SUUNNITTELUMALLIT OHJELMISTOTUOTANNON TUKENA

Seija Alho

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Kati Iltanen
Joulukuu 2014

Tampereen yliopisto
Informaatiotieteiden yksikkö, Tietojenkäsittelyoppi
ALHO, SEIJA: Suunnittelumallit ohjelmistotuotannon tukena
Pro gradu -tutkielma, 48 sivua, 3 liitesivua.
Joulukuu 2014

Tarkastelen tutkielmassani relaatiotietokannan päivityksen avulla ohjelmoinnin suunnittelumallin soveltamista kerrosarkkitehtuuriympäristöön. Kirjallisuuden perusteella esittelen toteutusympäristön ja työvälineet: UML-mallinnuskieli, relaatiotietokannan suunnittelu, yleisesti ohjelmistoarkkitehtuurit ja olio-ohjelmoinnin suunnittelumalleista Tehdasmetodi.

Tutkielmassani esitän, että Tehdasmetodilla kerrosarkkitehtuuriin toteutettu päivitys relaatiotietokantaan on monipuolisempi kuin suora päivitys relaatiotietokantaan. Toteutan tutkimuksen tiukasti rajatuilla päivitysesimerkeillä. Suunnitellut päivitykset kuvaan UML:n viestiyhteyskaaviolla. Tehdasmetodilla ja ilman suunnittelumallia suunnitellut päivitykset siirrän kerrosarkkitehtuuriympäristöön. *ATAM-arviointimenetelmällä* (architecture tradeoff and analysis method) vertailen suunnitelmien välisiä eroja. Raportoin arvioinnin löydökset.

Tutkimuksen perusteella voin todeta, että Tehdasmetodin käyttäminen relaatiotietokannan päivityksessä tuo monia etuja: selkeyttää ohjelmakoodia, ohjelmakoodi on uudelleenkäytettävää, dokumentoitavuus paranee ja toiminto on helposti toistettavissa. Haittapuolina nousi esiin ohjelman suorituksen hitaus ja virhetilanteiden hallinta. Suunnittelumalleja käyttävän yrityksen henkilöriippuvuus vähenee, kun ohjelmistot ovat hyvin dokumentoituja ja ratkaisut käyttävät vakiintunutta ohjelmointitapaa. Työn lisäksi säästyy myös rahaa, kun ohjelmiston komponenttien osia voidaan käyttää uudelleen.

Avainsanat: relaatiotietokanta, suunnittelumalli, Tehdasmetodi, kerrosarkkitehtuuri, UML

ESIPUHE

Kiitokset Seinäjoen Yliopistokeskuksen Vakava-hankkeelle, jossa olen saanut olla mukana. Hanke sai minut innostumaan pro gradun kirjoittamisesta ja tutkintoni loppuun saattamisesta. Mikko Männikön graduryhmässä on ollut mukava käydä. Ryhmän vertaistuki on ollut erinomaista. Kiitokset myös ryhmän jäsenille tuesta.

Kiitokset myös Tampereen yliopiston Erkki Mäkiselle ja muulle henkilökunnalle opastuksesta opintojeni päivittämisessä ajan tasalle ja avusta paluulleni yliopistomaailmaan vuosien poissaolon jälkeen.

Ohjaajalleni Kati Iltaselle kuuluu iso kiitos. Hänen ohjeillaan ja kannustavalla tuellaan pro graduni toteutui.

Kurjenkylässä 02.12.2014

Seija Alho

SISÄLLYS

1. JOHDANTO

Tieto on laaja käsite. Me kaikki olemme sen kanssa päivittäin tekemisissä. Tietoa varastoidaan, siirretään, muokataan ja käytetään. Yritämme päivittäin järjestää tietokaaostamme. Käsittelen pientä siivua tästä laajasta tietoavaruudesta: tiedon olemusta ohjelmistotuotannon suunnittelussa ja hyödyntämistä käytössä.

Tutkielmassa esitellään suunnittelussa käytettyjä työvälineitä, ympäristöä, tutkimuskysymyksiä ja tuloksia. Suunnitteluympäristöä kuvataan aloittamalla korkean tason arkkitehtuuriympäristöstä. Arkkitehtuuriympäristöksi on valittu kerrosarkkitehtuurityyli. Kerrosarkkitehtuuri on yleisesti käytetty arkkitehtuurityyli, ja sen hierarkkinen rakenne on helposti ymmärrettävä. Arkkitehtuurityyliä tukevat erilaiset suunnittelumallit. Kerrosarkkitehtuuria tukevaksi suunnittelumalliksi on tässä valittu luontimalleihin kuuluva *Tehdasmetodi* (factory method). Tästä jatketaan matkaa tarkentuvaan suunnitteluun: tiedon varastoimiseen. Relaatiotietokanta on maailmalla yksi suosituimpia tietokantamuotoja. Tutkimuksessa kuvataan relaatiotietokannan suunnittelua.

Toteutuksessa on käytetty *UML-mallinnuskieltä* (Unified Modeling Language). UML-mallinnuskieli on valittu työvälineeksi, koska se on yleisesti tunnettu mallinnuskieli. UML-mallinnuskielen etuna on laitteisto-, käyttöjärjestelmä-, ohjelmointikieli- ja kehitysympäristöriippumattomuus. UML-mallinnuskieltä tullaan käyttämään sekä tietokantasuunnitelman mallintamiseen että suunnittelumallin kuvaamiseen.

Tutkielman ensimmäisessä tutkimustehtävässä esitetään relaatiotietokannan päivityksen osamallinnus UML-kaavioiden avulla. UML-kaaviot jakautuvat kahteen esitysosaan: ilman suunnittelumallia tehty suora päivitys ja Tehdasmetodi-suunnittelumallilla tehty päivitystoteutus. Molemmissa osissa ovat UML:n luokka- ja viestiyhteyskaaviot kuvausvälineinä. Toisena tutkimustehtävänä on sijoittaa ja vertailla suunnittelumallilla ja suoralla päivityksellä tehtyjä relaatiotietokannan päivityksien toteutuksia kerrosarkkitehtuuriympäristössä. Vertailussa käytetään soveltaen arkkitehtuurin arviointimenetelmää (architecture tradeoff and analysis method) ATAM.

Aikaisemmat tutkimukset ovat sisältäneet yksittäisinä tai useamman aiheen yhdistelminä tutkimuksessa käsiteltäviä osa-alueita. Relaatiotietokantoja, arkkitehtuurityylejä ja suunnittelumalleja on tutkittu paljon; samoin kerrosarkkitehtuuria ja eri suunnittelumalleja on käytetty ohjelmointiprojektitutkimuksissa. Tehdasmetodia ei ole käytetty löytämissäni ohjelmointi- tai suunnitteluprojekteissa. Tässä työssä on tutkimuksen kohdetta tiukasti rajaamalla pyritty pelkistämällä näyttämään päivitystoteutuksen tulos.

Tutkimuksen rakenne on seuraava. Ohjelmiston arkkitehtuurityylejä käsitellään luvussa 2. *Suunnittelumalleja* (design patterns) esitellään yleisellä tasolla luvun 3 alussa. Luvussa 3 esitellään tarkemmin luontimalleihin kuuluva Tehdasmetodi. Myöhemmin luvussa 8 Tehdasmetodi-suunnittelumallia sovelletaan käytännössä tilausjärjestelmän esimerkissä. Relaatiotietokannan suunnittelua käsitellään luvussa 4. Luvussa 5 käydään läpi UML-mallinnuskieltä. UML-mallinnuskielestä esitellään keskeisimpiä kaavioiden piirtämisessä tarvittavia osia. Arkkitehtuurin arviointimenetelmää (ATAM) esitellään lyhyesti luvussa 6. Luvussa 7 kuvaillaan tutkimuksen aihetta ja tutkimustehtäviä. Luvussa 8 toteutetaan tutkielman ensimmäinen tutkimustehtävä. Toinen tutkimustehtävä toteutetaan luvussa 9. Luvun 10 lyhyessä yhteenveto-osiossa esitellään tutkimustehtävät, niiden toteutus ja tulokset. Loppupäätelmät esitetään luvussa 11.

2. ARKKITEHTUURITYYLIT

Arkkitehtuurimallit (architectural patterns) ovat koko järjestelmää luonnehtivia korkean tason sovellettavia suunnitteluratkaisuja. [Buschmann *et al*., 1996] Arkkitehtuurimalleja voidaan kutsua termillä arkkitehtuurityyli. [Koskimies ja Mikkonen 2005, s. 104]

2.1 Arkkitehtuurityyliryhmät

Arkkitehtuurityylit vaikuttavat parantavasti ohjelmiston laatuun ja ohjelmistokehitysprosessiin. Koskimies ja Mikkonen [2005, s. 9] jakavat arkkitehtuurityylit kolmeen eriperustaiseen ryhmään: arkkitehtuurityyli ryhmittelyn perustana, palveluperustaiset ja sovelluskohtaiset arkkitehtuurityylit.

2.1.1 Ryhmittelevät arkkitehtuurityylit

Arkkitehtuurityyliä voidaan käyttää apuna järjestelmien hahmottamisessa. Tällöin arkkitehtuurityylissä on mukana ryhmittely- tai hahmotustekniikkaa varsinaisen toteutuksen rakenteen selityksen lisänä. *Kerrosarkkitehtuuri* ja *tietovuoarkkitehtuuri* ovat tärkeimmät ryhmittelyyn käytetyt arkkitehtuurityylit.

Kerrosarkkitehtuuri on hyvin monipuolinen: sillä voidaan kuvata lähes mikä tahansa tietojärjestelmä. Kerrosarkkitehtuuri koostuu tasoista. Kerrosarkkitehtuurin perusajatuksena on tasolla olevan komponentin tai palvelun toteuttaminen käyttämällä alemman kerroksen tarjoamia komponentteja tai palveluja. Joskus joudutaan ohittamaan tasoja tehokkuussyistä, koska palvelu löytyy tehokkaampana alemmilta kerroksilta. Ohituksia tehdään myös, jos palvelua ei ole saatavilla seuraavaksi alemmalta kerrokselta. Kerrosarkkitehtuuritasoja voidaan kuvata kerrosvoileivällä. Tasojen ohitukset voidaan kuvata porrastamalla. Kerrosvoileivän päät taivuttamalla saadaan ympyränmuotoinen kuvaus. Tasot on järjestetty abstrahointiperiaatteen mukaan nousevaan järjestykseen. Laite-päässä olevat tasot ovat matalammalla kuin ihmistä lähellä olevat tasot. [Koskimies ja Mikkonen 2005, ss. 125-127] Kerrosarkkitehtuuri esitellään tarkemmin kohdassa 2.2.

Tietovuoarkkitehtuuria (pipes-and-filter architecture) sovelletaan tietovirtaa jalostavissa ja prosessointitoimintajärjestelmissä. Tietovuoarkkitehtuurissa on itsenäisiä tietoa käsitteleviä prosessointiyksiköitä (filter) ja passiivisia tietoa kuljettavia väyliä (pipe). Väylät toimivat prosessiyksiköiden välillä yhdistävänä tietovirtana. Tietovuoarkkitehtuurin soveltamisessa on etuna, että tiedon prosessointi voidaan jakaa pieniin, helposti hallittaviin osiin. Toinen etu on prosessiyksiköiden järjestyksen joustava vaihtaminen ja yksiköiden korvattavuus toisella yksiköllä. [Koskimies ja Mikkonen 2005, ss. 132-135]

2.1.2 Palveluperustaiset arkkitehtuurityylit

Palveluperustainen arkkitehtuurityyli jakautuu palvelun tarjoajan ja palvelun käyttäjän rooleihin. Roolijako ei ole tiukka, palvelun tarjoaja voi toimia toisen palvelun käyttäjänä. Usein palvelu on resurssi, jonka ympärille on rakennettu palvelua tarjoava ohjelmistokomponentti.

Asiakas-palvelin -arkkitehtuuri on yleinen suosittu arkkitehtuuriratkaisu. Periaatteena on palvelimelle sijoittuva resurssien hallinta, käyttäjät pyytävät palvelimelta tiettyä resurssia käyttöönsä toistensa pyynnöistä riippumatta. Hajautetut järjestelmät käyttävät usein asiakas-palvelin -arkkitehtuuria, koska sen selkeä työnjako muodostaa toimivan pohjan myös hajautukselle. [Koskimies ja Mikkonen 2005, ss. 136-137]

*Viestinvälitysarkkitehtuurin (*message dispatcher arhitecture, implicit invocation architecture*)* perusajatus on komponenttijoukon kommunikointi keskenään keskitetyn väylän tai viestinvälittäjän avulla. Komponenteilla on yhteinen rajapinta ja operaatiot viestien vastaanottamiseen. Viestit sisältävät tiedon komponenttien tehtävistä. Viestien reititykseen voidaan käyttää välittäjälle rekisteröityjä tietoja, konfiguraatiotiedostoja, jonoja tai postilaatikoita. Viestinvälitysarkkitehtuurissa ei rooleja kiinnitetä niin kuin tehdään asiakas-palvelin -arkkitehtuurissa. [Koskimies ja Mikkonen 2005, s. 139]

2.1.3 Sovellusaluekohtaiset arkkitehtuurityylit

Sovelluskohtaisen arkkitehtuurityylin sovelluksista on löydettävissä tietynlaisia vakiintuneita toteutustapoja. Sovelluskohtaisen arkkitehtuurityylin toteutuksessa sovellusalue rakennetaan palveluiden ja jonkinlaisen kerrostamisen varaan.

Malli-näkymä-ohjain -arkkitehtuurin (MVC, model-view-controller) ajatuksena on pitää erillään käyttöliittymä, sovelluslogiikka ja sovellusdata. Arkkitehtuurissa saavutetaan itsenäiset osa-alueet ja helppo ylläpidettävyys, esimerkiksi käyttöliittymän vaihtaminen. MVC-arkkitehtuurin osat ovat sovellusdataa tai loogista sovelluksen tilaa edustavat mallit, käyttöliittymää edustavat näkymät ja edellisten välissä toimivat sovittelevat ohjaimet. [Koskimies ja Mikkonen 2005, ss. 142-143]

Tietovarastoarkkitehtuurissa (repository architecture) perusajatuksena on järjestelmä- tai komponenttijoukon tietovarastossa ylläpitämä yhteinen tila. Jaettu tietovarasto on tietovarastoarkkitehtuurin perusta. Järjestelmän komponentit pääsevät tutkimaan ja muuttamaan tietovarastoa. Komponenttien kommunikointi ei tapahdu suoraan vaan tietovaraston kautta. Uusia komponentteja tai sovelluksia on helppo lisätä ja poistaa tietovarastosta. Esimerkkinä voisi olla erilaiset teksti- ja julkaisujärjestelmät, joissa samaa tekstin sisäistä esitystä käsittelee joukko työkaluja. [Koskimies ja Mikkonen 2005, s. 145]

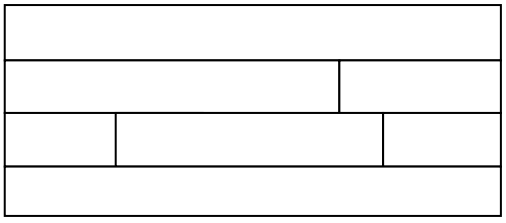
Tulkkipohjaisessa arkkitehtuurissa ajatuksena on, että tulkki lukee ja suorittaa tietyn muodon mukaista toiminnallista kuvausta käyttäen toteutusalustan palvelua. Toteutusalustana voidaan käyttää suoraan toteutuskieltä. Tulkkiarkkitehtuuri tekee ajon aikana koodista tietorakenteen, jota järjestelmä hallinnoi. Tulkkipohjaista arkkitehtuuria käytetään, kun halutaan sovelluksen toimivan erilaisilla alustoilla, sovelluksessa on peruspalvelujen ajoaikaista yhdistelyä tai sovelluksessa on abstrakteja toiminnallisuuksia. Tulkkiarkkitehtuurin sovellus on esimerkiksi Java-ohjelmointikieli, joka perustuu virtuaalikoneohjelmointijärjestelmään. [Koskimies ja Mikkonen 2005, ss. 146-147]

2.2 Kerrosarkkitehtuuri

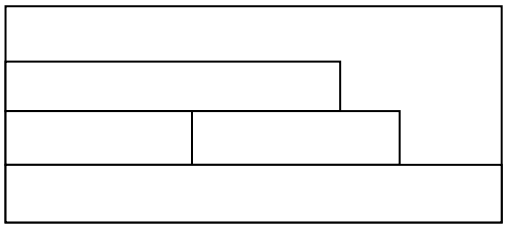
Kerrosarkkitehtuuri koostuu tasoista, joita voidaan myös nimittää kerroksiksi. Kerrokset sisältävät komponentteja tai yksittäisiä palveluja, joita toteutetaan alemman kerroksen tarjoamilla komponenteilla tai palveluilla. Tasot on järjestetty abstrahointiperiaatteen mukaan nousevaan järjestykseen. Esimerkkinä usein käytetyssä laite-ihminen -abstrahointiperiaatteessa ylemmällä abstraktiotasolla on graafisen käyttöliittymän palvelut ja laiteläheiset ovat alemmalla tasolla tarjoten laitteen tai käyttöjärjestelmän tarvitsemia yksinkertaisia toimintoja.

