid

11 Muut luokat

11.1 SecurityManager

SecurityManager-luokan tehtävänä on tarjota metodeja erilaisia salaus ja tietoturvatoimintoja varten. Luokka tarjoaa metodeja sekä symmetriseen salaukseen että tiivisteiden (hash) laskemiseen. Järjestelmässä oli tarkoitus käyttää symmetristä AES-salausta, mutta koska Javan kryptografinen rajapinta vaatii myös avainten hallinnan toteuttamista, ryhmä päätti siirtyä käyttämään muita menetelmiä. Kaikki järjestelmän SecurityManager kutsut käyttävät staattisia metodeja, joten SecurityManager toimii lopullisessa versiossa täysin kirjastoluokkana. Täten SecurityManagerin suhteesta muuhun järjestelmään ei esitettä erillistä UML-kaaviota.

SecurityManager-luokasta voi myös luoda ilmentymän, jolloin luokka toimii salauskomponenttina, jolle voi syöttää merkkijonoja ja käskeä luokkaa salaamaan tai purkamaan ko. merkkijonoja. Aina luotaessa uusi SecurityManager, olio luo uuden avaimen, jota käytetään salaukseen. Avaimen muodostamiseen käytetään sekä staattista aineistoa että satunnaisia byte arvoja.

SecurityManager() throws NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException

Oletuskonstruktori alustaa symmetrisessä salauksessa tarvittavat parametrit sekä luo salausalgoritmin ilmentymän. Jos salausalgoritmia ei tueta, metodi aiheuttaa poikkeuksen.

static SecretKey getPBESecretKey(String material) throws NoSuchAlgorithmException, InvalidKeySpecException

Metodi joka muodostaa salasanapohjaiseen salaukseen soveltuvan salaisen avaimen annetun materiaalin perusteella. Eri toteutukset (sovellusympäristöt) toimivat hieman eri tavalla, joten on täysin mahdotonta sanoa, miten hyvin sama avain voidaan generoida samasta aineistosta.

static String PBEEncode(String text, SecretKey sKey) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException

Metodi suorittaa salasanapohjaisen salauksen merkkijonolle *text* käyttäen salaista avainta *sKey*. Metodin käyttöä ei suositella, sillä tavallisen salasanapohjaisen salauksen purkami-

nen on usein helpohokoa. Jos metodi aiheuttaa poikkeuksen, johtuu se siitä että järjestelmä ei tue salasanapohjaista salausta. Tällöin järjestelmän java kirjastoja täytyy päivittää siten, että ne tukevat vaadittuja salausoperaatioita.

static String PBEDecode(String cipherText, SecretKey sKey) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException

Metodi suorittaa salasanapohjaisen purkuoperaation merkkijonolle *cipherText* käyttäen salaista avainta *sKey*. Metodin käyttöä ei suositella, sillä tavallisen salasanapohjaisen salauksen murtaminen on usein helpohkoa.

static String SHADigest(String text) throws NoSuchAlgorithmException

Metodi laskee SHA-1 hajautteen annetulle merkkijonolle. Hajautteet sopivat esim. salasanojen käyttöön, mutta hajautettuja merkkijonoja ei voi "purkaa"takaisin alkuperäiseen muotoon, minkä takia tiivistefunktiot soveltuvat ainoastaan tilanteisiin, joissa yksisuuntainen salaus on riittävää. Jos järjestelmä ei tue SHA1 algoritmia, aiheutuu NoSuchAlgorithmException.

static String MD5Digest(String text) throws NoSuchAlgorithmException

Metodi laskee MD5 hajautteen arvon annetulle merkkijonolle. Metodi on muuten sama kuin SHADigest paitsi että nyt käytetään MD5 hajautusfunktiota SHA-1 funktion sijasta.

String Encrypt(String text) throws InvalidKeyException, BadPaddingException, IllegalBlockSizeException, InvalidAlgorithmParameterException, UnsupportedEncodingException

Metodi salaa parametrina annettavan merkkijonon *text* käyttäen nykyistä avainta sekä luontivaiheessa luotua salausalgoritmin ilmentymää. Metodin virhearvot kuvataan purkumetodin yhteydessä.

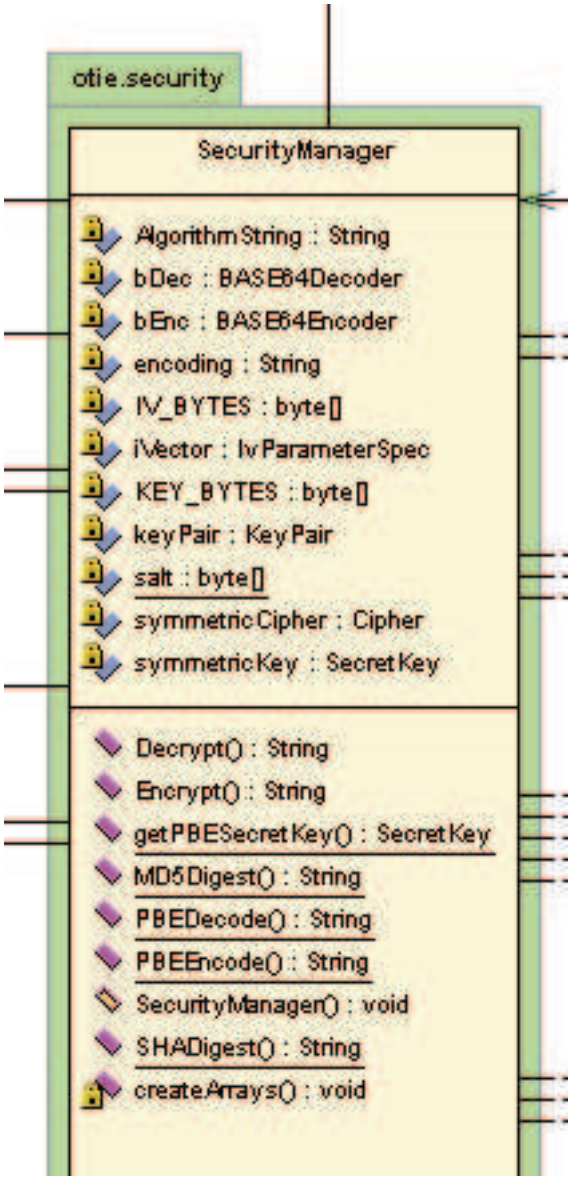
String Decrypt(String text) throws InvalidKeyException, BadPaddingException, IllegalBlockSizeException, InvalidAlgorithmParameterException, UnsupportedEncodingException, IOException

Metodi purkaa parametrina annetun *text* merkkijonon käyttäen sen hetkistä avainta. Jos ko. olion avain on virhellinen, aiheutuu *InvalidKeyException* keskeytys. Jos algoritmille määritetty täyttömoodia ei tueta järjestelmässä, aiheutuu *BadPaddingException*. BadPaddingException tarkoittaa sitä, että järjestelmä, jossa SecurityManageria käytetään, ei tue vaadittua algoritmia. Tällöin koodia täytyy muuttaa manuaalisesti siten, että käytettävää salausalgoritmia vaihdetaan / täyttömoodia vaihdetaan. Vastaavasti muut virheet liittyvät

järjestelmän käyttöympäristön ominaisuuksiin, eivätkä sinällään kerro virheestä koodissa/käyttäjän parametreissa.

private createArrays()

Metodi alustaa symmetrisen salauksen tarvitsemat suola ja avain taulukot satunnaisilla byte arvoilla.



12 Tietokannan kuvaus

OhtuTie-järjestelmän tietokantaan tallennetaan tietoa ohjelmistotuotantoprojekteista ja niiden osallistujista. Tietokanta on toteutettu relaatiotietokantana ja sen suunnittelussa huo-

mioidaan mahdolliset virhetilanteet ja viite-eheyden säilyminen. Tietokantakaavio on esitetty kuvassa 20.

Tietokantajärjestelmänä toimii Oraclen tietokantaohjelmiston versio 9.0 Helsingin yliopiston tietojenkäsittelytieteen laitoksen koneessa Bodbacka. Tietokantaa käytetään JDBC-rajapinnan avulla ja rajapinnan käyttöä helpottamaan tehdään erillinen *JDBCWrapper*-kirjasto.

Tietokannan erikoispiirteenä on, että aina kun vastuuhenkilö luo uusia metriikkasarjoja, järjestelmä luo tietokantaan uutta metriikkasarjaa vastaavan taulun. Järjestelmä nimeää uuden taulun metriikkasarjan nimellä. Tietokantaan luotavan uuden taulun pohjana käytetään MetricSerieInformation-taulua. Metriikkasarjaan liittyvät metriikat lisätään tauluun attribuutteina.

12.1 Project

Project-taulussa on projekteihin liittyviä hallintatietoja, kuten projektin nimi, projektimalli ja projektin ajankohta. Project-taulun rakenne on kuvattu taulukossa 3.

