Juha Rautiainen

UML-MALLINNUSKIELI JA SEN HYÖDYNTÄMINEN OHJELMISTOKEHITYKSESSÄ

Tietotekniikan kandidaatintutkielma

20.3.2011

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Juha Rautiainen

Yhteystiedot: juhar@iki.fi

Työn nimi: UML-mallinnuskieli ja sen hyödyntäminen ohjelmistokehityksessä

Title in English: UML Modeling Language and Using It for Software Development

Työ: Tietotekniikan kandidaatintutkielma

Sivumäärä: 37

Suuntautumisvaihtoehto: Ohjelmistotekniikka.

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: UML, mallinnuskieli, rakennekaavio, käyttäytymiskaavio, ohjelmistokehitysprosessi

Keywords: UML, modeling language, structural diagram, behavioral diagram, software development process

Tiivistelmä: Tutkielmassa tarkastellaan kirjallisuuden pohjalta Unified Modeling Language -mallinnuskielen käsitteistöä ja sen hyödyntämistä oliosuuntautuneessa ohjelmistokehitysprosessissa. Tutkielma esittelee UML:n historiaa, kaaviotyyppejä ja niihin liittyviä elementtejä. Lisäksi kuvataan, kuinka UML:a käytetään ohjelmistosuunnitteluprosessien eri toiminnoissa.

Abstract: The bachelor thesis deals with Unified Modeling Language terminology and how to use UML in object oriented software development process based on literature. The thesis introduces history of UML, UML diagrams and elements. It also describes how UML is used in different activities of software development processes.

Termiluettelo

CASE	(Computer Aided Software Engineering) eli tietokoneavusteinen ohjelmistosuunnittelu tarkoittaa ohjelmistoja, jotka automatisoivat ohjelmistokehitysprosessien toimintoihin liittyviä tehtäviä sekä auttavat toteutettavaan ohjelmistoon liittyvän informaation käsittelyssä.
Malli	on kohteena olevasta tietojärjestelmästä laadittu kuvaus, joka kuvaa järjestelmästä valitun asiakokonaisuuden käsittelyn kannalta oleelliset osat. Malleja käytetään ongelman ymmärtämisen ja rajaamisen apuna, sekä spesifikaatioiden ja dokumenttien laatimiseen.
Mallinnuskieli	on mallien esittämiseen tarkoitettu kuvaustapa, jonka rakenne ja rakenteiden merkitys on määritelty.
OMG	(Object Management Group) on organisaatio, joka vastaa muun muassa UML:n standardoinnista.
UML	(Unified Modeling Language) on ensisijaisesti tietojärjestelmien suunnitteluun tarkoitettu graafinen mallinnuskieli, jota käytetään olioperustaisessa ohjelmistosuunnittelussa tietojärjestelmän osien ja rakenteen määrittämiseen, visualisointiin ja dokumentointiin.

Sisältö

1 Johdanto

Ohjelmistokehityksessä kuvataan suunnittelun kohteena olevaa ohjelmistoa ja siihen liittyvää ongelma-aluetta yleisesti mallien avulla. Malleja hyödynnetään eri tavoin ohjelmistokehityksen eri vaiheissa. Esimerkiksi määrittelyvaiheessa mallit toimivat apuna määritettäessä sidosryhmien ohjelmistolle asettamat vaatimukset, ja toteutusvaiheessa ne ohjaavat ohjelmoijien työtä. Mallien esittämiseen käytetään erilaisia kuvaustapoja, kuten kaavioita, tekstiä tai taulukoita. Jos kuvaustapa on täsmällisesti määritelty, voidaan sitä kutsua mallinnuskieleksi.

Standardin [OMG 2009a, s. 9] mukaan Unified Modeling Language (UML) on yleiskäyttöinen graafinen tietojärjestelmien mallinnuskieli, joka on kehitetty sovellettavaksi mahdollisimman monen olemassa olevan oliosuuntautuneen ohjelmistokehitysmenetelmän prosessien yhteydessä. Kirjassa [Koskimies 2000, s. 20] todetaan UML:n eroavan useimmista sitä edeltäneistä kuvauskielistä. Se ei nimittäin ole sidoksissa tiettyyn prosessiin tai menetelmään, eikä näin ollen myöskään tiettyyn organisaatioon, kulttuuriin tai sovellusalueeseen. Useissa lähteissä, esimerkiksi [Koskimies 2000, s. 20] ja [Sommerville, s. 155], todetaan UML:n saavuttaneen olio-suuntautuneessa ohjelmistotuotannossa käytännön standardin aseman.

Tutkielmassa käsitellään lähdekirjallisuuden pohjalta Unified Modeling Language -mallinnuskielen kehitystä ja käsitteistöä sekä sen hyödyntämistä oliosuuntautuneessa ohjelmistokehitysprosessissa. Tutkielman avulla lukija voi perehtyä UML-mallinnuskielen keskeiseen käsitteistöön ja UML:n kuvaustapojen käyttöön ohjelmistokehitysprosessien eri vaiheissa. Vaikka UML:a voidaan ohjelmistotekniikan lisäksi soveltaa myös esimerkiksi liiketoimintamallien kuvaamiseen, tutkielma keskittyy ensisijaisesti UML:n käyttöön ohjelmistokehityksessä. Lukijan oletetaan tuntevan olio-ohjelmoinnin peruskäsitteistön. Myös ohjelmistokehityksessä käytettävien suunnittelumenetelmien perustoimintojen tuntemisesta on lukijalle hyötyä.