Kerrosarkkitehtuurin ryhmittelevä rakenne selkeyttää järjestelmän kokonaisuuden ymmärtämistä. Kerrokset voivat koostua käyttösuhteiden kannalta samassa asemassa olevista komponenteista. Puhtaat kerrosarkkitehtuurirakenteet ovat harvinaisia. Poikkeamissa palvelukutsut voivat kulkea alemmasta kerroksesta ylempään (hierarchy breach) tai palvelukutsu voi ohittaa kerroksia alaspäin kulkevissa kutsujärjestyksissä (bridging). Ohituksen syynä voi olla, ettei palvelua ole saatavana ko. kerroksessa tai alemmista kerroksista voi löytyä tehokkaampia palveluja toiminnon suorittamiseen. Tasojen ohitukset voidaan kuvata porrastamalla.

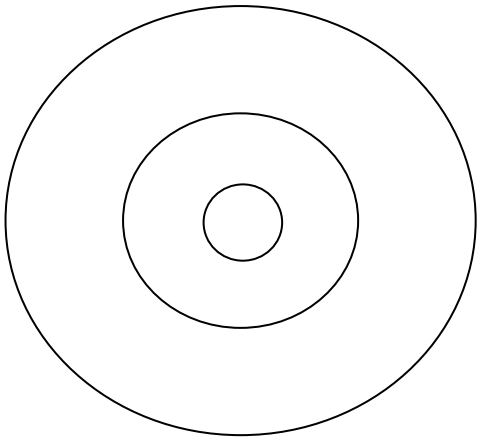
Kutsun kulkeminen alemmasta kerroksesta ylempään aiheuttaa alemmalle kerrokselle ongelmallisen riippuvuuden ylemmästä kerroksesta. Takaisinkutsuperiaatteella torjutaan riippuvuuden syntyä. Takaisinkutsussa alempi kerros tarjoaa rekisteröintioperaation ylemmän kerroksen käyttöön, jotta alempi kerros saisi ylemmän kerroksen rekisteröimän koodin käyttöönsä. [Koskimies ja Mikkonen 2005, ss. 126-127]



Kuva 2.1 Kerrosarkkitehtuurin kerrosvoileipämuoto, ei ohituksia



Kuva 2.2 Kerrosarkkitehtuurin kerrosvoileipämuoto, ohituksia



Kuva 2.3 Kerrosarkkitehtuurin esitys ympyränmuotoisena, ei ohituksia

Kerrosarkkitehtuuritasoja voidaan kuvata kerrosvoileivällä (kuva 2.1). Kerrosvoileipämuotoisessa kuvauksessa ohitukset esitetään porrasmaisilla kerroksilla (kuva 2.2). Kerrosvoileivän päät taivuttamalla saadaan ympyrämuotoinen kuvaus (kuva 2.3). Kerroksilla on kahdenlaisia rajapintoja; kerroksen toteutuksen käyttöön ja toiset tarjottavaksi toisille kerroksille. Kerrosten välisten rajapintojen tulee täsmätä keskenään. Jos kerroksen rajapinnat on hyvin määritelty, voidaan koko kerros irrottaa ja korvata vastaavia palveluja toteuttavalla kerroksella. [Koskimies ja Mikkonen 2005, ss. 127-128]

Kerrosarkkitehtuurin ympyrämuotoinen esitystapa on havainnollinen. Ympyrän keskipisteessä on hyvin suojattuna tietokanta. Seuraavat sovelluskerrokset ja käyttöliittymä suojelevat ja suodattavat virheellisen tiedon ennen tietokantaan tallentamista. Käyttöliittymäkerroksen päälle voidaan vielä lisätä esimerkiksi ohjelmallinen palomuuri, erilaiset laite- ja tietoliikennesuojaukset. Ympyrän tai paremminkin pallon ulkopuolelle jää bittiavaruus.

Kuva 2.4 Liiketoimintajärjestelmä, esimerkki kerrosarkkitehtuurin käytöstä [Koskimies ja Mikkonen 2005, s. 128, kuva 6.3]

Kuvan 2.4 esimerkissä on sovelluksen käyttöliittymä ylimpänä kerroksena. Käyttöliittymän alapuolella on sovelluksen logiikkaa toteuttava kerros. Seuraavassa sovellusaluelogiikkakerros toteuttaa sovellusalueen logiikkaa ja käsitteitä. Alimmassa kerroksessa on tietokanta tietoineen. [Koskimies ja Mikkonen 2005, ss. 128-129]

Kerrosarkkitehtuuri voi olla moniulotteinen. Kerrostus voi suuntautua yhteen tai useampaan suuntaan. Kaksiulotteisessa kerrosarkkitehtuurissa dimensiot ovat tavallisesti "looginen" ja "yleisyys". "Loogisessa" kerrosjako perustuu perinteiseen, sovelluksen tai sovellusalueen logiikkaan tai toteutustekniikkaan. "Yleisyys" tarkoittaa komponenttien yleiskäyttöisyyden astetta. Kerrosarkkitehtuurin riippuvuudet kannattaa suunnitella siten, että on vähän kerrosten ohituksia ja väärään suuntaan meneviä riippuvuuksia. [Koskimies ja Mikkonen 2005, ss. 129-130]

Kerrosarkkitehtuuri soveltuu lähes kaikkiin järjestelmiin pienemmässä tai isommassa mittakaavassa. Järjestelmä voidaan jakaa karkeasti osiin, ja osat voidaan myös jakaa edelleen omiin kokonaisuuksiin. Kerrosarkkitehtuuri on helposti ymmärrettävässä muodossa, siksi sillä voidaan esitellä järjestelmää hyvin erilaisille ohjelmiston sidosryhmille. Kerrosarkkitehtuuri ohjaa suunnittelua riippuvuuksia vähentävään suuntaan, kerrokset riippuvat vain alemmasta kerroksesta. Järjestelmän rakenne on selkeä.

Kerrosarkkitehtuurin ongelmana on usein tehottomuus. Palvelua voidaan joutua hakemaan kerrosten läpi alemmista kerroksista. Välittävissä kerroksissa tehdään päällekkäistä työtä analysoimalla tietoa uudelleen. Palvelun sijoituspaikkaa voi olla joskus vaikea löytää, kun palvelu ei liity selvästi mihinkään kerrokseen. Poikkeusten käsittely on myös hankalaa kerrosarkkitehtuurissa. Ylemmistä kerroksista lähtevät palvelupyynnöt käyttävät pitkiä kutsuketjuja. Virhetilanteessa syntynyttä poikkeusta kuljetetaan kutsuketjua ylöspäin kunnes löydetään poikkeuksen käsittelijä. Ylemmän kerroksen poikkeusten käsittelijän voi olla vaikeaa korjata alemmassa kerroksessa syntynyttä poikkeusta. [Koskimies ja Mikkonen 2005, ss. 130-131]

3. SUUNNITTELUMALLIT

3.1 Yleisesittely

Suunnittelumallissa (design pattern) nimetään ja esitetään yleinen suunnitteluratkaisu oliopohjaisessa järjestelmässä usein esiintyvään suunnitteluongelmaan. Suunnittelumalliin on dokumentoitu ongelman kuvaus, ratkaisu, soveltamistilanteet ja ratkaisun seuraukset. Ongelman ratkaisuun on koottu tietyssä järjestyksessä olevia olioita ja luokkia yhdessä ratkaisemaan ongelman. [Gamma *et al*., 2001, s. 362] Suunnittelumallit ovat kehittyneet ja hioutuneet pitkän ajan kuluessa. Ne ohjaavat valitsemaan ratkaisuja, jotka tekevät ohjelmistosta uudelleenkäytettävän. Suunnittelumalleilla voidaan käyttää uudelleen hyviä vanhoja ratkaisuja ja arkkitehtuureja. Suunnittelutyö nopeutuu suunnittelumalleja käytettäessä ja myös dokumentointi paranee. [Gamma *et al*., 2001, s. 2]

Suunnittelumallien historia on lähtöisin Christopher Alexanderin ongelmapohjaisesta ratkaisuperiaatteesta rakennusten ja kaupunkien suunnittelussa. Hän on sanonut: "Jokainen ratkaisumalli kuvaa ongelman, joka toistuu jatkuvasti ympäristössämme ja määrittelee ongelmalle ratkaisuperiaatteen, jota voidaan soveltaa miljoonia kertoja aina uudella tavalla" [Alexander *et al*., 1977]. Samaa periaatetta on sovellettu oliopohjaisiin suunnittelumalleihin. Eric Gamma, Richard Helm, Ralph Johnson ja John Vlissides kokosivat suunnittelumalleja hakemistoksi. Tutkijat tunnetaan myös yhteisnimellä the Gang of Four (GoF). Hakemisto sai alkunsa osana Gamman väitöskirjaa 1990-luvun alussa ja täydentyi työryhmän käsittelyssä julkaistuun kirjaan asti. Hakemistossa esitetään yksinkertaisia ja ytimekkäitä ratkaisuja usein toistuviin suunnitteluongelmiin. [Gamma *et al*., 2001, s. 355]

Suunnittelumallit luokitellaan kolmeen ryhmään käyttötarkoituksen mukaan: luonti-, rakenne- ja käyttäytymismallit. Luontimalleja käytetään olioiden luontiprosessissa. Rakennemallit ovat luokkien ja olioiden koosteita. Käyttäytymismallit kuvaavat luokkien ja olioiden vuorovaikutusta ja vastuiden jakamista. Toinen suunnittelumallien jakoperuste on kohde, joka kertoo, onko kyseessä luokka vai olio. Luokkamallit kuvaavat luokkien ja niiden aliluokkien suhteita. Suhteet kiinnitetään periytymisen

kautta, ja ne ovat staattisia eli käännösaikaisia. Oliomallit käsittelevät olioiden välisiä suhteita, ja ne ovat dynaamisia eli ajonaikaisia. Luontimalleihin ja luokkakohteisiin kuuluva Tehdasmetodi esitellään tarkemmin seuraavassa kohdassa. [Gamma *et al*., 2001, ss. 2-10] Myöhemmin luvussa 8 sovelletaan käytännössä tilausjärjestelmän esimerkissä Tehdasmetodi-suunnittelumallia.

Luontimalleja käyttämällä voidaan rakentaa järjestelmiä, jotka ovat riippumattomia tavasta luoda, koostaa ja esittää oliot. Luontimallit, jotka periyttävät luokkia ja luovat niistä ilmentymiä, ovat luokkiin pohjautuvia luontimalleja. Olioihin perustuvat luontimallit siirtävät ilmentymän luomisen toiselle oliolle. Sovelluskehityksessä tarvitaan enemmän olioiden koostamista kuin luokkien periyttämistä. Luontimalleissa määritellystä joukosta perustoimintoja saadaan yhdistelemällä luotua uusia monimuotoisia toimintoja. Luontimalleissa on kaksi yleistä toimintoa. Ne piilottavat tiedon sovelluksen konkreettisien luokkien käytöstä. Lisäksi ne kätkevät luokkien ilmentymien luonnin ja yhdistämisen toisiinsa. [Gamma *et al*., 2001, s. 81]

3.2 Tehdasmetodi-suunnittelumalli

Tehdasmetodi kuuluu käyttötarkoituksensa mukaan luonti-luokkaan. Tehdasmetodin tarkoitus on määritellä olion luontioperaation kutsumuoto. Aliluokan tehtäväksi jätetään päätös, mistä luokasta ilmentymä luodaan. Ilmentymät luodaan aliluokassa.

Ohjelmistokehys määrittelee sovellusalueensa yleiset suunnitteluratkaisut ja kokonaisrakenteen. Kokonaisrakenteesta selviää luokkiin ja olioihin jako sekä näiden päävastuut, yhteistyötavat ja kontrollin kulku. Ohjelmistokehykset ovat isompia arkkitehtuuriosia kuin suunnittelumallit. Suunnittelumalleja voidaan käyttää lähes kaikissa sovelluksissa, koska ne ovat vähemmän erikoistuneita kuin ohjelmistokehykset. [Gamma *et al*., 2001, ss. 26-28] Tehdasmetodissa ohjelmistokehykset vastaavat usein olioiden luonnista. Ohjelmistokehyksissä käytetään abstrakteja luokkia olioiden ja olioiden välisten suhteiden määrittämiseen. Ohjelmistokehyksen ongelmana on luoda ilmentymiä tuntemistaan abstrakteista luokista, joista ei kuitenkaan voi luoda ilmentymiä. Tehdasmetodi tarjoaa ratkaisun juuri tähän ongelmaan. Tehdasmetodi-malli

kapseloi aliluokassa tiedon siitä, mikä aliluokka on luotava, ja siirtää sen ohjelmistokehyksen ulkopuolelle. [Gamma *et al*., 2001, s. 107]

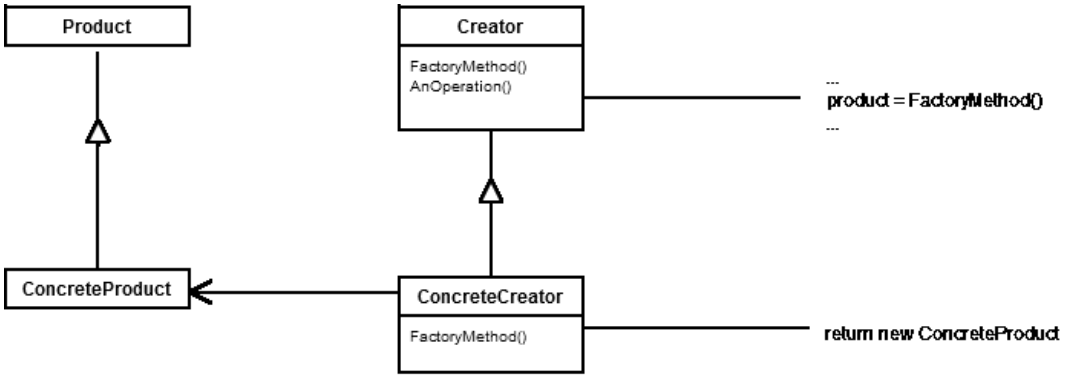
3.2.1 Soveltuvuus ja tunnettuja käyttökohteita

Tehdasmetodi-malli soveltuu käytettäväksi seuraavanlaisissa tapauksissa [Gamma *et al*., 2001, s. 108]:

1. Luotavan ilmentymän luokkaa ei tiedetä luokassa ennakolta.
2. Luokassa halutaan aliluokkien määrittävän luotavat oliot.
3. Luokat sijoittavat ja paikallistavat valtuuden yhdelle useasta avustavasta aliluokasta [Gamma *et al*., 2005, s. 108].

Tehdasmetodin tunnettuja käyttökohteita löytyy työkalupakeista ja ohjelmakehyksistä, esimerkiksi MacApp- ja ET++-toteutukset [Gamma *et al*., 2001, s. 115].

3.2.2 Rakenne ja osallistujat



Kuva 3.1 Tehdasmetodin rakennekaavio [Gamma *et al*., 2001, s. 108 mukailtuna]

Kuvassa 3.1 Product kuvaa Tehdasmetodin luomien olioiden rajapinnat. ConcreteProduct toteuttaa Product-rajapinnan. Creator määrittelee Tehdasmetodin, joka palauttaa Product-tyyppisen olion. Creator voi myös määritellä Tehdasmetodin oletustoteutuksen, joka palauttaa ConcreteProduct-olion. ConcreteCreator-oliossa tapahtuu Tehdasmetodin korvaus, jossa palautetaan ConcreteProduct-ilmentymä.