12.2 ProjectType

ProjectType-taulu sisältää projekteissa tuotettavien ohjelmistojen tyyppeihin liittyviä tietoja. OhtuTie-ryhmän tuottaman esimerkkitoteutuksen projektityyppejä ovat esimerkiksi informaatiojärjestelmät tai väliohjelmistot. ProjectType-taulun rakenne on kuvattu taulukossa 4.

12.3 ProjectModel

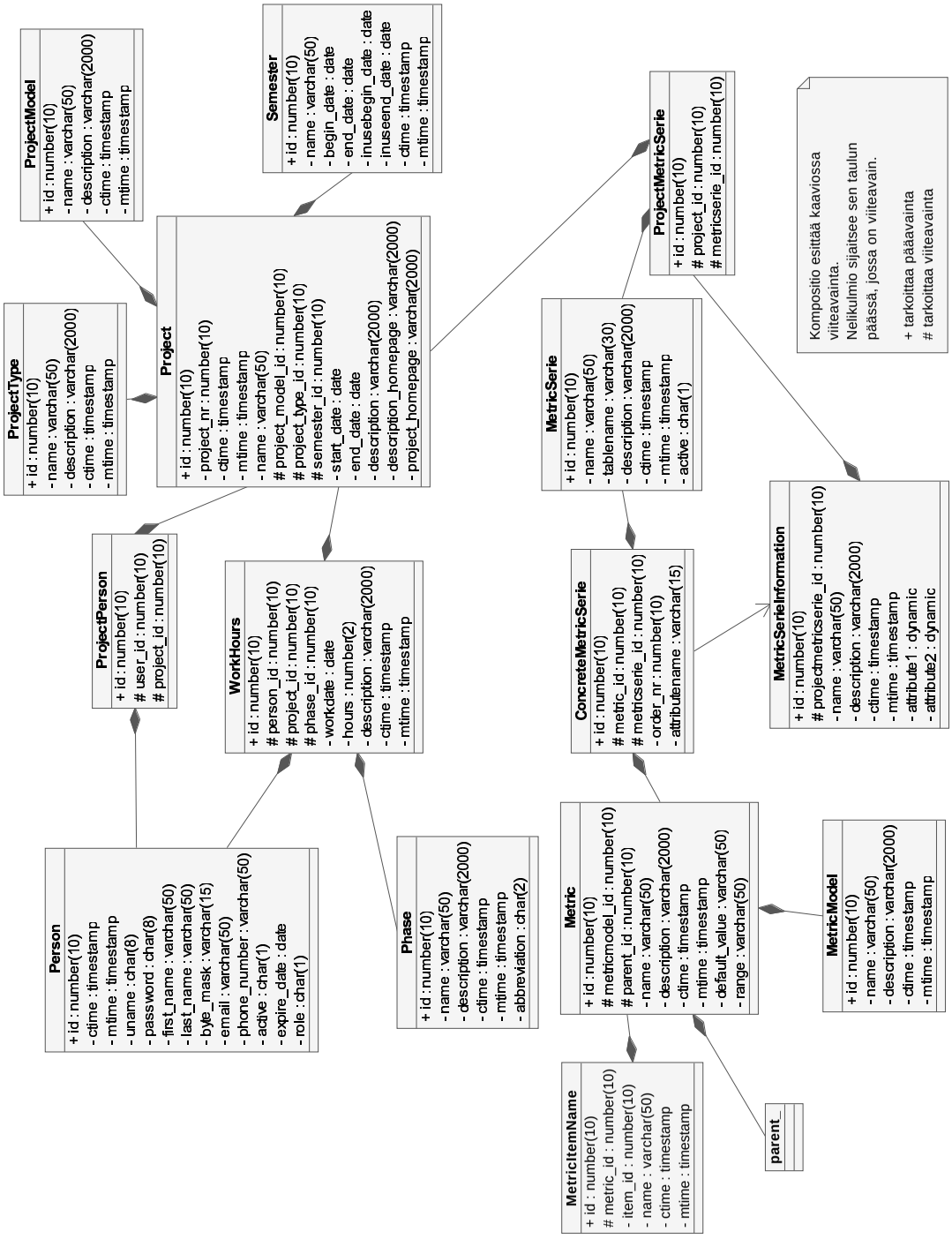
Taulussa ProjectModel on valittavana olevien projektimallien tietoja. ProjectModel-taulun rakenne on kuvattu taulukossa 5.

12.4 Person

Person-tauluun on talletettu tietoja järjestelmän käyttäjistä. Siitä löytyvät muun muassa kunkin käyttäjän käyttäjätunnus, salasana ja käyttöoikeudet. Person-taulun rakenne on kuvattu taulukossa 6.

12.5 ProjectPerson

ProjectPerson-taulua käytetään käyttäjätietojen yhdistämiseen projekteihin. Taulu sisältää käyttäjien tunnistenumeroiden ja projektien tunnistenumeroiden muodostamia pareja. ProjectPerson-taulun rakenne on kuvattu taulukossa 7.



Kuva 20: Tietokantakaavio

12.6 Semester

Semester-taulu pitää sisällään lukukausien tiedot. Semester-taulun rakenne on kuvattu taulukossa 8.

Taulukko 3: Project-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Projektin pääavaimena toimiva tunnistenumero*	*NOT NULL*
project_model_id	number(10)	*Projektissa käytettävään projektimalliin (taulu ProjectModel) viittaava tunnistenumero*	
project_type_id	number(10)	*Projektissa käytettävään projektityyppiin (taulu ProjectType) viittaava tunnistenumero*	
project_nr	number(10)	*Projektin ulkoinen tunnistenumero*	*NOT NULL*
ctime	timestamp	*Projektin luontiajankohta*	*NOT NULL*
mtime	timestamp	*Projektin viimeisin muokkausajankohta*	*NOT NULL*
name	varchar(50)	*Projektin nimi*	
semester_id	number(10)	*Projektin lukukauteen (taulu Semester) viittaava tunnistenumero*	
start_date	date	*Projektin alkupäivämäärä*	
end_date	date	*Projektin loppupäivämäärä*	
description	varchar(2000)	*Projektin kuvaus*	
description_homepage	varchar(2000)	*Projektin kuvauksen sisältävän www-sivun osoite*	
project_home_page	varchar(2000)	*Projektin kotisivun osoite*	

12.7 Metric

Metric-taulussa on yksittäisten metriikoiden tiedot. Metric-taulun rakenne on kuvattu taulukossa 9.

Taulukko 4: ProjectType-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Projektityypin pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Projektityypin nimi*	
description	varchar(2000)	*Projektin kuvaus*	
ctime	timestamp	*Projektityypin luontiaika*	*NOT NULL*
mtime	timestamp	*Projektityypin viimeisen muokkauksen aika*	*NOT NULL*

Taulukko 5: ProjectModel-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Projektimallin pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Projektimallin nimi*	
description	varchar(2000)	*Projektimallin kuvaus*	
ctime	timestamp	*Projektimallin luontiaika*	*NOT NULL*
mtime	timestamp	*Projektimallin viimeisen muokkauksen aika*	*NOT NULL*

12.8 MetricItemName

MetricItemName-taulu liittää listamuotoisten metriikoiden arvoihin niitd vastaavan nimen. MetricItem-taulun rakenne on kuvattu taulukossa 10.

12.9 MetricModel

MetricModel-taulussa on metriikkamallien tiedot. MetricModel-taulun rakenne on kuvattu taulukossa 11.

12.10 Phase

Projektien vaiheet kuvataan Phase-taulussa. Esimerkkitoteutuksessa vaiheita ovat muiden muassa suunnittelu, toteutus ja testaus. Phase-taulun rakenne on kuvattu taulukossa 12.

Taulukko 6: Person-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
ctime	timestamp	*Käyttäjän luontiajankohta*	*NOT NULL*
mtime	timestamp	*Viimeisimmän käyttäjän tietojen muokkauksen ajankohta*	*NOT NULL*
uname	char(8)	*Käyttäjän käyttäjätunnus*	
passoword	char(8)	*Käyttäjän salasana*	
first_name	varchar(50)	*Käyttäjän etunimi*	
last_name	varchar(50)	*Käyttäjän sukunimi*	
byte_mask	number(10)	*Käyttäjän oikeudet määrittelevä bittimaski*	
email	varchar(50)	*Käyttäjän sähköpostiosoite*	
phone_number	varchar(50)	*Käyttäjän puhelinnumero*	
active	number(1)	*Tieto siitä, onko käyttäjä aktiivinen*	
expire_date	date	*Käyttäjän tunnuksen vanhenemispäivämäärä*	
active	number(1)	*Tieto siitä, onko käyttäjä aktiivinen*	

12.11 ConcreteMetricSerie

ConcreteMetricSerie-taulu sisältää tiedon siitä, mitkä metriikat kuuluvat mihinkin metriikkasarjaan. Yhdistys tehdään metriikan ja sarjan tunnistenumeroiden muodostamien parien avulla. ConcreteMetricSerie-taulun rakenne on kuvattu taulukossa 13.