Tehdasmetodin yhteistyösuhteissa Creator antaa aliluokkiensa tehtäväksi toteuttaa oikean ConcreteProduct-ilmentymän palautuksen. [Gamma *et al*., 2001, ss. 108-109]

3.2.3 Seuraukset ja läheiset mallit

Tehdasmetodissa hyvänä ominaisuutena on välttyminen sovellusriippuvaisten luokkien käyttämiseltä koodissa. Tehdasmetodin haittana on, että Creator-luokasta joudutaan periyttämään jokaista ConcreteProduct-luokkaa kohden uusi aliluokka. Kaksi muuta Tehdasmetodiin liittyvää seurausta ovat seuraavat:

1. Tehdasmetodi tarjoaa aliluokalle koukun (hook). Olioiden suoraan luomista joustavampi tapa on luoda Tehdasmetodia käyttäen oliot luokan sisällä. Tehdasmetodin aliluokan koukkuun aliluokat voivat ripustaa oman laajennetun version oliosta.

2. Tehdasmetodi yhdistää rinnakkaiset luokkahierarkiat. Osavastuun delegointi toiselle erilliselle luokalle synnyttää rinnakkaisen luokkahierarkian. Tehdasmetodi voi määritellä kahden luokkahierarkian yhteyden. Yhteen kuuluvat luokat paikallistetaan metodin avulla. [Gamma *et al*., 2001, ss. 109-110]

Tehdasmetodia käytetään usein Abstrakti Tehdas -suunnittelumallin toteutuksessa. Toinen läheinen suunnittelumalli on Operaatiorunko, josta kutsutaan Tehdasmetodia. Prototyyppi-malli on myös hyvin lähellä Tehdasmetodia. Prototyypissä ei luoda Creator-aliluokkaa. Product-luokassa käytetään usein Initialize-operaatiota alustamaan olio. Tehdasmetodi ei käytä Initialize-operaatiota. [Gamma *et al*., 2001, s. 116]

3.2.4 Toteutus

Tehdasmetodin toteutuksessa on hyvä huomioida seuraavia asioita [Gamma *et al*., 2001, ss. 110-113]:

1. Tehdasmetodin kaksi päävariaatiota ovat seuraavat: a) Creator-luokka ei tarjoa Tehdasmetodin oletustoteutusta eli on abstrakti ja b) Creator-luokka tarjoaa Tehdasmetodin oletustoteutuksen eli on konkreettinen. Oletustoteutuksen määrittelevä abstrakti luokka on mahdollista tehdä, mutta sen käyttö on harvinaista. A-vaihtoehdossa toteutus määritellään aliluokissa. B-vaihtoehdossa luodaan oliot erillisellä operaatiolla, jotta luontitapa voidaan korvata aliluokissa.

2. Parametrisoidussa Tehdasmetodissa pystytään luomaan monentyyppisiä tuotteita. Tällöin Tehdasmetodille pitää antaa parametrina luotavan olion tyyppi. Product-rajapinta on sama kaikilla Tehdasmetodin luomilla olioilla.

3. Kieliriippuvaisia variaatioita ja näkökulmia tarvitaan kielien erilaisien syntaksien vuoksi. Eroja esiintyy esimerkiksi olion luonnin yhteydessä. *Laiskan alustuksen* (lazy initialization) tekniikalla vältetään kieliriippuvuutta. Tekniikassa konstruktori alustaa tuotteen arvon nollaksi, eikä luo konkreetista tuotetta. Luonnin yhteydessä tuotteen arvo tutkitaan ja päätellään tuotteen olemassaolo. Jos tuotetta ei ole, luodaan se.

4. Geneeristen luokkien käyttö perimisen sijasta vähentää turhien aliluokkien määrittelyä. Määritellään Creator-aliluokalle geneerinen aliluokka, jolle annetaan parametrin avulla Product-luokka.

5. Nimeämiskäytännössä kannattaa käyttää tapaa, josta selviää helposti Tehdasmetodin käyttö, esimerkiksi luokkien nimissä.

4. RELAATIOTIETOKANNAN SUUNNITTELU

Tämä luku käsittelee relaatiotietokannan suunnittelua. Suunnittelussa tähdätään tulevaisuuteen, pyritään varautumaan tietokannan kasvuun sekä muutostarpeisiin. Tietokannan suunnittelussa voidaan myös jo aluksi ajatella lopputulosta: Minkälaisia tietoja halutaan tietokannasta ulos? Mitä alkutietoja tarvitaan tietokantaan, että lopputulokseen päästään? Tietokantaan yritetään mallintaa reaalimaailman pienempi tai isompi toimintokokonaisuus tietoryhminä. Tietoryhmät (taulut) yhdistämällä saadaan aikaiseksi tietovarasto eli tietokanta. Toimintoja ryhmitellään pienempiin kokonaisuuksiin ja etsitään keskeisiä tietoja tai käsitteitä. Käsiteanalyysissa tietoja punnitaan ja arvioidaan mitkä ovat tarpeellisia ja mukaan otettavia käsitteitä. Käsiteanalyysissa muodostetaan käsitekaavio. Käsitekaaviota käytetään apuna taulujen suunnittelussa. Päällekkäiset tiedot karsitaan eli tehdään normalisointi.

4.1 Relaatiotietokanta

Codd [1970] kehitti *relaatiomallin* (relational model) ja käsittelyteorian vuonna 1970. Hän työskenteli ja johti ensimmäisten relaatiotietokantojen testiversioiden kehitystyötä IBM:n tutkimuslaboratoriossa. DB2, IBM:n kaupallinen relaatiotietokantatuote, julkaistiin vuonna 1983. [IBM, 2003]

Relaatiomalli määrittelee, miten tietoa organisoidaan ja käsitellään tietokannassa, ja se perustuu joukko-oppiin ja predikaattilogiikkaan. *Relaatiotietokanta* (relational database) on relaatiomallia noudattava, tietokannan hallintajärjestelmän avulla muodostettu tietokanta. Relaatiotietokannassa *taulu* (table) vastaa relaatiomallin relaatiota. Taulu koostuu kiinnostuksen kohteena olevien objektien ominaisuuksien arvoja sisältävistä *riveistä* (row) ja *sarakkeista* (column). Rivin relaatiomallivastine on monikko. Tietue on tietojen tallennukseen liittyvä termi, rivit tallennetaan tietueina. Sarake on samaan arvoalueen määrittelyyn liittyvien arvojen lista taulun eri riveillä. Tauluun ei tule tuplarivejä, kun käytetään *avaimena* (primary key) saraketta tai sarakejoukkoa yksilöimään taulun rivit. Coddin relaatiomallin avaineheyssäännön mukaan taulun avaimen arvo on pakollinen. *Viiteavain* (foreign key) on sarake tai sarakeyhdistelmä,

joka viittaa taulusta toisen taulun avaimeen tai tauluun itseensä. [Hovi *et al.,* 2005, ss. 343-347]

Yleisin relaatiokannan käyttökieli on *SQL* (Structured Query Language), joka on alunperin IBM:n kehittämä tietokannan kyselykieli [ATK-sanakirja, 1997, s. 171]. Sarakkeiden tietotyyppien muodot vaihtelevat hieman erilaisissa SQL-tietokannoissa. Seuraavassa on avattu muutamia standardimuotoisia tietotyyppejä [Elmasri and Navathe, 2007, ss. 238-239]:

- INTEGER/INT/SMALLINT - kokonaisluku
- REAL/FLOAT - liukuluku
- DECIMAL(L, S) - desimaaliluku, jossa on maksimissaan L numeroa, joista desimaaliosassa on S numeroa.
- CHAR(N) - vakiomittainen merkkijono, jonka pituus on (aina) täsmälleen N merkkiä.
- VARCHAR(N) - vaihtelevamittainen merkkijono, jonka pituus on korkeintaan N merkkiä.
- DATE - päivämäärä; esitetään tekstinä formaatissa 'YYYY-MM-DD'.
- TIME - kellonaika; esitetään tekstinä formaatissa 'HH:MM:SS'. Tunnit, minuutit ja sekunnit annetaan kahdella numerolla. Myös aikavyöhykettä voidaan käyttää, A TIME WITH TIME ZONE.
- BOOLEAN - totuusarvomuuttuja TRUE/FALSE (tosi/epätosi).

4.2 Taulujen suunnittelu suunnittelulomakkeilla

Taulujen suunnittelussa voidaan hyödyntää suunnittelulomakkeita. Lomake auttaa keräämään taulun tiedot. Kuvassa 4.1 on esimerkki tuotteisiin liittyvään tauluun täytetystä suunnittelulomakkeesta. Lomakkeen yläosassa on yleistietoa Tuote-taulusta ja sen ympäristöstä. Liittymät ja suhde muihin tauluihin on kerrottu lomakkeella.

Tuotetietojen yksityiskohdissa on suunniteltu sarakkeen nimi, lyhyt kuvaus, tietotyyppi, pakollisuus ja alkuarvo.

Järjestelmä	**Tilausjärjestelmä**			
Taulun nimi	Tuote			
Kuvaus	Tietoja tuotteesta			
Liittymät tauluihin	tilausrivi, suhde monta			
Nimi	**Kuvaus**	**Tietotyyppi**	**Pakollisuus**	**Alustus**
tuotenro	Tuotteen id numero	INTEGER(10)	kyllä	
tuotenimi	Tuotteen kuvaus	VARCHAR(50)	kyllä	
ahinta	Yksikköhinta	DECIMAL(15,5)	kyllä	
yksikko	Yksikkö	CHAR(5)	ei	kpl

Kuva 4.1 Tuotelomake

Järjestelmän kaikista tauluista täytetään vastaavanlaiset lomakkeet. Jokainen lomake on tauluehdokas luotavaan tietokantaan.

4.3 Taulujen suunnittelu käsiteanalyysin avulla

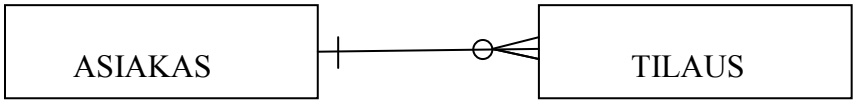
Taulujen suunnittelussa voidaan hyödyntää myös käsiteanalyysia. Sen avulla mallinnetaan kohdealuetta ja muodostetaan käsitekaavio. Käsitekaavio kuvaa tietokantaan tallennettavia tietoja ja niiden välisiä suhteita. [Hovi *et al.,* 2005, s. 339] Käsitekaavio on tuote- ja tietokantariippumaton. Chenin [1976] luoma *ER-mallinnusmenetelmä* (entity relationship modelling) on yksi varhaisimmista käsitteellisen mallintamisen menetelmistä.

4.3.1 Entiteetti, ominaisuus ja yhteys

Entiteetti (entity) on selvästi yksilöitävä 'ajatus', esimerkiksi henkilö, yritys tai tapahtuma. *Ominaisuudet* (attribute) kuvaavat entiteettejä; avaintieto yksilöi entiteetin esiintymät. [Chen, 1976]

Entiteettien välillä on yhteyksiä eli suhteita (relationship) [Chen, 1976]. Entiteettejä ja yhteyksiä voidaan löytää esimerkiksi tekstianalyysin avulla. Lauseen substantiivi on

entiteetti. Lauseen verbit kuvaavat yhteyksiä. Seuraavassa esimerkissä asiakas ja tilaus ovat entiteettejä. Tekee-verbi kuvaa entiteettien välistä yhteyttä, "ASIAKAS tekee TILAUKSEN". Tiedon eli attribuutin esimerkkinä voisi olla asiakkaan nimi.



Kuva 4.2 Esimerkki kahden käsitteen, asiakas ja tilaus välisestä yksi-moneen-yhteydestä [Hovi *et al.,* 2005, s. 38, kuva 3.8 mukailtuna]

Kuvan 4.2 esimerkin asiakkaalla voi olla monta tilausta, mutta yhdellä tilauksella sallitaan vain yksi asiakas eli oletuksena on, etteivät asiakkaiden yhteistilaukset ole sallittuja.

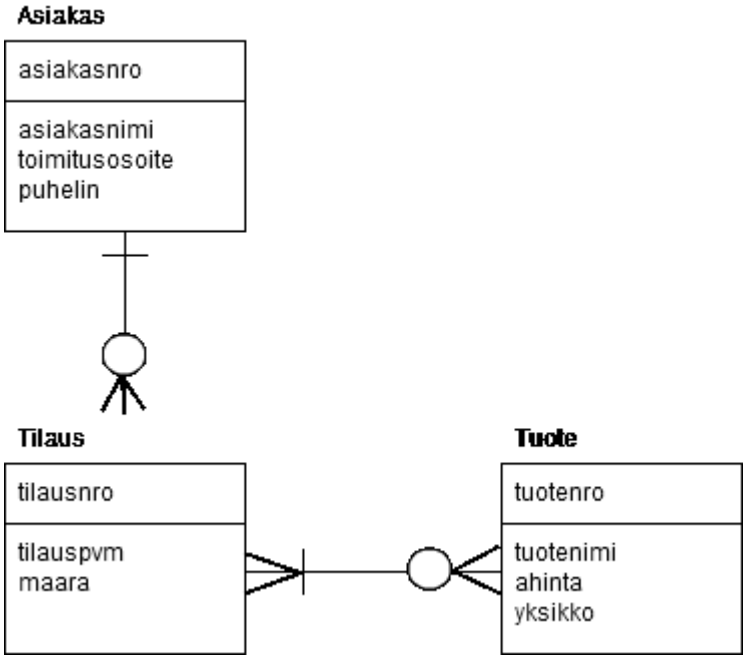
Entiteettien välisiä lukumääräsuhteita voi olla kolmenlaisia: yksi-yhteen, yksi-moneen ja moni-moneen. Yksi-yhteen -yhteys, jossa lapsientiteetillä on yksi isäentiteetti, on harvinainen. Maa ja sen pääkaupunki voisivat olla tästä esimerkkinä. Maalla on yksi pääkaupunki ja yksi pääkaupunki on yhtä maata kohti. Maa on isä, ja pääkaupunki on lapsi. Kaikkein yleisin on yksi-moneen -yhteys, jota kutsutaan myös isä-lapsi -yhteydeksi eli isällä on monta lasta ja lapsella on yksi isä. Kuvan 4.2 esimerkissä asiakas on isä, ja tilaukset ovat lapsia. Esimerkki moni-moneen -yhteydestä voisi olla asiakas ja pankkitili eli monella lapsella on monta isää. Pankin asiakkaalla voi olla monta pankkitiliä, ja samaa pankkitiliä voi käyttää monta asiakasta. Asiakas on isä, ja pankkitili on lapsi. [Hovi *et al.,* 2005, ss. 36-39]

4.3.2 Käsitekaavio

Käsitekaavio syntyy käsiteanalyysin tuloksena, ja se kuvaa kohdealuetta. Tietosisällöltään käsitekaavio on tuote- ja tietokantariippumaton loogisen tietosisällön kuvaus. Käsitekaaviossa on käsitteitä, niihin liittyviä ja niitä kuvaavia tietoja ja käsitteiden välisiä yhteyksiä. Käsitekaaviota voidaan käyttää tietokannan sisällön suunnittelun lisäksi kertomaan, mitä tietoja tietokannassa on.

Kuvan 4.3 esimerkissä entiteettejä ovat asiakas, tilaus ja tuote. Entiteettien välisiä yhteyksiä kaaviossa on kuvattu seuraavasti [Hovi *et al.,* 2005, s. 325]:

- Yhdellä asiakkaalla voi olla monta tilausta.
- Yhdessä tilauksessa voi olla vain yksi asiakas.
- Yhdellä tilauksella voi olla monta tuotetta.
- Yksi tuote voi olla monessa tilauksessa.