12.12 MetricSerieInformation

MetricSerieInformation-taulu tallentaa metriikkasarjan tiedot. Kun uusi metriikkasarja luodaan, generoi järjestelmä sarjasta tätä mallia noudattavan taulun. Taulun attribuuteiksi tulevat lisäksi myös sarjaan liittyvät metriikat. MetricSerieInformation-taulun rakenne on

Taulukko 7: ProjectPerson-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pädavaimena toimiva tunnistenumero*	*NOT NULL*
user_id	number(10)	*Person-tauluun viittaava tunnistenumero*	
project_id	number(10)	*Project-tauluun viittaava tunnistenumero*	
role	number(1)	*Käyttäjän rooli projektissa.*	

Taulukko 8: Semester-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Lukukauden nimi*	
begin_date	date	*Alkamispäivämäärä*	
end_date	date	*Päättymispäivämäärä*	
in_use_begin_date	date	*Lukukauden käyttöönottoaika*	
in_use_end_date	date	*Lukukauden käytöstäpoistoaika*	
ctime	timestamp	*Luontiaika*	*NOT NULL*
mtime	timestamp	*Viimeisimmdn muokkauksen ajankohta*	*NOT NULL*

kuvattu taulukossa 14.

12.13 MetricSerie

MetricSerie-taulussa kuvataan metriikkasarjat. MetricSerie-taulun rakenne on kuvattu taulukossa 15.

Taulukko 9: Metric-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
metric_model_id	number(10)	*MetricModel-taluun viittaava tunnistenumero*	
phase_id	number(10)	*Phase-tauluun viittaava tunnistenumero*	
parent_id	number(10)	*????????*	
name	varchar(50)	*Metriikan nimi*	
attribute_name	varchar(15)	*???????*	
description	varchar(2000)	*Metriikan kuvaus*	
ctime	timestamp	*Metriikan luontiaika*	*NOT NULL*
mtime	timestamp	*Metriikan edellinen viimeisin muuttamisajankohta*	*NOT NULL*
default_value	varchar(2000)	*Metriikan oletusarvo*	
range	varchar(50)	*Metriikan arvoväli*	

Taulukko 10: MetricItemName-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
ctime	timestamp	*Metriikan luontiaika*	*NOT NULL*
mtime	timestamp	*Metriikan edellinen viimeisin muuttamisajankohta*	*NOT NULL*
metric_id	number(10)	*Metric-tauluun viittaava tunnistenumero*	
item_id	number(200)	*Yhden listan vaihtoehdon tunnistenumero*	
name	varchar(50)	*Yhden listan vaihtoehdon nimi*	

12.14 ProjectMetricSerie

ProjectMetricSerie yhdistää projekteja ja metriikkasarjoja toisiinsa projektin tunnistenumeroiden ja metriikkasarjojen tunnistenumeroiden muodostamilla pareilla. ProjectMetricSerie-

Taulukko 11: MetricModel-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Metriikkamallin nimi*	
description	varchar(2000)	*Metriikkamallin kuvaus*	
ctime	timestamp	*Metriikkamallin luontiaika*	*NOT NULL*
mtime	timestamp	*Metriikkamallin viimeisimmän muokkauksen ajankohta*	*NOT NULL*
type	varchar(50)	*Tietokantaan ajonaikaisesti luotavan atrribuutin tyyppi*	*NOT NULL*

Taulukko 12: Phase-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Projektivaiheen nimi*	
description	varchar(2000)	*Projektivaiheen kuvaus*	
ctime	timestamp	*Vaiheen luontiaika*	*NOT NULL*
mtime	timestamp	*Vaiheen viimeisimmän muokkauksen ajankohta*	*NOT NULL*
abbreviation	char(2)	*Vaiheen lyhenne*	*NOT NULL*

taulun rakenne on kuvattu taulukossa 16.

13 Metadatakuvaus

13.1 Pääkonfiguraatiotiedosto

Pääkonfiguraatiotiedosto sisältää (absoluuttiset) hakemistopolut järjestelmän tarvitsemiin metatiedostoihin. Lisäksi tiedosto sisältää muut järjestelmän käyttämät parametrit. Erillisen konfiguraatiotiedoston avulla järjestelmän siirtäminen uuteen ympäristöön helpottuu merkittävästi. Taulukossa 17 kuvataan tiedoston sisältämät asiat.

Oheinen esimerkki havainnollistaa mahdollista pääkonfiguraatiotiedoston rakennetta.

Taulukko 13: ConcreteMetricSerie-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
metric_id	number(10)	*Metric-tauluun viittaava tunnistenumero*	
metric_serie_id	number(10)	*MetricSerie-tauluun viittaava tunnistenumero*	
attributename	varchar(15)	*Tietokantaan luotavan attribuutin nimi*	
order_nr	number(10)	*Metriikkasarjan metriikoiden keskinäisen esitysjärjestyksen kertova järjestysnumero*	

Taulukko 14: MetricSerieInformation-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
project_metric_serie_id	number(10)	*ProjectMetricSerie-tauluun viittaava tunnistenumero*	
name	varchar(50)	*Metriikkasarjan nimi*	
description	varchar(2000)	*Metriikkasarjan kuvaus*	
ctime	timestamp	*Metriikkasarjan luontiaika*	*NOT NULL*
mtime	timestamp	*Metriikkasarjan viimeisimmän muokkauksen ajankohta*	*NOT NULL*

```
<mainConfiguration>
     <dbddlocation>
         /home/group/otie/webapps/otie/otieDBDD.xml
```

Taulukko 15: MetricSerie-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
name	varchar(50)	*Metriikkasarjan nimi*	
table_name	varchar(15)	*?????*	
description	varchar(2000)	*Metriikkasarjan kuvaus*	
ctime	timestamp	*Metriikkasarjan luontiaika*	*NOT NULL*
mtime	timestamp	*Metriikkasarjan viimeisimmän muokkauksen ajankohta*	*NOT NULL*

Taulukko 16: ProjectMetricSerie-taulu

Attribuutti	*Tyyppi*	*Kuvaus*	*Erikoistietoja*
id	number(10)	*Taulun pääavaimena toimiva tunnistenumero*	*NOT NULL*
project_id	number(10)	*Project-tauluun viittaava tunnistenumero*	
metric_serie_id	number(10)	*MetricSerie-tauluun viittaava tunnistenumero*	
nr	number(10)	*???????*	

```
</dbddlocation>
<securitydesclocation>
    /home/group/otie/webapps/otie/otieSecurityDesc.xml
</securitydesclocation>
<layoutdesclocation>
    /home/group/otie/webapps/otie/otieLayoutDesc.xml
</layoutdesclocation>
<templatedirectory>
    /home/group/otie/webapps/otie/templates
</templatedirectory>
<imagedirectory>
    /home/group/otie/webapps/otie/images/
</imagedirectory>
<relativepath>
    http://db.cs.helsinki.fi/tomcat/otie/servlet
```

Taulukko 17: Pääkonfiguraatiotiedoston sisältämät parametrit.

PARAMETRI	***KUVAUS***
DBDDLOCATION	*Parametri sisältää tiedon tietokanta-asennustiedoston absoluuttisesta tiedostopolusta sekä itse tiedostonimestä.*
SECURITYDESCLOCATION	*Parametri sisältää tiedon tietoturvakuvaustiedoston sijainnista sekä tiedostonimestä.*
LAYOUTDESCLOCATION	*Parametri sisältää tiedon USERAGENT kuvauksista.*
TEMPLATEDIRECTORY	*Parametri sisältää tiedon hakemistosta, josta XML-sivupohjatiedostot ovat löydettävissä.*
LAYOUTDIRECTORY	*Parametri sisältää hakemiston, jossa XSL-tiedostot sijaitsevat.*
IMAGEDIRECTORY	*Parametri sisältää hakemiston, jossa kuvatiedostot sijaitsevat.*
RELATIVEPATH	*Parametri määrittää sovelluksen suhteellisen polun, jota käytetään Web-sivujen tuottamisessa.*
LANGUAGELISTFILE	*Parametri sisältää tiedon kielilistaustiedoston sijainnista.*
TEMPLATEFILE	*Kuvaa miten näkymät liittyvät sivupohjiin.*

```
    </relativepath>
    <languagelistfile>
        /home/group/otie/webapps/otie/langlist.xml
    </languagelistfile>
</mainConfiguration>
```

13.2 Tietokanta-asetustiedosto

Tietokanta-asetustiedostossa kuvataan tietokantayhteyttä varten tarvittavat tiedot. Samalla tietokanta-asetustiedosto toimii *JDBCWrapper*-luokan attribuuttien määrittelynä. Tiedostossa kuvataan käytettävän tietokanta-ajurin nimi sekä tarvittavat tietokantatiedot eli käyttäjätunnus, salasana, tietokannan sijainti (URL), käytettävä tietokantapalvelimen portti (jos tarvetta) sekä käytettävän tietokannan (katalogin) nimi. Tiedoston käyttämät parametrit on kuvattu taulukossa 18. Mikäli halutaan käyttää eritasoisia käyttäjätunnuksia, niin järjestelmään on luotava uusi tietokanta-asetustiedosto ja ohjelmallisesti on huolehdittava, että haluttu asetustiedosto (ja salasana) ladataan ajon aikana.