Kuva 4.3 Tilausjärjestelmän karkea käsitekaavio [Hovi *et al.,* 2005, s. 325 mukailtuna]

Käsitekaavio muunnetaan relaatiotietokannan tauluiksi seuraavanlaisilla muunnoksilla. Relaatiotietokannassa käsitettä tai entiteettiä vastaa termi taulu ja olioanalyysissä termi luokka. [Hovi *et al.,* 2005, s. 339] Käsitekaavion ominaisuuksista (attribuuteista) tulee relaatiotietokannassa taulujen sarakkeet. Entiteettien esiintymät ovat relaatiotietokannan rivejä. Relaatiotietokannan viiteavaimet ovat käsitekaavion entiteettien välisiä suhteita. Entiteetin esiintymän yksilöivät avaintiedot muodostavat relaatiotietokannassa perusavaimen. [Hovi *et al.,* 2005, ss. 6-7]

4.4 Normalisointi

Relaatiotietokantojen normalisoinnilla pyritään parantamaan tietokannan taulujen rakennetta [Codd, 1970]. Kukin tieto on tallennettava vain kertaalleen, mikä on muutosjoustavaa, mahdollistaa tehokkaat päivitykset ja estää päivitysanomalioita. Normalisointiteoriassa määritellään useita eri normaalimuotoja. Ensimmäinen, toinen ja kolmas normaalimuoto ovat yleisimmin käytetyt ja alkuperäiset normaalimuodot. Lisäksi on Boyce-Codd normaalimuodot, neljäs ja viides normaalimuoto. [Hovi *et al.,* 2005, s. 86]

Ensimmäisessä normaalimuodossa poistetaan relaatiotietokannan tauluista toistuvat ryhmät ja moniarvoiset sarakkeet. Taulun jokaisen attribuutin arvojoukko koostuu vain atomisista (so. jakamattomista) arvoista. Normalisointi toteutetaan jakamalla taulun tiedot kahteen tai useampaan osaan. [Hovi *et al.,* 2005, s. 88] Atominen arvo tarkoittaa, että normalisoitu taulu ei voi sisältää koottuja eikä moniarvoisia attribuutteja [Elmasri and Navathe, 2007, ss. 348-349].

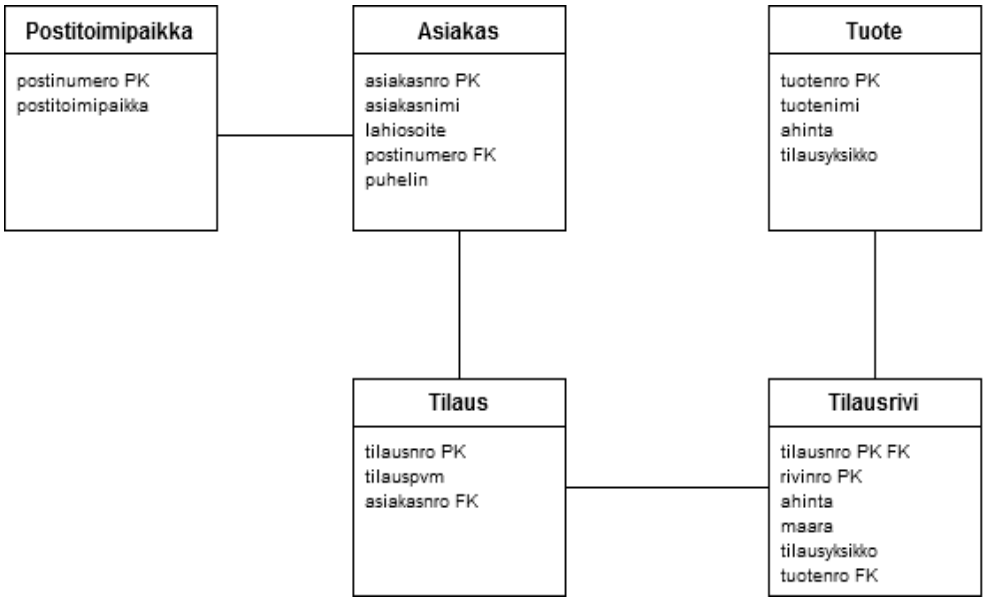
Toisen normaalimuodon säännön mukaan, jos taulussa on moniosainen perusavain, niin kaikkien sarakkeiden tulee olla funktionaalisesti riippuvia koko perusavaimesta (eikä siis vain osa-avaimesta). Funktionaalinen riippuvuus tarkoittaa, että attribuutin B arvo on yksikäsitteisesti selvillä, kun tiedetään attribuutin A arvo (f:x(A) $\rightarrow$x(B) eli relaatiokaavion R ilmentymissä jokainen A:n arvo kuvautuu yhdelle B:n arvolle). [Elmasri and Navathe, 2007, s. 352]

Kolmannen normaalimuodon sääntö on seuraavanlainen: taulun jokaisen ei-avainsarakkeen pitää olla funktionaalisesti riippuvainen ainoastaan koko perusavaimesta. Taulun on oltava toisessa normaalimuodossa. [Hovi *et al.,* 2005, s. 93] Taulussa ei saa olla osittaisia eikä transitiivisia riippuvuuksia. Transitiivinen riippuvuus määritellään kahden perusavaimeen kuulumattoman kentän välisenä funktionaalisena riippuvuutena. [Elmasri and Navathe, 2007, s. 354]

Kolme ensimmäistä normalisointimuotoa ovat yleisemmin käytettyjä ja niiden tuoma normalisoinnin taso on riittävä tavallisimmille rakenteille [Hovi *et al.,* 2005, s. 94]. Edellisiä vaativampia normaalimuotoja ns. Boyce-Codd, neljättä ja viidettä

normaalimuotoa käytetään harvemmin. Neljäs normaalimuoto käsittelee moniarvoisia riippuvuuksia, ja viides normaalimuoto keskittyy liitosriippuvuuksiin. [Elmasri and Navathe, 2007, ss. 386-397]

Tilausjärjestelmän ER-kaaviona piirretystä karkeasta käsitekaaviosta (kuva 4.3) on tehty normalisoitu tilausjärjestelmän tietokantakaavio (kuva 4.4). ER-kaavion asiakas-, tuote- ja tilaus-entiteettityypit on muutettu tauluksi. Taulujen väliin on piirretty taulujen välisiä viittauksia kuvaavat viivat. Tietokantakaaviossa on esitetty myös sarakkeet.

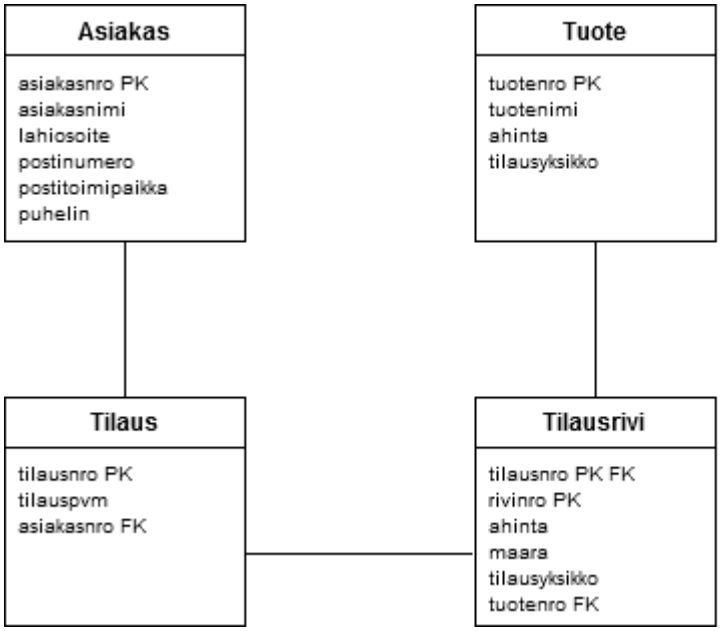


Kuva 4.4 Normalisoitu tilausjärjestelmän tietokantakaavio

Kuvan 4.4 tietokantakaavio on normalisoitu kolmanteen normaalimuotoon. Postitoimipaikka on erotettu omaksi tauluksi. Tilauksen ja tuotteen välinen monen suhde moneen on purettu uudeksi tilausrivi-tauluksi. [Hovi *et al.,* 2005, s. 325 mukailtuna] Postinumero, asiakasnro, tuotenro, tilausnro ja rivinumero on pääavaimen (PK, primary key) nimiä. Viiteavaimet on merkitty FK (foreign key)-lyhenteellä.

Denormalisointi on peruuttamista viimeisestä normaalimuodosta. Puretaan tietojen toiston minimointia ja sitä, että tiedot olisivat vain kerran tallessa. Denormalisoinnin hyöty tulee esiin tietojen käytön yhteydessä. Tietojen lukeminen nopeutuu, kun taulujen välisiä liitoksia ja levyn lukukertoja on vähemmän. [Hovi *et al.,* 2005, ss. 95-96]

Denormalisointi esimerkkikuvan 4.4 kaaviossa voisi olla postitoimipaikka-taulun poistaminen. Postitoimipaikan käsittelyn palauttaminen asiakas-taulun tiedoksi nopeuttaa käsittelyä. Kuvan 4.5 esimerkissä on toteutettu denormalisointi eli postitoimipaikka-taulu on poistettu kaaviosta. Postitoimipaikan tiedot löytyvät asiakas-taulun attribuutteina.



Kuva 4.5 Tilausjärjestelmän denormalisoitu tietokantakaavio

5. UML

5.1 Johdanto

UML (Unified Modeling Language) on mallinnuskieli. UML-mallinnuskieltä voidaan käyttää sovellusten arkkitehtuurin, käyttäytymisen ja rakenteen mallintamiseen, lisäksi se soveltuu liiketoimintaprosessien ja tiedon rakenteen kuvaamiseen. [OMG, 2013]

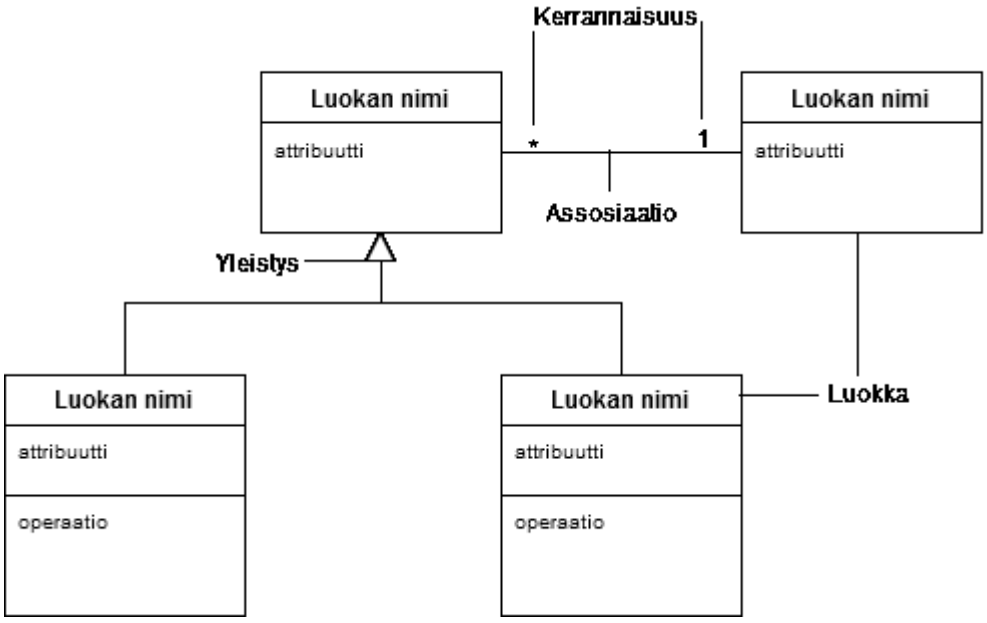
OMG:n UML 2.0 -version esittelyssä mainitaan UML-mallinnuskielen sisältävän 13 erilaista kaaviota. Edelleen kaaviot jakautuvat kolmeen kaaviotyyppiin: kuusi rakennetta (structure diagrams), kolme käyttäytymistä (behavior diagrams) ja neljä vuorovaikutusta (interaction diagrams) kuvaavaa kaaviota. Rakennekaavioita ovat komponentti- (component diagram), kooste- (composite structure diagram), luokka- (class diagram), olio- (object diagram), pakkaus- (package diagram) ja sijoittelukaavio (deployment diagram). Vuorovaikutuskaavioita ovat ajoitus- (timing diagram), kokoava vuorovaikutus- (interaction overview diagram), kommunikointi- (communication diagram) ja sekvenssikaavio, joka tunnetaan myös nimellä viestiyhteyskaavio (sequence diagram). Kolme käyttäytymiskaaviota ovat aktiviteetti- (activitity diagram), käyttötapaus- (use case diagram) ja tilakaavio (state (machine) diagram). [OMG, 2013]

Ensimmäinen standardi syntyi vuonna 1997 Object Management Groupin (OMG) toimesta. [Fowler and Scott, 2002, ss. 2-5] Nykyisin on käytössä UML 2.0 -versio ja UML 2.5 -versio on kehitystyön alla. Yleisen ja standardoidun mallinnuskielen luominen ja yhteisistä pelisäännöistä sopiminen on tärkeää, että eri suunnittelijat ymmärtäisivät suunnitelmien sisällön samalla tavalla.

UML-mallinnuskielestä esitellään keskeisimpiä kaavioiden piirtämisessä tarvittavia peruskäsitteitä rakennekaavioihin kuuluvan luokkakaavion avulla. Vuorovaikutuskaavioista esitellään tarkemmin viestiyhteyskaavio. UML-mallinnuskieltä tullaan käyttämään sekä tietokantasuunnitelman mallintamiseen että suunnittelumallin kuvaamiseen.

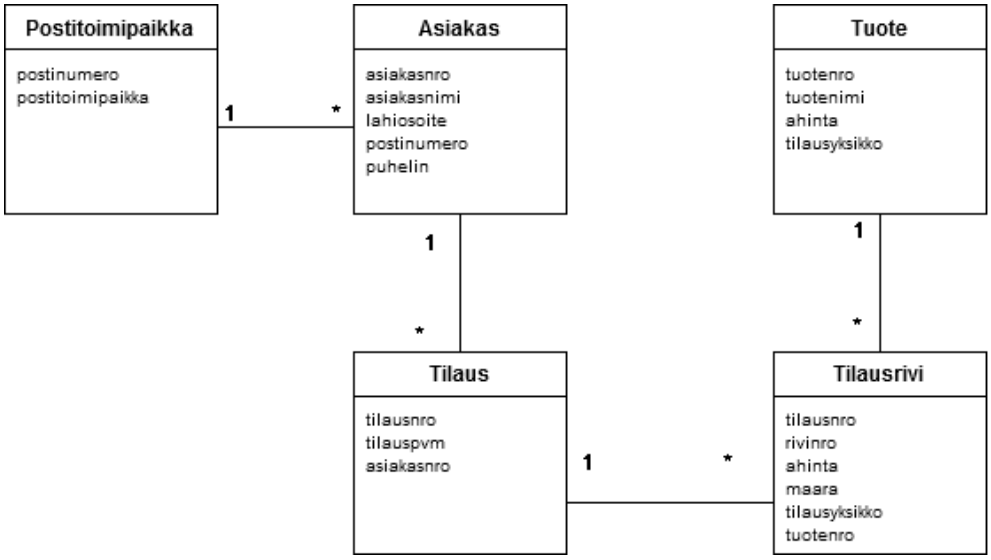
5.2 Peruskäsitteet ja luokkakaavio

Luokkakaavio on yleisesti käytetty oliokeskeisessä suunnittelussa. Fowler and Scott [2002, ss. 45-46] jakavat oliotyyppien väliset staattiset suhteet kahteen päätyyppiin: assosiaatiot (associations) ja alityypit (subtypes). Luokkakaavion muita peruskäsitteitä ovat luokan attribuutit (attribute) ja operaatiot sekä rajoitukset (constraints). Rajoituksia on olioiden välisissä kytkennöissä. Kuvaan 5.1 on kerätty esimerkkejä luokkakaavion erilaisia peruskäsitteitä.



Kuva 5.1: Luokkakaavio [Grässle *et al.,* 2005, s. 212 mukailtuna]

Kuvassa 5.1 käytettyjen luokkakaavion peruskäsitteiden määritelmiä on koottu alla olevaan luetteloon [Fowler and Scott, 2002, ss. 48-55]: *Luokka* on joukko, jonka alkioilla on jokin yhdistävä ominaisuus. Oliotekniikassa luokka on malli, jonka ilmentymät ovat samankaltaisia olioita. [ATK-sanakirja, 1997, s. 98] *Operaatiot* ovat prosesseja, jotka luokka toteuttaa. *Yleistys* tarkoittaa ohjelmointikielessä periytymistä. *Assosiaatio* edustaa luokkien ilmentymien välisiä suhteita. *Attribuutti* voi olla sekä ominaisuus tai määre, että relaatiotietokannassa taulun (relaation) sarakkeen nimi. *Kerrannaisuus* ilmaisee assosiaatioon osallistuvien olioiden määrän ala- ja ylärajan. Tavallisimmat kerrannaisuusarvot ovat 1 (vain yksi), * (ääretön) ja 0..1 (ei yhtään tai yksi).