Oheinen esimerkki havainnollistaa yhtä mahdollista tietokanta-asetustiedoston rakennetta.

```
<database_deployment_descriptor>
<driver>
<driver_filename>com.mysql.jdbc.Driver</driver_filename>
</driver>
```

Taulukko 18: Tietokanta-asetustiedoston sisältämät parametrit.

PARAMETRI	***KUVAUS***
DRIVER_FILENAME	*Tietokanta-ajurin tiedostonimi. Ajurin on oltava Tomcat-palvelimen lib-hakemistossa.*
LOCATION	*Tietokannan osoite (URL).*
USERNAME	*Käyttäjätunnus tietokantaan.*
PASSWORD	*Salasana tietokantaan.*
PORT	*Tietokantapalvelimen käyttämä porttinumero.*
DATABASENAME	*Käytettävän tietokannan nimi.*

```
<database_information>
<location>jdbc:mysql://localhost/</location>
<username>SCOTT</username>
<password>TIGER</password>
<port></port>
<databasename>Demo</databasename>
</database_information>
</database_deployment_descriptor>
```

13.3 Tietoturva-asetustiedosto

Tietoturva-asetustiedosto sisältää tiedot symmetrisessä salauksessa käytettävistä avainaineksista, itse salausalgoritmeista sekä käytettävistä tiivistefunktioista. Tietoturvasyistä tiedoston rakenne kuvataan erillisessä tietoturvadokumentissa, joka palautetaan valmiin järjestelmän yhteydessä.

13.4 Ulkoasuasetustiedosto

Ulkoasutiedostossa määritetään, mikä selain liitetään mihinkin ulkoasutiedostoon. Lisäksi tiedostossa määritetään ns. oletusulkoasutiedosto, jota käytetään jos muuta ei ole määritelty. Tiedostossa määritetään vain pääulkoasutiedoston nimi. Sivukomponentteja varten käytetään toisenlaista mekanismia: jokaiselle määriteltävälle selaimelle annetaan erillinen merkkijononimi, jonka avulla selain liitetään järjestelmään liitettyyn hakemistoon. Haettaessa sivukomponenttien ulkoasupohjaa, haetaan ensisijaisesti määritellystä hakemistosta. Jos pohjaa ei löydy, haetaan sivupohjaa oletushakemistosta. Tiedoston parametrit kuvataan taulukossa 19.

Oheinen esimerkki havainnollistaa yhtä mahdollista ulkoasuasetustiedoston rakennetta.

```
<default>
         <directory>common</directory>
<xsl_file>layout_default.xsl</xsl_file>
```

Taulukko 19: Ulkoasuasetustiedoston sisältämät parametrit.

PARAMETRI	***KUVAUS***
`DIRECTORY`	*Jokaiselle selaimelle määritetään tiedosto, jossa selaimen käyttämät sivupohjat sijaitsevat.*
`XSL_FILE`	*Jokaiselle selaimelle määritetään pääulkoasutiedoston nimi, jota käytetään muutettaessa sivu lopulliseen muotoonsa.*

```
</default>
<user_agent string="Mozilla/5.0 (Windows; U; Win 98 4.90)
                      Gecko/20020502 CS 2000 7.0/7.0">
         <directory>mozilla</directory>
<xsl_file>layout_default.xsl</xsl_file>
</user_agent>
<user_agent string="Nokia7250/1.0 (3.14) Profile/MIDP-1.0
                    Configuration/CLDC-1.0">
        <directory>nokia</directory>
        <xsl_file>layout_wml.xsl</xsl_file>
</user_agent>
```

13.5 Oikeustiedosto

Tietokantaan käyttäjistä tallennettavat käyttöoikeustiedot ovat muotoa *merkkijono1 # merkkijono2 # · · ·*. Varsinainen käyttöoikeuksien tarkistus suoritetaan sivukomponenttien sisällä, joten järjestelmän kannalta tietoa kaikista oikeuslyhenteistä ei tarvita. Kuitenkin, jotta järjestelmän myöhempi käyttö ja laajentaminen olisi mahdollisimman helppoa, palautetaan järjestelmän mukana erillinen tiedosto, jossa on kuvattu kaikki käyttöoikeuslyhenteet.

13.6 Kieliasetustiedosto

Kieliasetustiedostossa kuvataan alalyhenteet, jotka liittävät erikieliset sivut järjestelmään. Oletuskielenä järjestelmässä on suomi ja tällöin tiedostonimien perässä ei tarvita alalyhennettä: esim. *index.xml*. Järjestelmästä ei OhtuTie-projektin puitteissa toteuta muun kielistä versiota, mutta järjestelmä rakennetaan siten, että uuden kielivaihtoehdon lisääminen on suoraviivaista. Esimerkiksi jos järjestelmästä halutaan englannin kielinen versio, niin kieliasetustiedostossa määritetään kielen nimi `ENGLISH` sekä käytettävät alalyhenteet, esim. `EN`. Tällöin järjestelmä tietää, että jos kieleksi on valittu englanti, niin käytetään sivua *index_en.xml* mikäli sellainen on olemassa. Taulukossa 20 kuvataan kieliasetustiedoston parametrit.

Oheinen esimerkki havainnollistaa mahdollisen kieliasetustiedoston rakennetta.

Taulukko 20: Kieliasetustiedoston sisältämät parametrit.

PARAMETRI	***KUVAUS***
LANGUAGE	*Elementti, jonka teksti kertoo lyhenteen:*

```
<language>
    <name>ENGLISH</name>
    <acronym>EN</acronym>
    <acronym>en</acronym>
</language>
<language>
    <name>SWEDISH</name>
    <acronym>SWE</acronym>
    <acronym>SE</acronym>
    <acronym>se</acronym>
</language>
<language>
    <name>DEUTSCH</name>
    <acronym>de</acronym>
</language>
```

14 Käyttöliittymähahmotelma

Luvussa käsitellään yleisellä tasolla OhtuTie-järjestelmän käyttöliittymän rakenne. Kuvissa 21, 22 ja 23 esitetään käyttöliittymän ulkoasuun liittyvät välilehtirakenteet.

OhtuTie-järjestelmän käyttöliittymä suunnitellaan toimimaan yleisillä WWW-selaimilla. Näin ollen sivut koodataan W3C:n XHTML 1.1 -suosituksen mukaisesti ja selainkohtaisia laajennuksia välttäen. Käyttöliittymä tulee olemaan WWW-selaimeen aukeava XHTML-sivu, joka jakautuu otsikkotieto- ja välilehtiosaan. Otsikkotieto-osan muodostaa palkki välilehtisivujen yläpuolella. Otsikkotietopalkissa näytetään käyttäjän perustiedot, tervetulotoivotus sisäänkirjautumisvaiheessa ja linkki ohjetoimintoon. Lisäksi otsikkotietopalkkiin tulevat tekstikentät sisäänkirjautumistietojen syöttämistä varten.

Kurssin vastuuhenkilö huolehtii kunkin käyttäjän tunnuksen luomisesta järjestelmään. Tunnuksen luova henkilö (vastuuhenkilö tai hänen valtuuttamansa henkilö) valitsee tunnukselle sopivan käyttöoikeuden erillisestä järjestelmän tarjoamasta listasta.

14.1 Sisäänkirjautuminen järjestelmään

OhtuTie-järjestelmän käyttö aloitetaan esimerkiksi Ohjelmistotuotantoprojekti-kurssin kotisivuilla olevan WWW-linkin kautta. Sovellus aukeaa linkkiä klikkaamalla, mutta ilman sisäänkirjautumista käyttäjä voi vain katsella järjestelmän tuottamia perusgraafeja.