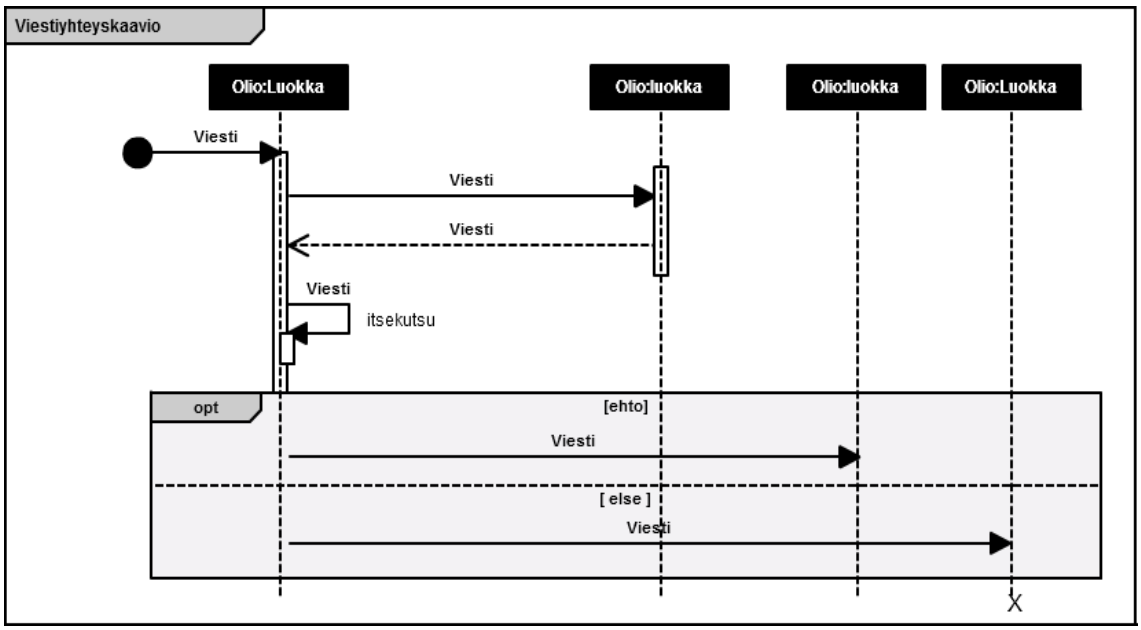
Kuva 5.2 Tilausjärjestelmän luokkakaavio

Kuvan 5.2 esimerkissä luokkia ovat Postitoimipaikka, Asiakas, Tilaus, Tilausrivi ja Tuote. Postitoimipaikka-luokan attribuutteja ovat postinumero ja postitoimipaikka. Tilaus- ja Tilausrivi-luokan välillä on assosiaatio. Kerrannaisuusarvoina ovat 1 (vain yksi) ja * (ääretön) eli esimerkiksi yhdellä tilauksella voi olla monta tilausriviä.

5.3 Viestiyhteyskaavio

Viestiyhteyskaavio kuuluu vuorovaikutuskaavioiden ryhmään. Vuorovaikutuskaaviot mallintavat olioryhmien yhteistyötä. Viestiyhteyskaavion vahvuus on se, että nähdään tapahtuvien asioiden järjestys. [Fowler and Scott, 2002, ss. 59-66]

Viestiyhteyskaaviota käytetään tässä tutkimuksessa suunnitelmien mallintamiseen. Kuvassa 5.3. on esimerkillä esitelty viestiyhteyskaavion peruskäsitteitä.



Kuva 5.3 Viestiyhteyskaavio

Kuvassa käytettyjen viestiyhteyskaavion peruskäsitteiden määritelmiä on koottu alla olevaan luetteloon [Fowler and Scott, 2002, ss. 60-64]: *Elämänviiva* (lifeline) edustaa olion elämää vuorovaikutuksen aikana (oliosta lähtevä pystysuora katkoviiva). V*iesti* (message / query event) tarkoittaa olioiden välistä vuorovaikutusta (elämänviivojen välinen nuoli). Viestien tapahtumajärjestys on ylhäältä alas. *Paluuviestiä* (return) merkitään katkonuolilla. Paluunuolet jätetään usein pois kaavion pitämiseksi selkeänä. *Itsekutsu* (self-call) on viesti, jonka olio lähettää itselleen (viestinuoli, joka palaa takaisin samaan elämänviivaan). *Tuhoaminen* (deletion) on olion poistaminen itse tai viestillä (merkitään suurella X:llä). Ohjaustiedoissa on kaksi tärkeää käsitettä: ehto ja iterointimerkki. Hakulauseissa oleva *ehto* (condition) ilmaisee, milloin viesti lähetetään, esimerkiksi [lisätilaustarve]. *Iterointimerkillä* lisätään viestiin tieto, mikä kertoo lähetyskerrat usealle oliolle. Iterointijoukon voi kirjoittaa hakasulkeisiin, esimerkiksi * [kaikille tilausriveille].

6. ARKKITEHTUURIN ARVIOINTIMENETELMÄ

Tunnettu arkkitehtuurin arviointimenetelmä ATAM (architecture tradeoff and analysis method) esitellään tässä luvussa. Arviointimenetelmää tullaan käyttämään myöhemmin arkkitehtuurin ja suunnittelumallien vertailussa. Software Engineering Institute Carnegie Mellon University on kehittänyt ATAM-arviointimenetelmän [ATAM, 2014].

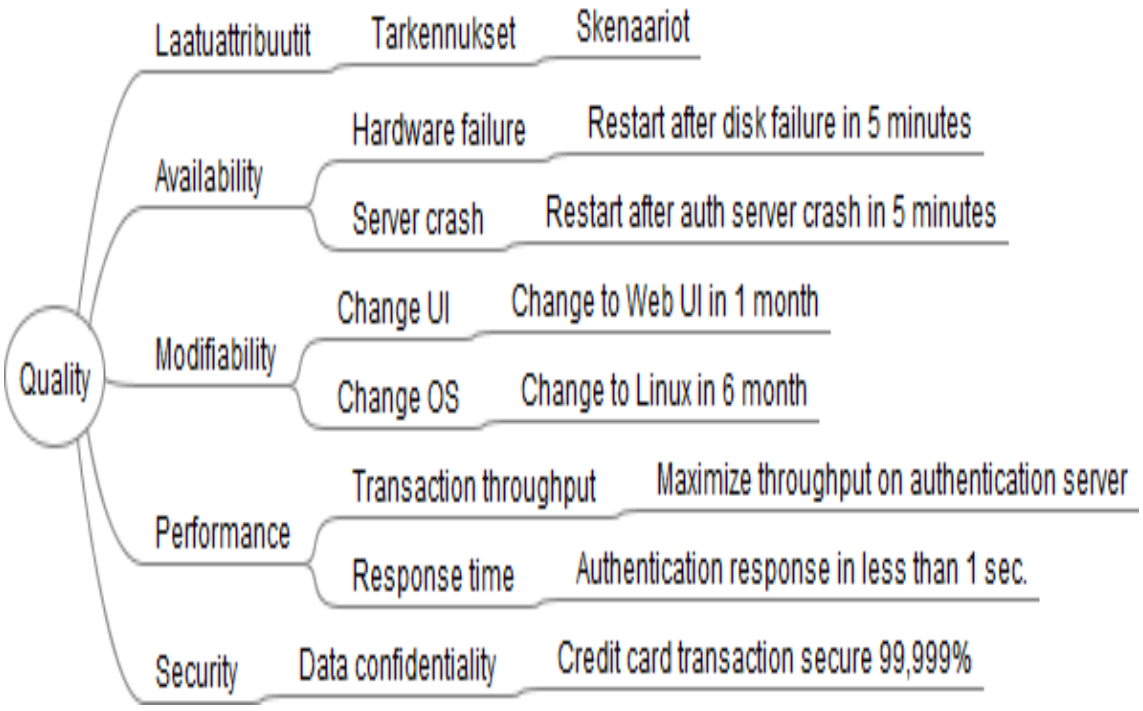
ATAM-menetelmällä voidaan tutkia ennen järjestelmän toteutusta seuraavia asioita [Clements *et al*., 2002, s. 323]: arvioida arkkitehtuurin sopimista suunnitellulle järjestelmälle, etsiä parhaiten soveltuva arkkitehtuuri valittavista vaihtoehdoista ja arvioida järjestelmään tulevia ominaisuuksia laadun kannalta. ATAMissa käytetään skenaariotekniikkaa, ja arvioinnissa etsitään konkreettisia esimerkkitilanteita. Esimerkiksi tarkasteltaessa järjestelmän turvallisuutta skenaariossa käsitellään jotakin uhkatilannetta järjestelmän käytössä. Skenaarioiden etuna on helppo löydettävyys, ymmärrettävyys ja konkreettisuus. [Haikala ja Mikkonen, 2011, s. 201]

ATAM koostuu neljästä osa-alueesta: esittely, analyysi, testaus ja viimeisenä raportointi. Ensimmäisessä vaiheessa ovat esittely- ja analyysiosat ja toisessa vaiheessa ovat testaus- ja raportointiosat. Ensimmäiseen vaiheeseen osallistuvat arviointiryhmän lisäksi tuotantoprojektin vastuuhenkilöt. Toisessa vaiheessa lisätään muita sidosryhmiä, esimerkiksi asiakkaan edustajia. Arviointiprosessi kestää noin kolme päivää. Tulosten kertaamiseen uusille osanottajille käytetään päivä. [Haikala ja Mikkonen, 2011, ss. 201-202]

Esittelyosion tavoitteena on, että kaikki ymmärtäisivät järjestelmän toimintaympäristön. ATAM-arviointimenetelmästä kerrotaan osallistujille, ja arkkitehtuurin tekninen ympäristö ja rajapinnat muihin järjestelmiin esitellään. Myöskin järjestelmän vaatimukset ja tavoitteet liiketoiminnalle sekä järjestelmärajoitteet tuodaan esille.

Analyysiosiossa tunnistetaan olennaiset arkkitehtuuriratkaisut, joihin on liitetty laatuvaatimuksia. Analysoidaan järjestelmän laatuvaatimukset, kuvataan skenaariot ja etsitään arkkitehtuurin kriittiset kohdat. Tämän jälkeen laaditaan laatupuu (utility tree) (kuva 6.1). Siinä täsmennetään laatuominaisuuksia ja esitetään skenaariota eli laatuominaisuuksien esimerkkejä. Skenaarioita painotetaan kahdella ominaisuudella:

skenaarion tärkeys järjestelmän suhteen ja skenaarion toteuttamisen vaikeus järjestelmään. Seuraavaksi laatupuu ja arkkitehtuuriratkaisut liitetään yhteen. Laatupuun perusteella tunnistetaan riskit, turvalliset ratkaisut, herkkyyskohdat ja tasapainokohdat.



Kuva 6.1 Laatupuu [Haikala ja Mikkonen, 2011, s. 203 mukailtuna]

Testiosiossa sidosryhmät täydentävät ja testaavat analyysin tuloksia käyttäen skenaarioita apuna. Sidosryhmät tekevät skenaarioita omien kiinnostusten pohjalta. Samalla sidosryhmien välillä syntyy keskustelua ja saavutetaan yhteinen näkemys järjestelmän tärkeistä laatuominaisuuksista. Skenaarioita verrataan laatupuussa jo oleviin skenaarioihin. Jos skenaariota ei löydy puusta, se lisätään uudeksi lehdeksi laatupuuhun.

Raporttiosiossa ATAM-prosessin tulokset esitellään arviointiraporttina koko ryhmälle. Keskeinen ATAM-menetelmän tavoite on löytää arkkitehtuurin laatuominaisuuksiin vaikuttavat ratkaisut skenaarioiden avulla ja myös analysoida ratkaisut. Arvioinnilla saadaan tietoa arkkitehtuurin laatuominaisuuksista, jotka auttavat ymmärtämään arkkitehtuuria ja hallitsemaan sen riskejä. Arkkitehtuurin korjaaminen ei kuulu ATAMin tehtäviin, mutta joskus tuloksena on arkkitehtuurin parannusehdotuksia.

7. TUTKIMUSTEHTÄVÄT JA -METODI

Tässä luvussa kuvataan tutkimuksen aihetta, tutkimustehtäviä ja -metodia. Aluksi pohditaan tutkimusmetodia.

Tutkimusotteeni on kontekstualismin meta level analysis across cases. Teoria ja käytäntö yhdistyvät luonnollisella tavalla kontekstualismissa. Kontekstualismi on kiinnostunut tutkimaan tapauksia (event) omassa ympäristössään. Tulokset saadaan esiin kvalitatiivisella vahvistamisella. Jos konteksti muuttuu, tieto muuttuu myös. Meta level analysis across cases -tutkimuksessa tutkimustuloksissa on teoreettisia yleistyksiä ja vertailuja. [Järvinen ja Järvinen, 1994, ss. 55-58] Lisäksi tutkimuksessa on käytetty apuna prosessin havainnointia.

Edellisissä luvuissa on esitelty taustakirjallisuuden perusteella toteutusympäristö ja työvälineet: UML-mallinnuskieli, relaatiotietokannan suunnittelu, yleisesti ohjelmistoarkkitehtuurit, ATAM-arviointimenetelmä ja olio-ohjelmoinnin suunnittelumalleista Tehdasmetodi. Työvälineitä käytetään tutkimustehtävän esittämiseen. Tarkastelen tutkielmassani relaatiotietokannan päivityksen avulla ohjelmoinnin suunnittelumallin soveltamista kerrosarkkitehtuuriympäristöön ja ohjelmistotuotantoon.

Ensimmäisenä tutkimustehtävänä on esittää yksinkertaisilla pienillä UML-kaavioilla relaatiotietokannan päivitys suunnittelumallia hyödyntäen ja ilman suunnittelumallia. Suunnittelumalliksi on valittu Tehdasmetodi. Tutkimustehtävä on toteutettu luvussa 8.

Toisena tutkimustehtävänä on sijoittaa ja vertailla suunnittelumallilla ja ilman suunnittelumallia tehtyjä relaatiotietokannan päivityksiä arkkitehtuuriympäristössä. Kerrosarkkitehtuuri-arkkitehtuurityyli on valittu arkkitehtuuriympäristöksi. Toinen tutkimustehtävä on toteutettu luvussa 9.

Testaan hypoteesia, että Tehdasmetodilla kerrosarkkitehtuuriin toteutettu päivitys relaatiotietokantaan on monipuolisempi kuin suora päivitys relaatiotietokantaan ilman suunnittelumallia. Tutkimus toteutetaan tiukasti rajatuilla tutkimustehtävien päivitysesimerkeillä. Suunnitellut päivitykset kuvataan UML:n viestiyhteyskaaviolla. Tehdasmetodilla ja ilman suunnittelumallia suunnitellut päivitykset siirretään kerrosarkkitehtuuriympäristöön. Arkkitehtuurin ATAM-arviointimenetelmällä vertaillaan suunnitelmien välisiä eroja. Arvioinnin löydökset raportoidaan.

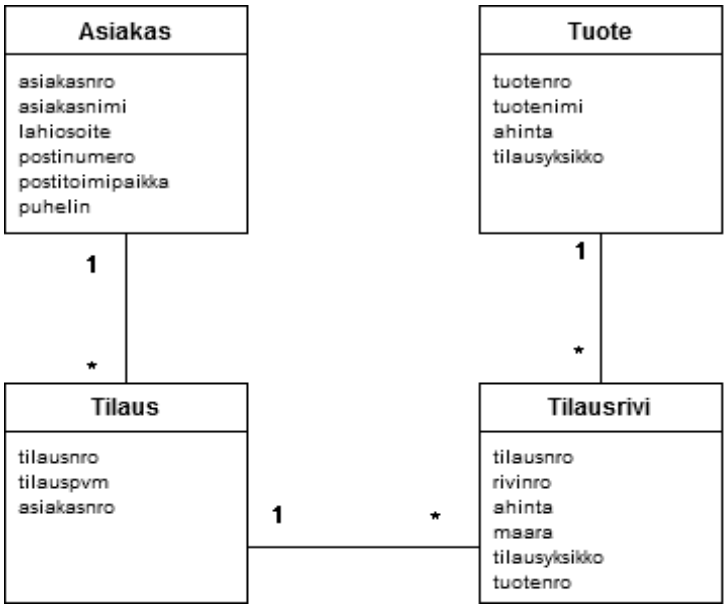
8. RELAATIOTIETOKANTAPÄIVITYKSET

Tässä luvussa toteutetaan tutkielman ensimmäinen tutkimustehtävä, esittelemällä relaatiotietokannan päivityksen osamallinnus UML-kaavioiden avulla. UML-kaaviot jakautuvat kahteen esitysosaan: ilman suunnittelumallia ja Tehdasmetodi-suunnittelumallilla tehtyyn toteutukseen. Molemmissa osissa ovat UML:n luokka- ja viestiyhteyskaaviot kuvausvälineinä.