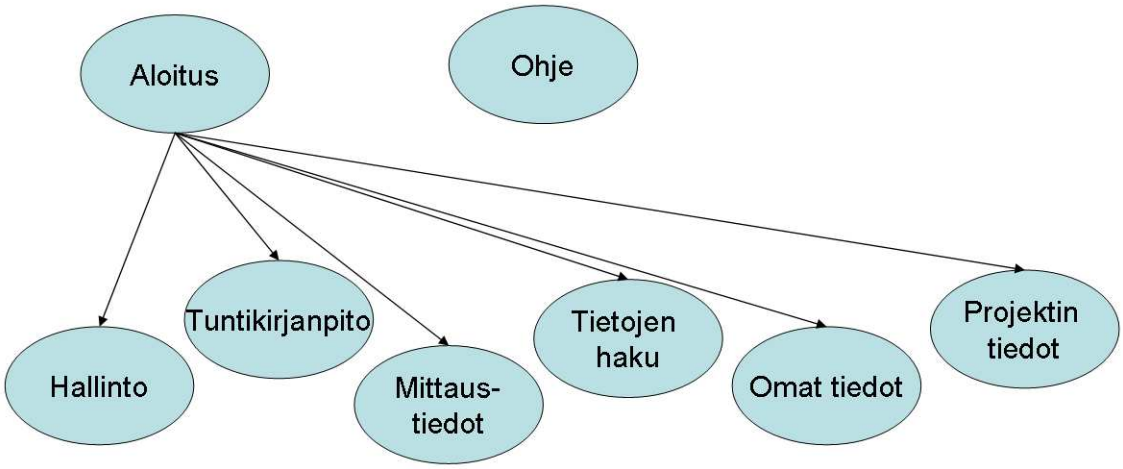
Järjestelmän käyttäjäksi kirjaudutaan etusivulla kysyttävien käyttäjätunnuksen ja salasanan avulla. Sisäänkirjautumisen jälkeen kullekin kirjautujalle latautuu sama avausnäkymä eli välilehtisivu. Se tulee olemaan kaikilla sisäänkirjautuneilla käyttäjillä *Tietojen haku* -välilehti.

14.2 Käyttöliittymän pääosat

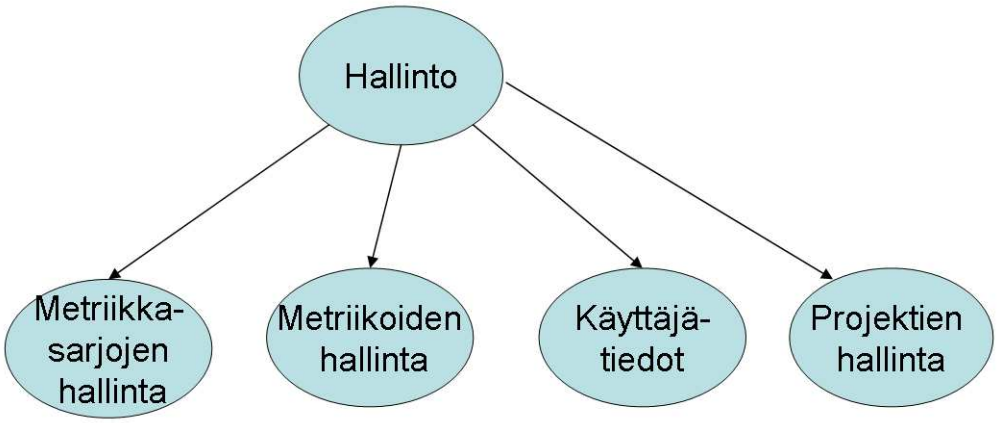
Sovelluksessa ei siis ole varsinaista valikkorakennetta, vaan vastaava toiminnallisuus toteutetaan välilehtinäkymien eli välilehtisivujen avulla. Välilehtisivuja käytetään asiakokonaisuuksien sisältöpohjaiseen ryhmittelyyn ja sisällön ryhmittelyperusteena käytetään aihealuetta eli metriikkasarjaa. Kuvissa 21, 22 ja 23 esitetään käyttöliittymän välilehtirakenteet.

Järjestelmä rajaa käyttäjälle näkyvien välilehtisivujen määrän käyttöoikeuden mukaisesti. Tietojen hallinnointiin tarkoitettu *hallinto*-sivu tulee näkyviin vain vastuuhenkilölle ja ohjaajille. Kaikille käyttäjille näkyvät sivut ovat *mittaustiedot*, *omat tiedot*, *projektin tiedot*, *tuntikirjanpito* ja *tietojen haku*.

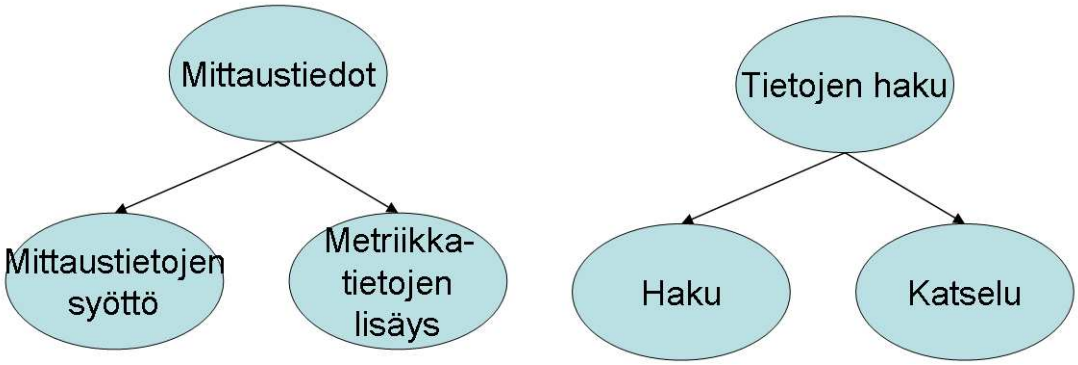
Päävälilehtisivuista *hallinto*-sivulle sekä *mittaustietojen syöttö* ja *tietojen katselu* -sivuille tehdään kullekin omat alavälilehtisivunsa. Näille välilehtisivuille rakennetaan siis eräänlainen sisäinen valikkorakenne alavälilehtien avulla. Hallintosivun alisivuina tulevat olemaan *käyttäjätiedot*, *metriikkasarjojen hallinta*, *metriikoiden hallinta* ja *projektien hallinta*. Metriikkatietosivulla esitetään metriikkasarjat ja niiden muokkaustoiminnot sekä myös tiettyyn sarjaan kuuluvat metriikat ja niiden muokkaustoiminnot. Mittaustiedot-välilehti jakautuu kahteen alavälilehteen. Ensimmäinen alavälilehti on *mittaustietojen syöttö*, jossa näytetään varsinaiset tietojen syöttökentät. Toinen alavälilehti on *metriikkatietojen lisäys*, jossa opiskelija lisää omaan projektiinsa siinä tarvittavat mittaus- eli metriikkasarjat. Tietojen haku -välilehden alavälilehdistä ensimmäinen on *haku* ja toinen *katselu*. Haku-välilehdellä esitetään hakuehdot ja katselu-välilehdellä näytetään tietokannan palauttamat hakutiedot.



Kuva 21: Käyttöliittymän pääosat



Kuva 22: Hallintavälilehden alilehdet



Kuva 23: Mittaustietojen syöttö ja tietojen haku -välilehdet alilehtineen

15 Testaus

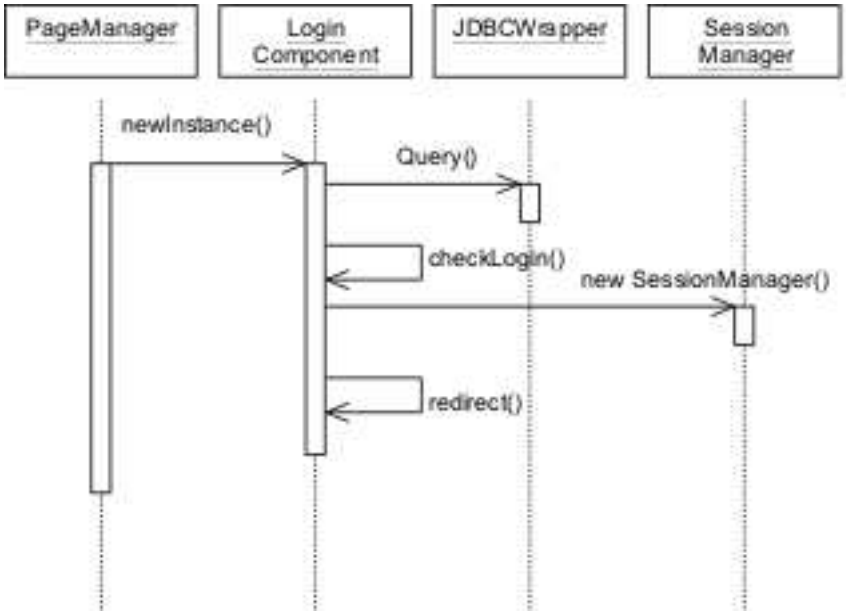
Ohjelmiston testauksen tavoitteena on taata asiakkaalle toimitettavan ohjelmiston virheettömyys ja asiakkaan toiveiden toteutuminen. Koska testaukseen käytettävissä oleva aika on hyvin rajallinen, keskitytään testauksessa ohjelmiston keskeisimpien toimintojen oikeellisuuden varmistamiseen. Kunkin komponentin ohjelmoija testaa samasta syystä itse tuottamaansa koodia. Testauksen tarkkuutta pyritään kuitenkin ylläpitämään liittämällä testien suunnittelu kiinteästi ohjelmointityöhön JUnit-testausohjelman avulla.