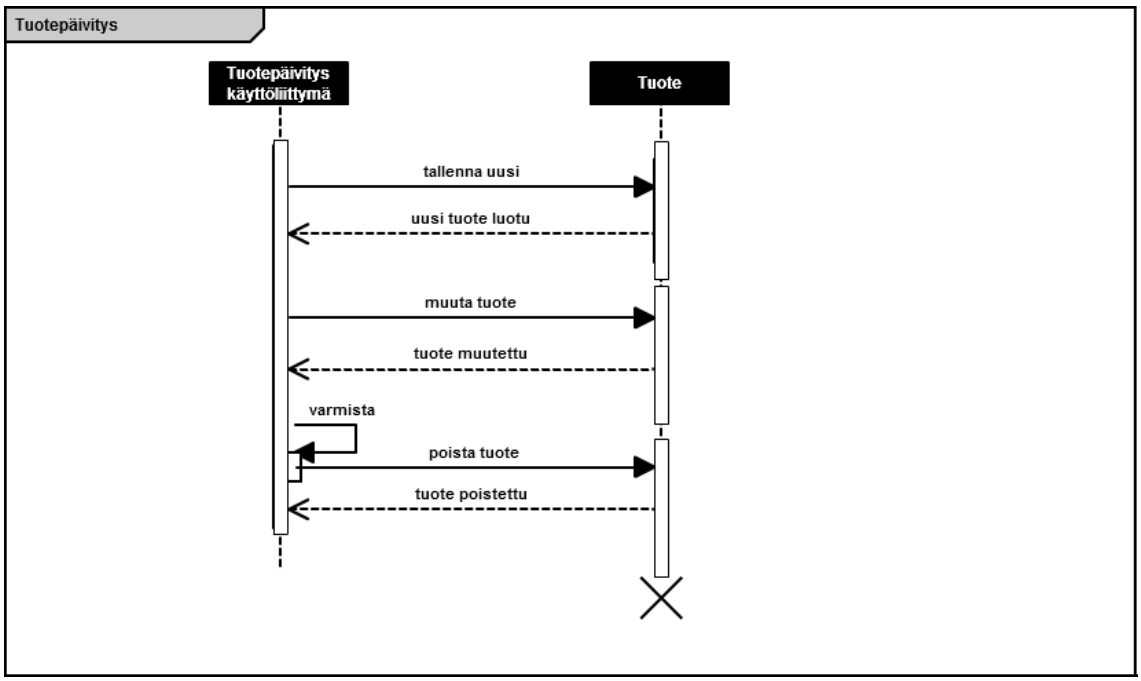
Luvussa 4 suunniteltiin relaatiotietokanta ja esimerkkinä käytettiin tilausjärjestelmän tietokantakaaviota. Tilausjärjestelmän tietokantakaaviosta on tehty luokkakaavio (kuva 8.1), josta käytetään Tuote-luokka -osaa edelleen esimerkkinä relaatiotietokannan päivityksen esittämiseen viestiyhteyskaavioilla.

8.1 Päivitys ilman suunnittelumallia

Ilman suunnittelumallia tehtävä relaatiotietokannan päivitys tarkoittaa staattista päivitystä suoraan tietokannan tuote-tauluun, ja olioita ei käytetä päivityksessä.



Kuva 8.1 Tilausjärjestelmän luokkakaavio, Tuote-luokkaa käytetään esimerkkinä päivityksessä



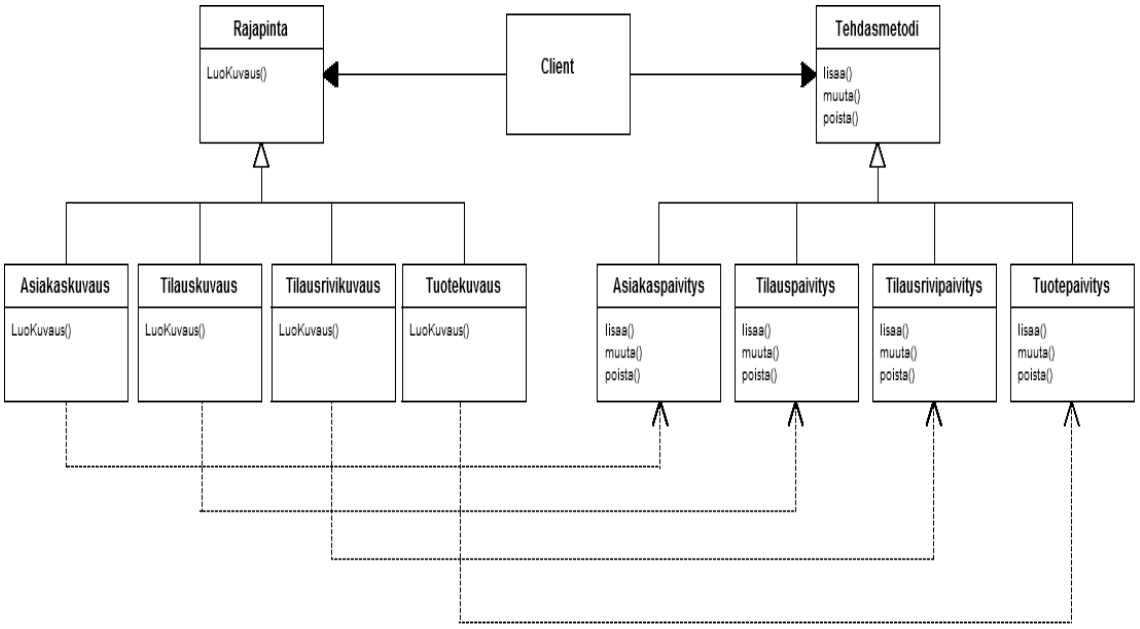
Kuva 8.2 Tuotetietojen päivitys esitettynä ilman suunnittelumallia viestiyhteyskaaviolla

Kuvan 8.2 viestiyhteyskaaviossa käyttöliittymältä tulee pyyntö uusien tuotetietojen lisäämiseksi tietokantaan. Tuoterivi tallennetaan tuote-tauluun, ja palautetaan ilmoitus tuotetietojen lisäämisestä käyttöliittymälle. Muutospäivityksessä toteutetaan käyttöliittymältä tullut tuotetietojen muutospyyntö tuote-tauluun. Muutospäivitys voi koskea vain osaa tuote-taulun sarakkeista. Muutospäivityksen jälkeen käyttöliittymä saa ilmoituksen tapahtuman onnistumisesta. Ennen tuotetietojen poiston toteuttamista tuote-taulusta tehdään käyttöliittymällä poiston varmistuskysely. Varmistuskyselyllä halutaan estää käyttäjää vahingossa tekemästä poistopäivitystä. Jos käyttäjä vastaa "kyllä" poistopäivityksen varmistuskyselyyn, poistetaan rivi tuote-taulusta. Käyttöliittymälle palautetaan ilmoitus tapahtuman onnistumisesta.

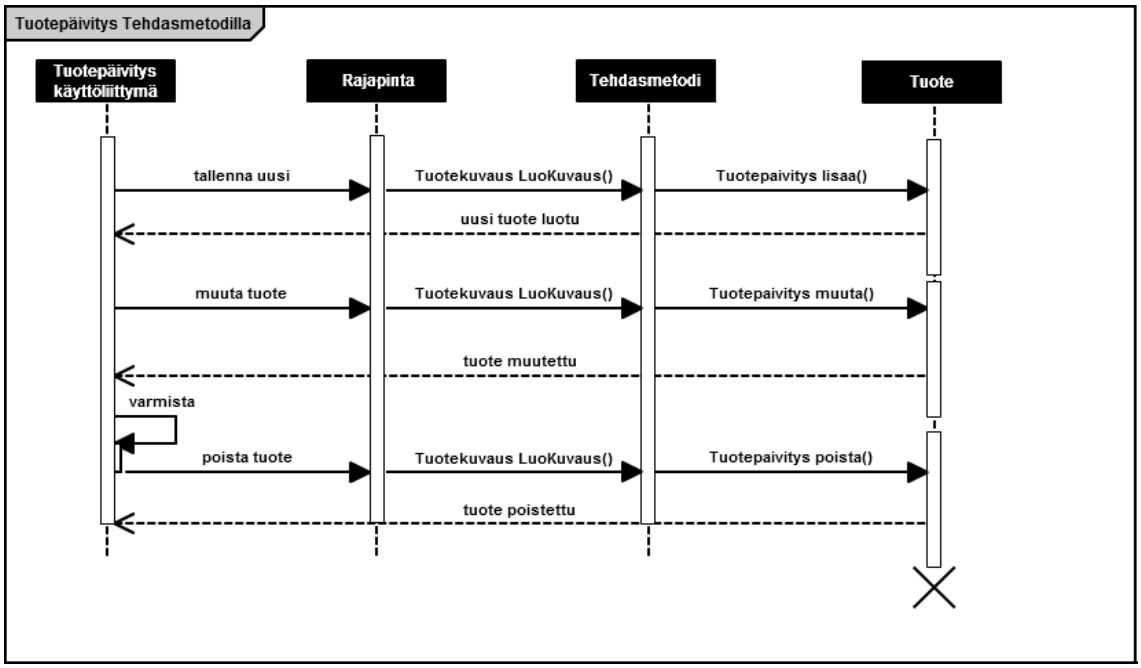
8.2 Päivitys Tehdasmetodi-suunnittelumallilla

Kuvassa 8.3 on Tehdasmetodi-suunnittelumallilla toteutettu UML-luokkakaavio tilausjärjestelmästä. Rajapinta-luokka määrittelee tilausjärjestelmän olioiden rajapinnan, joita Tehdasmetodi-luokka luo. Tilausjärjestelmä voi käyttää Rajapinta-luokan avulla Rajapinta-luokan aliluokissa toteutettuja ilmentymiä. Tehdasmetodi-suunnittelumalli määrittelee Rajapinta- ja Tehdasmetodi-luokkien välisen luokkahierarkioiden yhteyden.

Tehdasmetodi-luokasta löytyvät päivitysoperaatiot; lisaa(), muuta() ja poista(). Esimerkiksi Tuotepaivitys korvataan Tehdasmetodin operaatiolla, joka palauttaa Tuotekuvaus-ilmentymän. Tehdasmetodissa luokat ovat sovellusriippumattomia, koska koodi käsittelee Rajapinta-luokkaa.



Kuva 8.3 Tilausjärjestelmä Tehdasmetodi-suunnittelumallilla



Kuva 8.4 Tuotetietojen päivitys Tehdasmetodi-suunnittelumallilla esitettynä viestiyhteyskaaviolla

Kuvan 8.4 viestiyhteyskaaviossa käyttöliittymältä tulee pyyntö uusien tuotetietojen lisäämiseksi tietokantaan. Luodaan Rajapinta- ja Tehdasmetodi-luokkien avulla ilmentymä, jonka lisaa()-operaatiolla tuoterivi tallennetaan tuote-tauluun. Lisäksi palautetaan ilmoitus tuotetietojen lisäämisestä käyttöliittymälle ilmentymien kautta. Muutospäivityksessä luodaan ilmentymä samoin kuin uuden lisäyksessä. Muutospäivityksessä toteutetaan käyttöliittymältä tullut tuotetietojen muutospyyntö ilmentymän muuta()-operaatiolla tuote-tauluun. Muutospäivitys voi koskea vain osaa tuote-taulun sarakkeista. Muutospäivityksen jälkeen käyttöliittymä saa ilmoituksen tapahtuman onnistumisesta ilmentymien kautta. Ennen tuotetietojen poiston toteuttamista tuote-taulusta tehdään käyttöliittymällä poiston varmistuskysely. Varmistuskyselyllä halutaan estää käyttäjää vahingossa tekemästä poistopäivitystä. Jos käyttäjä vastaa "kyllä" poistopäivityksen varmistuskyselyyn, poistetaan rivi tuote-taulusta Rajapinta- ja Tehdasmetodi-luokkien avulla luodun ilmentymän poista()-operaatiolla. Käyttöliittymälle palautetaan ilmoitus tapahtuman onnistumisesta.

9. RELAATIOTIETOKANTA KERROSARKKITEHTUURI-YMPÄRISTÖSSÄ

Toisena tutkimustehtävänä on sijoittaa ja vertailla suunnittelumallilla ja ilman suunnittelumallia tehtyjä relaatiotietokannan päivityksiä kerrosarkkitehtuuri-ympäristössä. Luvussa kuvataan ensin päivitykset kerrosarkkitehtuuriympäristöön ja sitten on vuorossa vertailuosuus.

9.1 Päivitys ilman suunnittelumallia

Kuva 9.1 Tilausjärjestelmä kerrosarkkitehtuurissa ilman suunnittelumallia

Kuvan 9.1 tilausjärjestelmän ylimmäinen arkkitehtuurikerros on käyttöliittymä. Käyttöliittymän alapuolella on tilausjärjestelmäsovelluksen logiikkaa toteuttava kerros. Alimmassa kerroksessa on tilausjärjestelmän tietokanta tietoineen. Käyttöliittymästä on tehty myös suora porras tilausjärjestelmätietokantakerrokseen. Porras kuvaa tietoa, että suorat haut ja päivitykset tietokantaan ovat mahdollisia käyttöliittymältä. Sovelluslogiikkakerroksen ohittaminen ei ole hyvän ohjelmointitekniikan mukaista ja tietoturvallista. Käyttöliittymältä tulevat käskyt, esimerkiksi tiedon lisääminen tietokantaan, kannattaa kierrättää sovelluslogiikkakerroksen kautta.

Poikkeuksien hallintaa voidaan tehdä kaikissa kolmessa arkkitehtuurikerroksessa. Käyttöliittymällä voidaan nopeasti suodattaa tiedon muotoon liittyviä poikkeuksia, esimerkiksi päiväys. Käyttöliittymällä tapahtuvat tarkistukset ovat hyviä, koska poikkeamat saadaan heti syöttövaiheessa kiinni, ja käyttäjälle voidaan antaa palaute ja korjausmahdollisuus. Sovelluslogiikassa voidaan tehdä vaativampia tiedon tarkastuksia pienien ohjelmapätkien avulla, lisäksi voidaan hyödyntää tietokannan tietoja.

Sovelluslogiikkakerroksesta poikkeukset palautetaan käyttöliittymäkerrokselle. Tietokantakerrokseen asti pääseviä poikkeuksia tulisi välttää, koska poikkeusten palauttaminen käyttöliittymäkerrokselle on hankalampaa kuin ylemmistä kerroksista.

9.2 Päivitys Tehdasmetodi-suunnittelumallilla

Kuva 9.2 Tilausjärjestelmä kerrosarkkitehtuurissa Tehdasmetodi-suunnittelumallilla

Kuvan 9.2 tilausjärjestelmän ylimmäinen arkkitehtuurikerros on käyttöliittymä. Käyttöliittymän alapuolella on tilausjärjestelmäsovelluksen logiikkaa toteuttava kerros. Tilausjärjestelmäsovellus- ja tilausjärjestelmätietokantakerroksen välissä on tietokantalogiikkaa sisältävä tilausjärjestelmätehdasmetodikerros. Alimmassa kerroksessa on tilausjärjestelmän tietokanta tietoineen.

Kerrosten ohituksia ei ole tehty kuvan 9.2 kaavioon. Kaavion kutsut etenevät ylemmästä kerroksesta alaspäin. Mahdollinen ohitusporras olisi voitu tehdä käyttöliittymä- ja tilausjärjestelmäsovelluskerroksen välille eli käyttöliittymältä olisi kutsuttu suoraan tilausjärjestelmätehdasmetodin metodeja. Nykyisessä kaaviossa ko. kutsu välitetään tilausjärjestelmäsovelluskerroksen kautta. Tilausjärjestelmätehdasmetodikerrokseen on keskitetty tilausjärjestelmätietokantakerroksen tietokantalogiikka, siksi tämän kerroksen ohittaminen ei ole suositeltavaa ja ohitus mitätöisi Tehdasmetodisuunnittelumallin hyödyt.

Poikkeuksien käsittely on hyvin samankaltaista kuin ilman suunnittelumallia tehdyssä toteutuksessa. Tehdasmetodisuunnittelumallin kerrosarkkitehtuurissa on yksi kerros

enemmän. Poikkeustilanteiden viestinvälityksessä yksi lisäkerros hidastaa hiukan käsittelyä, kun viesti kulkee kerroksen läpi.

9.3 Päivitystoteutuksien laatuvertailu

Laatu-sanan määritteleminen ja laadun mittaaminen ovat vaikeita tehtäviä. Laatuun vaikuttaa paljon tarkastelunäkökulma. Laatu näyttää erilaiselta esimerkiksi käyttäjän, testaajan, suunnittelijan ja ohjelmiston näkökulmista tarkasteltuna. Yleisiä laatumalleja on luotu yrityksille avuksi standardisoimaan laatua esimerkiksi ISO9000-sarja laatustandardi, ISO/IEC 25010-laatumääritelmä (SQuaRE software product requirements and evaluation) ja amerikkalainen CISQ (Consortium for IT Software Quality). [Kasurinen, 2013, ss. 135-139]

Tutkimuksen laatuvertailussa valitaan ohjelmistotuotannon näkökulmasta laatutekijöitä, jotka ovat osittain samoja kuin yleisissä laatumallien standardeissa. Laatuvertailussa käytetään ATAM:in kaikkia neljää osa-aluetta: esittely, analyysi, testaus ja viimeisenä raportointi. Yleensä ensimmäisen menetelmävaiheen esittely- ja analyysiosaan osallistuvat arviointiryhmän lisäksi tuotannon vastuuhenkilöt. Toiseen menetelmävaiheen testaukseen ja raportointiin osallistuvat muut sidosryhmät, esimerkiksi asiakas. Tutkimuksessa ATAM-arkkitehtuurin arviointimenetelmässä käytetään kevyttä yhden henkilön arviointia. Kevyt organisaatio nopeuttaa myös arviointiprosessin läpivientiä, kun muille henkilöille ei jouduta esittelemään tilausjärjestelmän toimintaympäristöä.

9.3.1 Esittelyosio

ATAM-arkkitehtuurin arviointimenetelmää käytetään tilausjärjestelmän kahden eri toteutustavan laatuvertailuun ja parhaiten tilausjärjestelmälle sopivan päivitystoteutuksen etsimiseen. Toteutukset on esitelty edellä kerrosarkkitehtuuriympäristössä; päivitystoteutus ilman suunnittelumallia (kohta 9.1) ja päivitystoteutus Tehdasmetodi-suunnittelumallilla (kohta 9.2). Käytetty tiukasti rajattu tietokantapäivitys on vain osa tilausjärjestelmän toteutusta. Tietokantapäivitys on tärkeä osa tilausjärjestelmän liiketoiminnallista logiikkaa. ATAM:ssa käytettävään

skenaariotekniikkaan on etsitty tilausjärjestelmän toteutuksesta keskeisiä toteutuksen laatuun vaikuttavia tekijöitä.

9.3.2 Analyysiosio

Tilausjärjestelmästä on kerätty analysoitavaksi kuvassa 9.3 a-i kohdissa esiteltyjä laatutekijöitä. Laatupuuhun on kerätty keskeisemmät ohjelmistotuotannon laatuun vaikuttavat tekijät:

a) Ylläpidossa arvioidaan tilausjärjestelmän muunneltavuuden helppoutta ja korjausten toteuttamista järjestelmässä sekä sitä, pitääkö sama muutos tehdä moneen paikkaan. Liitetty skenaario eli laatuominaisuuden esimerkki on uuden kentän lisäys tuotteelle.

b) Suorituskyvyssä kiinnitetään huomio tilausjärjestelmän nopeuteen suorittaa päivitystapahtumia. Skenaario on lisäystapahtuman nopeus.

c) Muunneltavuusskenaariossa analysoidaan tilausjärjestelmän osien tai toimintojen vaihdettavuuden helppoutta. Skenaariossa vaihdetaan poikkeuskäsittely erilaiseksi tietokannan päivityksen yhteydessä.

d) Monistamisessa tilausjärjestelmän osia tai vaiheita voidaan suoraan käyttää tai toisintaa toisien järjestelmien osina. Skenaariossa tuotetietojen päivitys monistetaan asiakastietojen käsittelyyn.

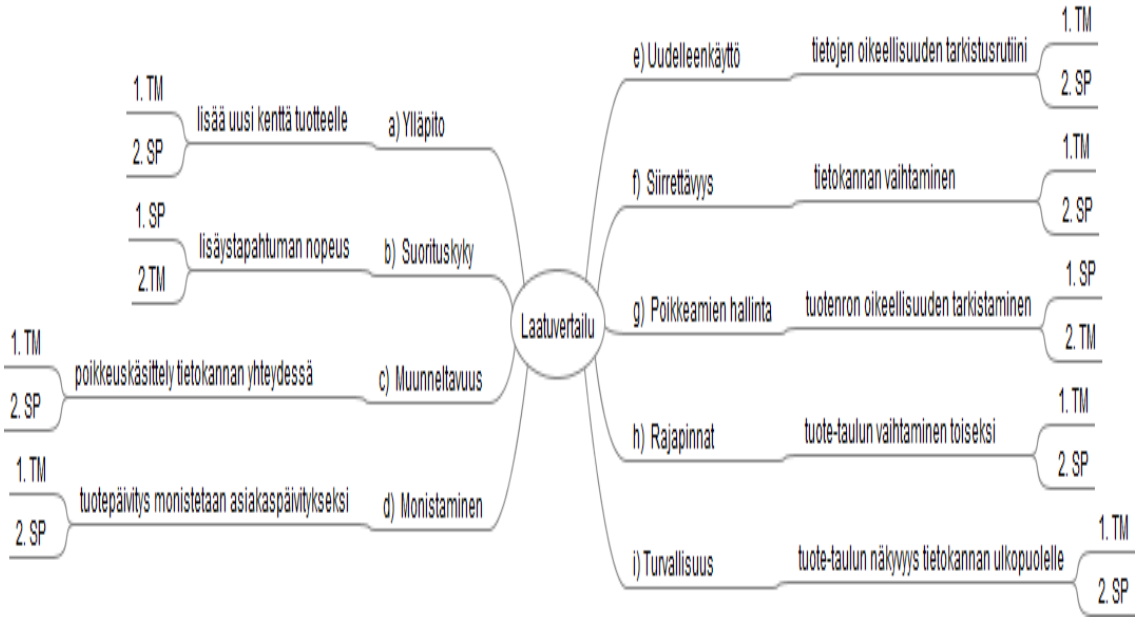
e) Uudelleenkäytössä analysoidaan tilausjärjestelmän osien ja vaiheiden käyttämistä sellaisenaan tai vähäisesti muuttamalla esimerkiksi toisessa kohtaa tilausjärjestelmää. Skenaariossa testataan tietojen oikeellisuuden tarkistusrutiinin uudelleenkäytettävyyttä.

f) Siirrettävyydessä arvioidaan tilausjärjestelmän osien tai vaiheiden muutos- ja sovitustyön määrää siirrettäessä niitä järjestelmän sisällä tai osaksi toista järjestelmää. Siirrettävyyden skenaariossa testataan tietokannan vaihtamista.

g) Poikkeamien hallinnassa keskitytään virhetilanteista toipumiseen ja virheilmoituksien toimivuuteen. Poikkeamaskenaario on tuotenumero-kentän oikeellisuuden tarkistaminen.

h) Rajapinnoissa tarkastellaan niiden olemassaoloa ja hyödyllisyyttä. Skenaariossa vaihdetaan tuote-taulu toisenlaiseksi.

i) Turvallisuutta katsotaan tiedon kannalta ja ulkopuoliselta uhkalta suojautumisena. Turvallisuusskenaariossa tutkitaan tuote-taulun tietojen näkyvyyttä tietokannan ulkopuolelle.



Kuva 9.3 Laatupuuvertailu tilausjärjestelmän kahden erilaisen päivitystoteutuksen paremmuudesta

9.3.3 Testiosio

Testiosiossa on liitetty yhteen laatupuu ja tilausjärjestelmän kahden eri toteutustavan laatuvertailu ja parhaiten tilausjärjestelmälle sopiva päivitystoteutuksen etsiminen. Toteutukset ovat päivitystoteutus ilman suunnittelumallia (lyhenne SP=suorapäivitys) ja päivitystoteutus Tehdasmetodi-suunnittelumallilla (lyhenne TM=Tehdasmetodi). Arviointi on toteutettu antamalla 1. ja 2. (palkinto)sijoituksia toteutuksille. Ykkössijoituksen saa toteutus, joka toteuttaa parhaiten skenaarion tilausjärjestelmässä. Eniten ykkössijoituksia kerännyt toteutus vastaa parhaiten tilausjärjestelmän laatuominaisuuksia. Seuraavaksi on esitetty perusteluita skenaarioiden arviointeihin.

a) Ylläpidossa Tehdasmetodi toteuttaa uuden kentän lisäyksen tuotteelle paremmin kuin suora päivitys, koska muutos voidaan tehdä helposti vain yhteen paikkaan. Suorassa

päivityksessä muutettavia kohtia voi olla monta, esimerkiksi tuotteen tietoja päivitetään monesta kohtaa tilausjärjestelmässä.

b) Suorituskyvyn skenaarion lisäystapahtuman päivitysnopeudesta tietokantaan toteuttaa suorapäivitys nopeammin kuin Tehdasmetoditoteutus. Suorapäivitys käyttää vähemmän keskusyksikön suoritusaikaa tehden lisäyspäivityksen suoraan tietokantaan. Tehdasmetoditoteutuksessa keskusyksikön suoritusaikaa kuluu vähän enemmän suoritettaessa useita kerroksia ennen varsinaista tietokannan lisäyspäivitystä.

c) Tehdasmetodissa on helppoa toteuttaa muunneltavuusskenaario. Skenaariossa tietokannan päivityksen yhteydessä oleva poikkeuskäsittely vaihdetaan erilaiseksi. Tehdasmetodissa poikkeuskäsittely on toteutettu vain kertaalleen, kun suorapäivityksessä poikkeuskäsittelyä voi olla useammassa kohdassa toteutusta. Yhden kohdan muuttaminen on helposti löydettävissä ja nopea vaihtaa toisenlaiseen toteutukseen.

d) Monistamisskenaariossa tuotetietojen päivitys monistetaan asiakastietojen käsittelyyn. Skenaario on hoituu Tehdasmetodilla paremmin kuin suoralla päivityksellä, koska Tehdasmetodi on toteutettu rakenteellisilla komponenteilla.

e) Skenaariossa testataan tietojen oikeellisuuden tarkistusrutiinin uudelleenkäytettävyyttä. Uudelleenkäyttö on helpompaa Tehdasmetodilla, koska tarkistusrutiini on omana kokonaisuutena tilausjärjestelmässä. Suoran päivityksen yhteydessä voi olla vain yksittäisen tiedon tarkistus, erilliset tarkistukset täytyisi kerätä ensin yhteen uudelleenkäytön mahdollistamiseksi.

f) Siirrettävyyden skenaariossa testataan tietokannan vaihtamista. Tietokannan vaihtamisesta selviää Tehdasmetodi paremmin kuin suorapäivitys. Tehdasmetodin komponenttien siirtäminen ja sovittaminen uuteen paikkaan on työmäärältään pienempi kuin suorapäivityksessä. Komponenttien rajapinnat helpottavat työtä, koska komponenttia kutsuvaan päähän ei tarvitse tehdä isoja muutoksia. Suorapäivityksen tietokantapäivitysten runsas määrä lisää työmäärää.

g) Poikkeamaskenaario on tuotenumero-kentän oikeellisuuden tarkistaminen. Suorapäivityksessä onnistuu virheilmoituksen välitys ja toipuminen virhetilanteesta

paremmin kuin Tehdasmetodissa. Tehdasmetodissa joudutaan virheilmoitusta kuljettamaan usean kerroksen ja komponentin kautta, suorassa päivityksessä välitys on suoraviivaisempaa ja selkeämpää, kun virheilmoitus on lähempänä käyttäjää.

h) Rajapintaskenaariossa vaihdetaan tuote-taulu toisenlaiseksi. Tehdasmetodissa on komponenteilla selvät rajapinnat, joiden avulla komponentin sisällön vaihtaminen erilaiseksi on helppoa. Suorapäivityksessä tuote-taulun käsittelyä voi olla useassa kohdassa, mikä lisää tuote-taulun vaihtamisen työmäärää moninkertaiseksi ja samalla on myös kalliimpaa kuin kertaalleen tehty työ.

i) Turvallisuusskenaariossa tutkitaan tuote-taulun tietojen näkyvyyttä tietokannan ulkopuolelle. Tehdasmetodissa voidaan tieto suojata ja piilottaa ulkopuolisilta paremmin kuin suorapäivityksessä, jossa tiedot voivat olla hyvinkin avoimesti näkyvissä ulkopuolisille. Tehdasmetodissa voidaan näyttää ulkopuolelle vain rajapinnan tiedot ja komponentin sisällä businesslogiikka on piilossa.

9.3.4 Raporttiosio

Testiosiossa on testattu ja analysoitu skenaarioiden avulla tilausjärjestelmän laatuominaisuuksia kahdella erilaisella päivitystoteutuksella. Samalla on syntynyt luettelo tärkeistä laatuominaisuuksista ja tulos päivitystoteutuksien välisestä paremmuudesta yksittäisten laatuominaisuuksien kohdalla. Laatupuuhun on kirjattu yhdeksän laatuominaisuutta. Kahdessa laatuominaisuudessa suorapäivitystoteutus oli parempi kuin Tehdasmetoditoteutus, lopuissa seitsemässä laatuominaisuudessa Tehdasmetoditoteutus oli ykkönen. Suorituskyvyn ja poikkeusten hallinnassa suorapäivitystoteutus oli ykkönen. Muissa ominaisuuksissa Tehdasmetodilla toteutettu päivitys loisti komponenttirakenteensa avulla. Komponenttirakenteessa on selkeät rajapinnat, ja ne kokoavat myös päivitykset suoritettavaksi vain yhdessä paikassa. Tehdasmetoditoteutus oli ykkönen ylläpidettävyydessä, muunneltavuudessa, monistamisessa, uudelleenkäytössä, siirrettävyydessä, rajapinnassa ja turvallisuudessa. Tehdasmetodin hyvät laatuominaisuudet tekevät tilausjärjestelmästä helposti hallittavan ja muunneltavan. Tehdasmetodi tukee isojen sekä pienien ohjelman osien monipuolista käyttämistä tilausjärjestelmässä tai toisissa järjestelmissä. Tehdasmetodi-suunnittelumallin käyttö lisää tilausjärjestelmän selkeyttä ja dokumentointia.

10. YHTEENVETO

Tässä yhteenvedossa esitellään tutkimusta, sen toteutusta ja tuloksia. Tutkimuksen aiheena oli ohjelmoinnin suunnittelumallin soveltaminen kerrosarkkitehtuuri-ympäristöön. Alkuluvuissa esiteltiin työvälineitä: arkkitehtuurityylejä, suunnittelumalleja, UML-mallinnuskieli, tietokanta ja ATAM-arviointiväline. Tutkimustehtäviä on kaksi. Ensimmäisessä tutkimustehtävässä toteutettiin UML-kaaviolla relaatiotietokannan päivitys suunnittelumallilla ja ilman suunnittelumallia. Suunnittelumallina vertailussa käytettiin Tehdasmetodia. Toisessa tutkimustehtävässä edellä mainitut päivitykset sijoitettiin kerrosarkkitehtuuriympäristöön, jossa niiden sopivuutta arvioitiin.