Testaus jakautuu kolmeen vaiheeseen. Yksikkötestauksessa varmistetaan pääasiassa JUnit-testaustyökalua hyväksikäyttäen yksittäisten luokkien toimivuus. Integraatiotestauksessa yhdistellään yksikkötestattuja komponentteja ja testataan niiden yhteistoimintaa. Testauksen päättää järjestelmätestaus, jossa testataan koko järjestelmää ja jo päättyneistä projekteista muodostettavan testiaineiston avulla. Testaus kuvataan tarkemmin erillisessä testaussuunnitelmassa.

16 Sekvenssikaaviot käyttötapauksista

16.1 YH1 Kirjautuminen järjestelmään

Kuvassa 24 esitetään sekvenssikaavio järjestelmään kirjautumisesta. Kuvassa ei esitetä järjestelmän arkkitehtuurirakenteen päätietovuota, joka on esitetty kuvassa 6.



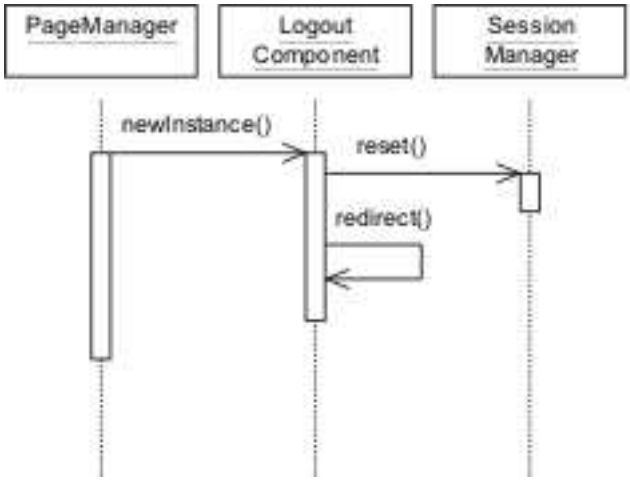
Kuva 24: Sekvenssikaavio järjestelmään kirjautumisesta.

Järjestelmään kirjautuminen vaatii ensiksi käyttäjää syöttämään salasanan ja käyttäjätunnuksen sivun vasemmassa yläkulmassa oleviin tekstikenttiin. Tämän jälkeen käyttäjän painaessa kirjaudu-painiketta, ladataan erillinen kirjautumissivu. Kirjautumissivu koostuu yhdestä *LoginComponent*-komponentista, joka vastaa kirjautumisen tarkistamisesta. Ensiksi siis *PageManager*-komponentti luo uuden *LoginComponent*-olion, jolle välitetään parametrinä käyttäjän syöttämät parametrit (salasana ja käyttäjätunnus). LoginComponent tekee tietokantakyselyn ja saatujen tulostietojen perusteella päättää oliko kirjautuminen onnistunut vai ei. Mikäli kirjautuminen onnistui, ohjataan käyttäjä järjestelmän aloitussivulle *redirect()*-kutsulla. Muuten käyttäjä ohjataan takaisin edelliseen näkymään ja sivulle tulostuu virheteksti, jossa ilmoitetaan kirjautumisen epäonnistumisesta. Onnistuneen kirjautumisen yhteydessä käyttäjälle luodaan uusi istunto, *SessionManager*, johon asetetaan oikeat käyttäjätiedot. Päätösprosessi, eli oliko kirjautuminen onnistunut vai ei, kuvataan erillisessä tietoturvadokumentissa.

16.2 YH2 Kirjautuminen ulos järjestelmästä.

Myös järjestelmästä uloskirjautuminen toteutetaan erillisenä komponenttina. Käyttäjän painaessa *kirjaudu ulos* -painiketta *PageManager*-komponentti luo uuden *LogoutComponent*-

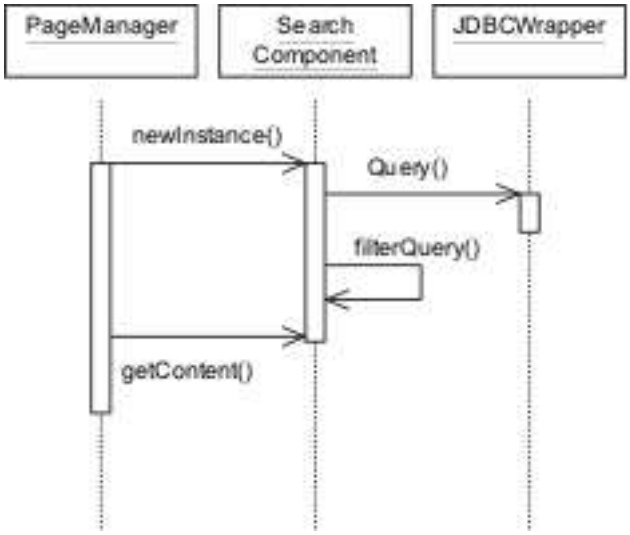
olion. *LogoutComponent* vapauttaa varatut istuntoresurssit ja ohjaa käyttäjän oikealle sivulle.



Kuva 25: Sekvenssikaavio järjestelmästä uloskirjautumisesta.

16.3 YH3 Tietojen haku

Tietojen haku toteutetaan erillisen *SearchComponent*-komponentin avulla. Käyttäjän syötettyä hakuehdot ja painettua *hae*-nappia *PageManager*-komponentti luo uuden *SearchComponent*-olion, joka vastaa kyselyn suorittamisesta. Kysely on standardi tietokantakysely, joten se suoritetaan *JDBCWrapper.Query()*-metodin avulla. Kyselyn palauttamat tiedot muokataan sopivaan muotoon ja *getContent()*-metodin kutsulla PageManager pääsee käsiksi kyselyn tuloksiin, joista generoidaan sivunäkymä.



Kuva 26: Sekvenssikaavio tietojen hakemisesta järjestelmästä.

16.4 OP1 Mittaustietojen syöttäminen

Mittaustietojen syöttäminen tapahtuu erillisen *InputHandler*-luokan kautta. Metriikka tulostetaan ruudulle *MetricHandler*-luokan avulla. Käyttäjän syöttää haluamansa tiedot metriikkaan ja painaa tallenna painiketta, jonka seurauksena ladataan uusi näkymä, joka koostuu vain *InputHandler*-komponentista. InputHandler käsittelee metriikkaan syötetyt tiedot eli tarkistaa syötteiden oikeellisuuden jne. Tämän jälkeen InputHandler lisää syötetyt tiedot tietokantaan *JDBCWrapper.Insert()*-kutsun avulla.

16.5 OP2 Tuntikirjanpidon syöttäminen

Tuntikirjanpito on järjestelmään määritetty metriikka, joten toiminnallisuudeltaan tämä käyttötapaus ei olennaisesti eroa käyttötapauksesta **OP1**.

16.6 OP3 Tuntikirjanpidon syöttäminen erillisestä tiedostosta.

Tuntikirjanpito on järjestelmään määritetty metriikka, joten toiminnallisuudeltaan tämä käyttötapaus ei olennaisesti eroa käyttötapauksesta **OP1**.

16.7 OP4 Henkilötietojen muuttaminen

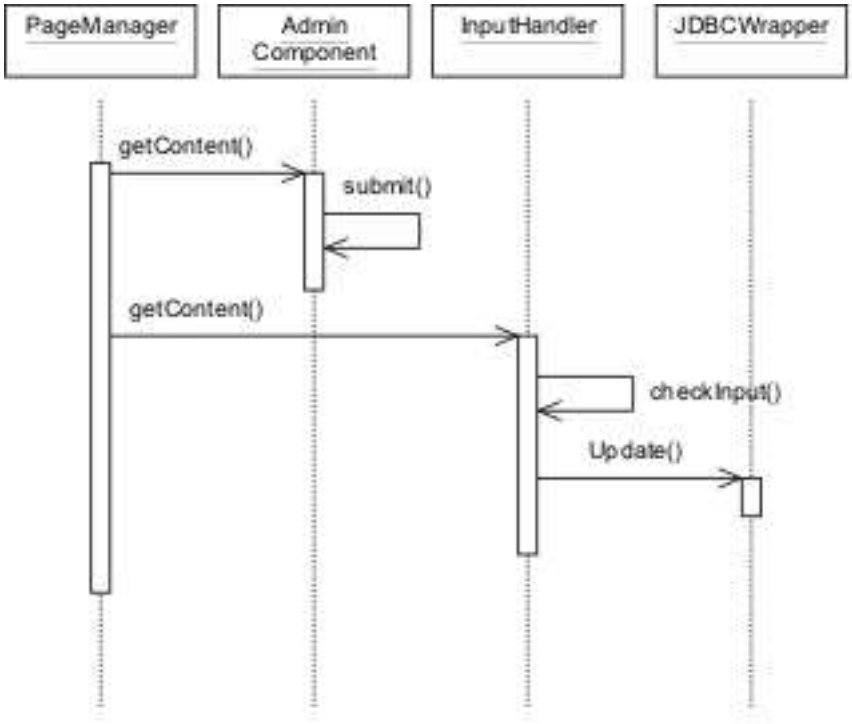
Henkilötietojen lisääminen, muokkaaminen ja poisto toteutetaan saman *AdminComponent*-hallintakomponentin avulla. Komponentti määrittää parametrit ja käytettävän operaation. Kun käyttäjä lähettää muutospyynnön palvelimelle *PageManager* komponentti luo uuden *InputHandler*-olion, joka huolehtii mm. syötteiden tarkistamisesta. Jos syöte on kunnossa, tietokanta päivitetään *JDBCWrapper.Update()*-metodin avulla. Vastaavasti voidaan käyttää *JDBCWrapper.Update()* ja *JDBCWrapper.Insert()* -metodeja. Kaaviossa 27 on esitetty sekvenssikaavio tietojen muokkaamisesta.

16.8 OP5 Projektin tietojen muuttaminen

Projektin tietojen muuttaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.

16.9 OH1 Opiskelijan lisäys ja opiskelijatietojen muokkaaminen

Käyttötapaus vastaa toiminnallisesti käyttötapausta **OP4**, joka on kuvattu kaaviossa 27.



Kuva 27: Sekvenssikaavio tietojen muokkaamista varten.

16.10 OH2 Projektin tietojen muokkaaminen

Projektin tietojen muokkaaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.

16.11 OH4 Arvostelu

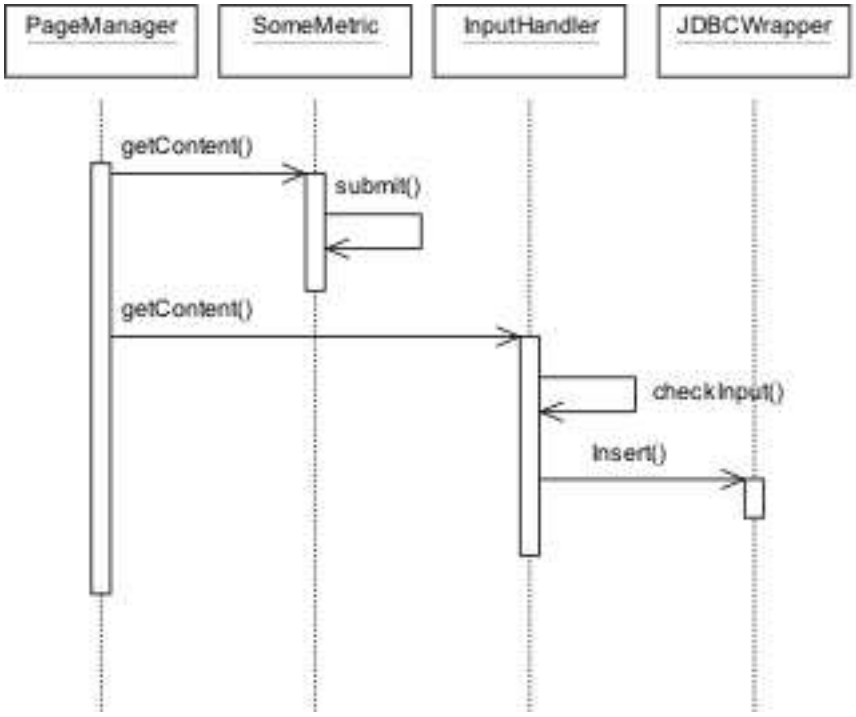
Arvostelu toteutetaan metriikkana, joten käyttötapaus vastaa mittaustietojen syöttö käyttötapausta (**OP1**), joka on kuvattu kaaviossa 28.

16.12 OH5 Asiakkaan arvostelun syöttäminen

Asiakaan arvostelu toteutetaan metriikkana, joten käyttötapaus vastaa mittaustietojen syöttö käyttötapausta (**OP1**), joka on kuvattu kaaviossa 28.

16.13 VH1 Projektitietojen muokkaaminen

Projektitietojen muokkaaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.



Kuva 28: Sekvenssikaavio mittaustietojen syöttämiseksi.

16.14 VH2 Metriikkatietojen muokkaaminen

Metriikkatietojen muuttaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.

16.15 VH3 Metriikkasarjojen tietojen muokkaaminen

Metriikkasarjojen tietojen muuttaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.

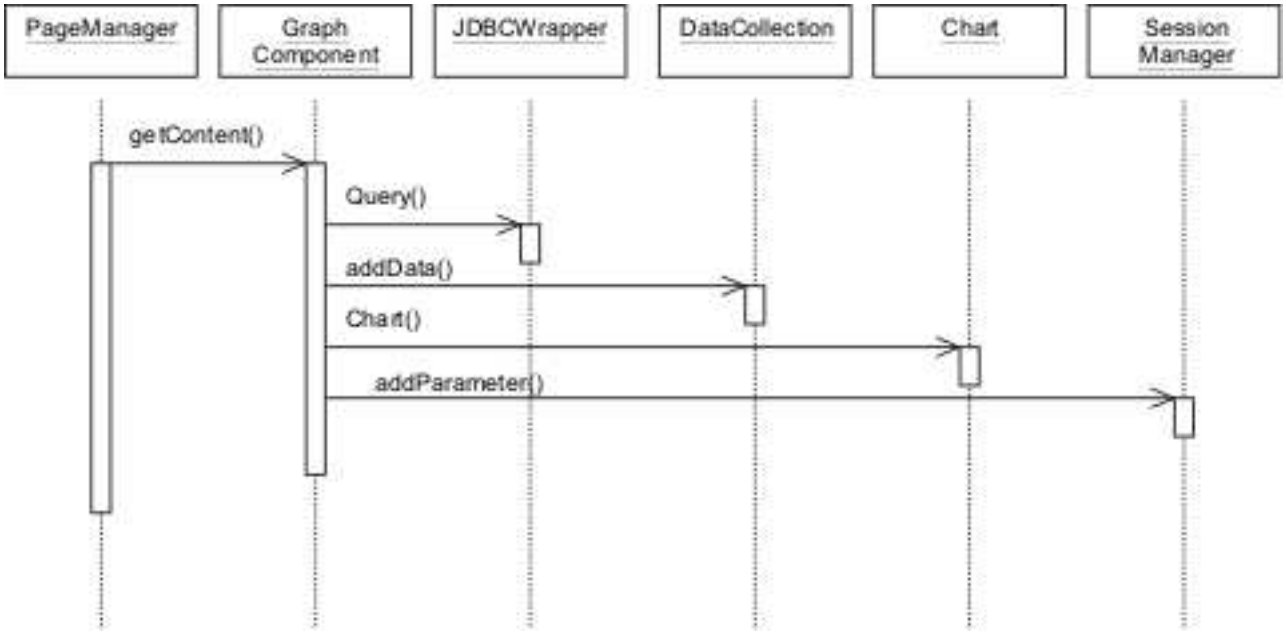
16.16 VH4 Käyttäjätietojen muokkaaminen

Käyttäjätietojen muuttaminen tapahtuu saman prosessin välityksellä kuin henkilötietojen muokkaaminen. Kaaviossa 27 esitetään käyttötapauksen toiminnot.

16.17 VH6 Ajankäyttökaavioiden tulostaminen

Ajankäyttökaavioiden generointi on yksi järjestelmän monimutkaisimmista prosesseista. Ensiksi *PageManager*-komponentti luo *GraphComponent*-olion, joka vastaa kyselyjen suorittamisesta ja graafin piirtämisestä. *GraphComponent*-olio tekee tietokantakyselyn *JDBCWrapper.Query()*-metodin avulla. Kyselyn tuloksista *GraphComponent*-olio luo

uuden *DataCollection*-olion, jonka avulla komponentti luo uuden *Chart*-olion, joka voidaan näyttää ruudulla. Lisäksi istuntotietoihin tallennetaan aikaleima siitä, milloin kuva on viimeksi ladattu. Tämä toiminnallisuus on tarpeen siksi, että kuvaa ei tarvitsisi joka kerta generoida tyhjästä, vaan se voidaan tallentaa muistiin ja generoida uudestaan vain tarvittaessa.



Kuva 29: Sekvenssikaavio ajankäyttökaavioiden tulostamiseksi.

Liite 1. Metriikkamallit

Metriikka on mitattava tieto. Metriikalla on nimi, kuvaus ja tyyppi (metriikkamalli).

Esimerkkinä metriikka koodirivien lukumäärä, joka käyttää metriikkamallia "kokonaisluku", sekä metriikka käytetty ohjelmointikieli, joka käyttää metriikkamallia "kokonaisluku nimetyillä alkioilla".

Kullakin metriikkamallilla on ulkoasu, joka määrittää, miten se näkyy käyttäjälle käyttöliittymässä.