Vertailutoteutus tapahtui tiukasti rajatuilla päivitysesimerkeillä. Tuotetietojen päivityksestä tehtiin toteutuksessa UML-kaaviot Tehdasmetodin suunnittelumallilla ja ilman suunnittelumallia. Päivitykset siirrettiin kerrosarkkitehtuuriympäristöön kahdeksi erilliseksi osatoteutukseksi. Toteutuksessa etsittiin ATAM-analyysivälineellä eroja Tehdasmetodi-suunnittelumallilla ja ilman suunnittelumallia tehdyistä osatoteutuksista. Analyysissa raportoitiin osatoteutusten yhdeksän eri skenaarion keskinäisestä paremmuudesta.

ATAM-analyysin tuloksena Tehdasmetodi-suunnittelumallilla toteutetut laatuominaisuudet olivat ylivoimaisesti parempia kuin ilman suunnittelumallia toteutetut. Ilman suunnittelumallia toteutetut suorituskyky ja poikkeamien hallinta olivat parempia laatuominaisuuksiltaan kuin Tehdasmetodilla toteutetut. Tehdasmetodi-suunnittelumallilla toteutetut laatuominaisuudet olivat ykkösenä ylläpidettävyydessä, muunneltavuudessa, monistamisessa, uudelleenkäytössä, siirrettävyydessä, rajapinnassa ja turvallisuudessa. Tehdasmetoditoteutuksen hyvään menestykseen auttoivat suunnittelumallin komponenttirakenne, selkeät rajapinnat ja päivityksien suorittaminen keskitetysti yhdessä paikassa. Tehdasmetodi-suunnittelumalli tukee pienten ja isojen ohjelman osien uudelleenkäyttöä, lisäksi järjestelmän muunneltavuus ja hallittavuus paranee. Tehdasmetodi selkeyttää järjestelmän rakennetta ja auttaa dokumentoinnissa.

11. LOPPUPÄÄTELMÄT

Tämä luku alkaa työn kriittisellä arvioinnilla. Lisäksi kerrotaan suunnittelumallien käyttämisen soveltuvuudesta oppivan organisaation tiedonvälityksessä ja relaatiotietokannassa. Lopuksi pohditaan suunnittelumallien tulevaisuutta.

11.1 Työn kriittinen arviointi

Tutkielman tiukasti tehty rajaus ei anna oikeutta ja riittävää kuvaa olemassa olevasta suunnittelumallien laajasta tarjonnasta. Tarjonnasta löytynee monia vaihtoehtoja relaatiotietokannan päivityksen toteuttamiseksi. Tehdasmetodi ja sen soveltava esittely on vain yksi vaihtoehto muiden joukossa. Esimerkiksi luontimalleissa olevaa Abstrakti Tehdas -suunnittelumallia olisi voitu käyttää soveltuvin osin Tehdasmetodin tilalla.

11.2 Suunnittelumallit, ohjelmistoprosessi ja oppivaa organisaatiota

Riikka Ahlgren [2011] käsitteli ohjelmistoprosessin parantamista organisaation oppimisen kannalta. Ohjelmistoprosessin parantamiseen tarvitaan sekä yksilöiden oppimista että opitun tiedon sisäistämistä osana koko organisaation toimintatapoja ja -rakenteita, eli organisaation oppimista. Ahlgrenin tarkoituksena oli löytää käytännön keinoja ohjelmistoprosessin parantamiseen. Tutkimuskohde tarkentui erityisesti tietämyksen jakamiseen ja hallintaan. Tutkimuksessa analysoitiin suunnittelumalleja ohjelmistosuunnittelussa käytettävän tiedon jakamisvälineenä.

Tutkimuksen tulokset osoittivat, että tehokkaalla tiedon hallinnalla voidaan tukea organisaation oppimista. Suunnittelumalleja käytettiin väylänä dokumentoida, tallentaa ja jakaa osaamista, jota tarvitaan ohjelmistojen suunnittelussa. Lisäksi suunnittelumallit tarjoavat organisaation eri tasoille selkeät ja mitattavat tavoitteet prosessin parantamiseen. Suunnittelumallien hyödyntämisen ohjelmistojen kehityksessä pitää olla etukäteen hyvin suunniteltua ja päivittäistä rutiinia.

Ahlgren yhdisti tutkimuksessaan suunnittelumallit uudella mielenkiintoisella tavalla organisaation oppimiseen ja ohjelmistoprosessien parantamiseen. Tutkimuksen tuloksia

voidaan hyödyntää ohjelmistoyrityksissä ohjelmistokehittäjien oppimisen ja kommunikoinnin tukemiseen. Ahlgrenin tutkimus avaa uusia suunnittelumalleja hyödyntäviä tutkimuskohteita. [Ahlgren, 2011, s. 57]

11.3 Relaatiotietokantojen suunnittelumallit

Haraty ja Stephan [2013] esittelevät 24 suunnittelumallia relaatiotietokannalle avoimen lähdekoodin ja web-sovellusten ympäristössä. Objekteihin pohjautuvassa ohjelmistosuunnittelussa on jo vuosia hyödynnetty hyvällä menestyksellä suunnittelumalleja parantamaan toimitusnopeutta ja nostamaan tuotteen laatua. Tietokannan suunnittelussa ei ole vielä osattu hyödyntää suunnittelumallien tarjoamia mahdollisuuksia. Suunnittelumallit sopivat yhtä hyvin tietokannan suunnitteluun kuin objekteihin pohjautuvaan ohjelmistosuunnitteluun. Relaatiotietokannan suunnittelumallien aiheista muutama esimerkki: käyttäjien hallintataulu, taulun tietosisällön luontitiedot ja lokitietojen kerääminen virhetilanteessa. Haratyn ja Stephanin mielestä relaatiotietokannan suunnittelumallit tarjoavat opastavan avun varsinkin aloitteleville suunnittelijoille.

11.4 Suunnittelumallien tulevaisuus

Suunnittelumallit ovat vanha hyvä keksintö, joka on säilyttänyt yllättävän hyvin ajankohtaisuutensa nykypäivään asti. Suunnittelumallit säilyvät myös tulevaisuudessa, jos niitä käytetään luovasti uudistaen ja nykypäivän ympäristöön soveltaen. Tästä on hyvänä esimerkkinä edellä mainitut Ahlgrenin suunnittelumallien käyttö sekä Haratyn ja Stephanin esittämä suunnittelumallien laajennettu käyttö relaatiotietokannan suunnittelussa.

Vanhoja suunnittelumalleja ei kannata keksiä uudelleen, mutta niitä voi hyödyntää ja parannella. Suunnittelijoiden kannattaa säilyttää vanhat suunnittelumallit edelleen työkalupakkinsa sopukoissa ja kehittää lisää uusia suunnittelumalleja. Suunnittelumalleissa on valmiiksi mietitty tiettyyn ongelmaan sopiva rakenteellinen ratkaisu. Voimme löytää suunnittelumallien joukosta sopivia ratkaisuja järjestämään päivittäistä tietokaaostamme.

LÄHTEET

[Ahlgren, 2011] Riikka Ahlgren. *Software Patterns, Organizational Learning and Software Process Improvement*. Jyväskylän yliopisto, Informaatioteknologian tiedekunta, Väitöskirja, Jyväskylä, 2011.

[Alexander *et al*., 1977] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angel. *A Pattern Language.* Oxford University Press, New York, 1977.

[ATAM, 2014] Software Engineering Institute, Carnegie Mellon University. 2014. *Architecture Tradeoff Analysis Method*. http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm. Checked 12.06.2014

[ATK-sanakirja, 1997] Tietotekniikan liitto ry:n sanatoimikunta, *ATK-sanakirja*. 9. painos. Espoo: Suomen Atk-kustannus Oy, Jyväskylä, 1997.

[Buschmann *et al*., 1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerland, and Michael Stal, *Pattern-Oriented Software Architecture, volume 1: A System of Patterns.* Wiley, 1996.

[Chen, 1976] Peter Pin-Chan Chen , *The Entity-Relationship Model--Toward a Unified View of Data,* ACM Transactions on Database Systems, **1**, 1, March, 1976, Pages 9–36.

[Clements *et al*., 2002] Paul Clements, Rick Kazman and Mark Klein, *Evaluating Software Architectures,* SEI Series in Software Engineering, Addison-Wesley, 2002.

[Codd, 1970] Edgar F. Codd, *A Relational Model of Data for Large Shared Data Banks,* Communications of the AMC, **13**, 6, June, 1970, Pages 377-387.

[Elmasri and Navathe, 2007] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* 5th Edition, Addison Wesley, Boston, 2007.

[Fowler and Scott, 2002] Martin Fowler and Kendall Scott. *UML.* Suom. E. Sarkkinen, 2. laitos, 1. painos, Docendo Finland Oy, Jyväskylä, 2002.

[Gamma *et al*., 2005] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software.* 32nd Printing, Addison-Wesley, 2001.

[Gamma *et al*., 2001] Erich Gamma, Richard Helm, Ralph Johnson ja John Vlissides. *Olio-ohjelmointi suunnittelumallit.* Suom. A. Toivonen, IT Press, Oy Edita Ab, Helsinki, 2001.

[Grässle *et al.,* 2005] Patrick Grässle, Henriette Baumann and Philippe Baumann. *UML 2.0 in Action, a Project-Based Tutorial.* Packt Publishing Ltd, Birmingham, UK, 2005.

[Haikala ja Mikkonen, 2011] Ilkka Haikala ja Tommi Mikkonen. *Ohjelmistotuotannon käytännöt.* 12. uudistettu painos, Talentum Media Oy, Hämeenlinna, 2011.

[Haraty and Stephan, 2013] Ramzi A. Haraty and Georges Stephan, *Relational Database Design Patterns,* IEEE 16th International Conference on Computational Science and Engineering, Lebanese American University, Beirut, Lebanon, 2013, Pages 818-824.

[Hovi *et al.,* 2005] Ari Hovi, Jouni Huotari ja Tapio Lahdenmäki. *Tietokantojen suunnittelu & indeksointi*. Docendo Finland Oy, Jyväskylä, 2005.

[IBM, 2003] IBM Archives Edgar F. Codd. Appeared 23.04.2003. http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html, Checked 05.06.2014.

[Järvinen ja Järvinen, 1994] Pertti Järvinen ja Annikki Järvinen, *Tutkimustyön metodeista*. Opinpajan kirja, Tampere, 1994.

[Kasurinen, 2013] Jussi-Pekka Kasurinen, *Ohjelmistotestauksen käsikirja,* Docendo, 2013.

[Koskimies ja Mikkonen, 2005] Kai Koskimies ja Tommi Mikkonen. *Ohjelmisto-arkkitehtuurit.* Talentum Media Oy, Jyväskylä, 2005.

[OMG, 2013] Object Management Group (OMG). *Getting started with UML.* 2013, http://www.uml.org/. Checked 16.11.2013

LIITE: SUUNNITTELUMALLIEN HAKEMISTO

Hakemisto sisältää 23 suunnittelumallin nimet ja tarkoitukset suomennetusta Design Patterns kirjasta [Gamma *et al*., 2001, ss. 8-9]. Suunnittelumallien ryhmittely luonti-, rakenne- ja käyttäytymismalleihin tulee englanninkielisestä Design Patterns kirjasta [Gamma *et al*., 2005, s. etukansi].

Luontimallit (creational patterns)

Abstrakti Tehdas (abstract factory) Tuottaa rajapinnan, jolla luodaan toisiinsa liittyvien olioiden muodostamia olioperheitä määrittelemättä olioiden konkreettisia luokkia.

Rakenteja (builder) Malli erottaa toisistaan monimutkaisen olion rakentamisessa käytetyn prosessin ja olion esitysmuodon, jolloin samalla rakentamisprosessilla voidaan tuottaa erilaisia esitysmuotoja.

Tehdasmetodi (factory method) Määrittelee olion luontioperaation kutsumuodon, mutta jättää aliluokkien tehtäväksi päättää, mistä luokasta ilmentymä luodaan. Tehdasmetodia käyttämällä luokka voi siirtää ilmentymien luonnin aliluokille.

Prototyyppi (prototype) Määrittelee prototyyppi-ilmentymää käyttämällä millainen olio luodaan, ja luo uusia olioita tätä prototyyppiä kopioimalla.

Ainokainen (singleton) Varmistaa, että luokasta luodaan vain yksi ilmentymä, ja tarjoaa globaalin tavan päästä käsiksi tähän ilmentymään.

Rakennemallit (structural patterns)

Sovitin (adapter) Konvertoi luokan rajapinnan toiseksi rajapinnaksi, joka vastaa sovelluksen tarpeista. Sovittimen avulla saadaan sellaiset luokat, jolla on epäyhteensopivat rajapinnat, toimimaan yhdessä.

Silta (bridge) Erottaa rajapinnan toteutuksesta, jolloin kumpaakin voidaan muutella toisistaan riippumattomasti.

Rekursiokooste (composite) Malli esittää oliot rekursiivisesti koostettuna puurakenteena (part-whole hierasrchy). Yksittäisiä olioita ja oliokoosteita voidaan käsitellä samalla tavalla.

Kuorruttaja (decorator) Lisää olioille dynaamisesti uusia vastuita. Kuorruttaja tarjoaa joustavan vaihtoehdon perinnälle, kun on tarpeen laajentaa toiminnallisuutta.

Julkisivu (facade) Tarjoaa yhtenäisen rajapinnan alijärjestelmän rajapintojen joukolle. Julkisivu tarjoaa korkeamman tason rajapinnan, jonka kautta alijärjestelmää on helpompi käyttää.

Hiutale (flyweight) Mahdollistaa yhteiskäytön kautta runsaslukuisten hienojakoisten olioiden tehokkaan käytön.

Edustaja (proxy) Tuottaa olioille korvikkeen tai paikanpitäjän, joka kontrolloi olioon kohdistuvia pyyntöjä.

Käyttäytymismallit (behavioral patterns)

Vastuuketju (chain of responsibility) Pyynnön lähettäjän sitominen vastaanottajaan vältetään antamalla useammalle kuin yhdelle oliolle mahdollisuus pyynnön käsittelyyn. Vastaanottavista olioista muodostetaan ketju, ja pyyntöä siirretään ketjussa kunnes joku olioista käsittelee sen.

Komento (command) Kapseloi pyynnön olioksi. Tämän ratkaisun avulla voidaan asiakas parametroida lähettämään erilaisia pyyntöjä, laittaa pyynnöt jonoon, pitää niistä lokia ja peruuttaa operaatioita.

Tulkki (interpreter) Määrittelee annetun kielen kieliopille esitysmuodon ja tulkin, joka käyttää esitysmuotoa kielen lauseiden tulkitsemiseen.

Iteraattori (iterator) Tarjoaa tavan, jolla kokoelmaolion alkiot saadaan läpikäytyä peräkkäisjärjestyksessä paljastamatta rakenteen sisäistä esitystapaa.

Välittäjä (mediator) Esittelee olion, johon kapseloidaan oliojoukon väliset vuorovaikutustavat. Välittäjä edistää löyhää sidontaa estämällä olioita viittaamasta suoraan toisiinsa ja mahdollistaa vuorovaikutuksen muuttamisen kaikkia yhteyksiä muuttamatta.

Muisto (memento) Rikkomatta kapselointia kokoaa ja ulkoistaa olion sisäisen tilan siten, että olio voidaan myöhemmin palauttaa tähän tilaan.

Tarkkailija (observer) Määrittelee olioiden välille yksi moneen -riippuvuuden siten, että kun yhden olion tila muuttuu, siitä riippuvat oliot saavat ilmoituksen ja päivittyvät automaattisesti.

Tila (state) Malli saa olion muuttamaan käyttäytymistään, kun sen sisäinen tila muuttuu. Näyttää siltä kuin olio olisi vaihtanut luokkaansa.

Strategia (strategy) Määrittelee algoritmiperheen, kapseloi kunkin algoritmin ja tekee niistä keskenään vaihdettavia. Algoritmia voidaan muuttaa muuttamatta sovellusta, joka sitä käyttää.

Operaatiorunko (template method) Määrittelee algoritmin rungon operaatiossa ja jättää jotkin sen osat aliluokkien toteutettavaksi. Operaatiorunko sallii aliluokkien uudelleen määritellä tietyt algoritmin kohdat, mutta algoritmin rakenne pysyy muuttumattomana.

Vierailija (visitor) Edustaa operaatiota, joka suoritetaan oliorakenteen elementeille. Elementtien luokkia ei tarvitse muuttaa, kun luodaan uusi niihin kohdistuva operaatio.