Kaikilla metriikkamalleilla olevat tiedot
- metriikkamallin tyyppi / arvotyyppi
- metriikan nimi (näkyy käyttöliittymässä)
- metriikan ohje (voi näkyä käyttöliittymässä)

Sanastoa:
Kokonaisluku nimetyillä alkioilla = esim. vaihtoehdot 1,2,3, 1=vesiputous, 2=iteratiivinen...
Kokonaisluku nimetyillä sarakkeilla = metriikan nimi on sarakkeen nimi.
KokonaislukuTaulukko nimetyillä sarakkeilla = kunkin sarakkeen päällä on nimi

Taulukko 21: Metriikkamallit

M1	Tekstikenttä
M2	Tekstialue
M3	Kokonaisluku
M4	Kokonaisluku nimetyillä alkioilla
M5	Totuusarvo
M6	Päivämäärä

Taulukko 22: Valmiiden metriikoiden käyttämät metriikkamallit OhtuTie-määrittelydokumentissa kuvattujen käyttötapausten mukaisesti ryhmiteltynä.

	Metriikka	***Metriikkamalli***
A1 projektin tiedot		
	projektin nimi	M1
	projektin kotisivu	M1
	projektin lukukausi ja vuosi	Ei toteuteta metriikkana
	projektin tyyppi	M4
	käytetty projektimalli	M4
B1 projektin tiedot		
	projektimalli	M4
	pääohjelmointikieli	M4
	muut projektissa käytetyt tekniikat	M1
	palaverien määrä	M3
	ryhmäpalaverien määrä	M3
	suunnittelupalaverien määrä	M3
	katselmointien määrä	M3
	asiakastapaamisten määrä	M3
B2 tuntikirjanpito		
		Ei toteuteta metriikkana
B3 Määrittely		
	järjestelmävaatimusten lukumäärä prioriteetin mukaan jaoteltuna	AVOINNA
	käyttötapausten lukumäärä	M3
	toiminnallisten vaatimusten lukumäärä	M3
	laadullisten vaatimusten lukumäärä	M3
B4 Suunnittelu		
	suunniteltujen vaatimusten lukumäärä prioriteetin mukaan jaoteltuna	AVOINNA
	muutettujen ominaisuuksien lukumäärä	M3
	lisättyjen ominaisuuksien lukumäärä	M3
	luokkien lukumäärä	M3

Taulukko 22: (jatkuu)

	Metriikka	***Metriikkamalli***
B5 Toteutus		
	koodirivien lukumäärä	M3
	ohjelmakoodia sisältävien rivien lukumäärä	M3
	tyhjien rivien lukumäärä	M3
	kommenttirivien lukumäärä	M3
	muilla kuin pääasiallisella ohjelmointikielellä kirjoitettujen rivien lukumäärä	M3
	toteutuneiden suunnitteludokumentissa mainittujen luokkien lukumäärä	M3
	toteutuneiden suunnitteludokumentissa mainitsemattomien luokkien lukumäärä	M3
B6 Testaus		
	yksikkötestattujen luokkien lukumäärä	M3
	testattujen koodirivien lukumäärä	M3
	yksikkötestauksessa löydettyjen virheiden lukumäärä	M3
	integraatiotestauksessa löydettyjen virheiden lukumäärä	M3
	järjestelmätestauksessa löydettyjen virheiden lukumäärä	M3

Taulukko 22: (jatkuu)

	Metriikka	***Metriikkamalli***
B7 Tarkastus		
	tarkastettujen sivujen lukumäärä	M3
	tarkastuksen kesto minuutteina	M3
	projektin vaihe johon tarkastus liittyy	M4
	löydettyjen puutteiden lukumäärä: toiminnallisuus ja logiikka	M3
	löydettyjen puutteiden lukumäärä: I/O ja syötteen sekä tulosteen käsittely	M3
	löydettyjen puutteiden lukumäärä: ylläpidettävyys	M3
	löydettyjen puutteiden lukumäärä: rajoitteiden noudattaminen	M3
	löydettyjen puutteiden lukumäärä: käytettävyys	M3
	löydettyjen puutteiden lukumäärä: muut	M3
	löydettyjen virheiden lukumäärä: toiminnallisuus ja logiikka	M3
	löydettyjen virheiden lukumäärä: I/O ja syötteen sekä tulosteen käsittely	M3
	löydettyjen virheiden lukumäärä: ylläpidettävyys	M3
	löydettyjen virheiden lukumäärä: rajoitteiden noudattaminen	M3
	löydettyjen virheiden lukumäärä: käytettävyys	M3
	löydettyjen virheiden lukumäärä: muut	M3

A Tekstikenttä

```
- arvotyyppi: String
- syötteen maksimipituus

esimerkki
nimi: Projektin nimi

-ulkoasu
nimi: tekstikenttä
```

B Tekstialue

```
-arvo String

esimerkki
nimi: Projektissa käytettävät muut tekniikat (vapaa tekstisyöte
      - vrt. pääohjelmointikieli)

-ulkoasu
nimi: tekstikenttä
```

C Kokonaisluku

```
- arvotyyppi : int
- has_minimiarvo : boolean
- has_maksimiarvo : boolean
- minimiarvo : int
- maksimiarvo : int

- ulkoasu
nimi: tekstikenttä
```

D Kokonaisluku nimetyillä alkioilla

```
- arvotyyppi : int
- alkioiden määrä : int
- alkioiden nimet : String[]

esimerkki
nimi: Käytetty projektimalli
alkioiden nimet:
1="Syklinen",
```

2="Vesiputous",
3="Muu"

- ulkoasu:
nimi: valintalista jossa nimet

E Totuusarvo

- arvotyyppi: boolean
- ulkoasu:
nimi: checkbox

F Päivämäärä

- arvotyyppi: Date
- ulkoasu:
tekstikenttä, tai valintalistat vuodelle, kuukaudelle ja päivälle.

Liite 2. Metriikkojen tulostaminen

A Metriikkasarjan tulostaminen

Metriikkasarjalle tulostetaan mahdolliset nimi ja ohjekentät sekä avataan nimetty lomake.

Kullekin metriikkasarjan metriikalle tulostetaan vastaavaa metriikkamallia vastaava pohja. Sen kentät nimetään tietokantaa vastaavasti ja niille annetaan alkuarvoksi mahdollinen tietokannasta löytyvä arvo.

Käyttäjälle merkitään ajankohta, jolloin tiedot on haettu tietokannasta. Tieto voidaan joko lähettää selaimeen piilokenttänä tai säilyttää palvelimen istuntotiedoissa. Kun käyttäjä lähettää muuttamansa tiedot, verrataan tätä ajankohtaa tietokannasta löytyvään. Jos tietokannassa kyseisten tietojen muokkausajankohta on uudempi kuin käyttäjältä tuleva aika, annetaan käyttäjälle virheilmoitus, että joku muu on muokannut tietoja, ja hylätään päivitys.

Metriikkasarjalle tulostetaan *Talleta*- tai *Lähetä*-nappi, johon annetaan arvo, josta järjestelmä lähetettäessä tunnistaa mikä metriikkasarja on kyseessä. Lopuksi tulostetaan lomakkeen sulkeminen.

Esimerkki metriikkamallin tulosteesta:

```
<form name="project_information" method="post" action="kasittelija">
< HIDDEN name="timestamp" type=text" value="15.6.2004 12:31:24"
(tai koodattuna)>

metriikka1
metriikka2
...

<input type="submit" value="Tallenna" /> <br />
<input type="reset" value="Palauta alkuarvot" /> <br />
</form>
```

B Metriikoiden tulostaminen

Metriikka tulostetaan metriikkasarjan jo avattuun lomakkeeseen, josta se saa myös lähetysnapin. Metriikan tulostamiseen käytetään metriikkamallin HTML-pohjaa, johon lisätään metriikan nimi, ohje, sekä tietokannasta haetut arvot oletusarvoiksi.

Esimerkki metriikan tulosteesta:

```
Projektin nimi <input type="text" name="project_name" \\
value="" /> <br />
```

Liite 3. Metriikkamallien HTML-ulkoasu

A Tekstikenttä

```
$metric_instruction
$metric_name <input type="text" name="$metric_id"
value="metric_value" maxlength="$maxlength" /> <br />
```

B Tekstialue

```
$metric_instruction
$metric_name <textareaname="$metric_id" rows=""
cols="" /> <br />
input here
</textarea>

- HTML:n tekstikentälle ei käytössä value-arvoa.
```

B.1 Kokonaisluku

```
$metric_instruction
$metric_name  <input type="text" name="$metric_id"
value="metric_value" /> <br />
```

C Kokonaisluku nimetyillä alkioilla

```
$metric_instruction
$metric_name <select name="$metric_id">
while (has more element)
<option value="n" label="$element_name[n]" jos defaultarvo
(selected="selected")>
$element_name[n]
</option>
</select>
```

D Totuusarvo

```
$metric_instrucion
<input type="checkbox" name="$metric_id" value="$metric_id" />
$metric_name
```