Luku 2 kuvaa UML:n historiaa ja sitä edeltäneitä kuvausmenetelmiä. Luvussa 3 esitellään UML:en liittyvää käsitteistöä, yleisiä merkintätapoja, mallinnuselementtejä ja laajennusmekanismeja, joilla UML on sovitettavissa erilaisiin tarpeisiin. Luvuissa 4 ja 5 kuvataan

UML:n kaaviotyypit. Luvussa 6 tarkastellaan UML:n suhdetta ohjelmistosuunnittelumenetelmien määrittämien prosessien toimintoihin. UML-mallin laatimista on havainnollistettu esimerkin avulla luvussa 7.

2 Historiaa

Unified Modeling Languagea (UML) edelsi joukko erilaisia olioperustaisia suunnittelumenetelmiä, joita yhdistämällä se on kehitetty. Luvussa 2.1. esitellään UML:a edeltäneiden suunnittelumenetelmien kehitystä ja luvussa 2.2 UML-mallinnuskielen muodostamista niiden pohjalta. Luku 2.3 käsittelee UML-standardin syntyä, ja luvussa 2.4 esitellään lyhyesti UML:n versioiden kehitystä. Luku perustuu pääasiassa lähteisiin [Booch, s. xvii-xx], [Eriksson, s. 3-5] ja [Fowler, s. 7-9].

2.1 Suunnittelumenetelmiä ennen UML:a

Perinteisiä **proseduraalisia ohjelmointikieliä**, kuten Cobolia ja Fortrania, tukevat **määrittely- ja suunnittelumenetelmät** kehitettiin 1970-luvulla, sekä niiden käyttö yleistyi 1980-luvun aikana. Menetelmien odotettiin helpottavan monimutkaistuvien ja laajenevien tietojärjestelmien suunnittelun hallintaa. Näistä menetelmistä käytetyimpiä oli kirjan [Haikala, s. 179] mukaan SA-menetelmä (Structured Analysis). Menetelmien hyödyntämisestä saadut edut eivät aina vastanneet niille asetettuja odotuksia. Niihin kuitenkin sisältyi joukko hyviä ideoita, joita onnistuttiin satunnaisesti käyttämään tehokkaasti suuria järjestelmiä kehitettäessä.

Kirjassa [Fowler, s.7] todetaan olioperustaisen ohjelmoinnin alkaneen levitä tutkimuslaitosten ulkopuolelle 1980-luvulla Smalltalk- ja C++-ohjelmointikielten menestyksen myötä. **Olioperustaisten suunnittelumenetelmien** perustan muodostavat teokset julkaistiin vuosien 1988 ja 1992 välisenä aikana. Näistä ensimmäisinä ilmestyivät Sally Shlaerin ja Steve Mellorin sekä Peter Coadin ja Ed Yourdonin kirjat. Pian tämän jälkeen julkaisivat teoksissaan Rational Softwaren Grady Booch OOAD-menetelmän, General Electricin Jim Rumbaugh työryhmineen OMT-menetelmän ja Smalltalk-yhteisö oman menetelmänsä. Mainittavan arvoinen on myös Ivar Jacobsonin vuonna 1992 julkaisema Objectory-menetelmä. Se perustui osin aiempiin menetelmiin, mutta siinä näkökulma keskittyi niistä poiketen käyttötapauksiin ja ohjelmistokehitysprosessiin.

2.2 Menetelmien yhdistäminen

Kirjassa [Fowler, s. 7] todetaan, että 1990-luvun alussa julkaistuissa lukuisissa oliopohjaisia suunnittelumenetelmiä käsittelevissä kirjoissa kirjoittajat käyttivät toisistaan poikkeavia käsitteistöjä, määrittelyjä, merkitsemistapoja, termistöjä ja prosesseja. Joissain teoksissa esitettiin uusia hyödyllisiä käsitteitä, mutta yleisesti ottaen kirjoittajat tarkoittivat erilaisilla termeillä toisiaan vastaavia rakenteita. Ensimmäiset yritykset olioperustaisten suunnittelumenetelmien yhdistämiseksi tehtiin jo 1990-luvun alussa. Esimerkiksi Coleman työryhmineen pyrki yhdistämään Fusion-menetelmään OMT-, Booch- ja CRC-menetelmien käsitteet.

Ensimmäinen onnistunut suunnittelumenetelmien yhdistyminen käynnistyi vuonna 1994, kun OMT-menetelmän pääkehittäjä Rumbaugh palkattiin Rational Software Corporation -yhtiöön, jossa Booch työskenteli. Rumbaugh ja Booch alkoivat yhdistää OMT- ja Booch-menetelmien käsitteistöä **Unified Method** -menetelmäksi, josta julkaistiin ensimmäinen ehdotus vuonna 1995. Samaan aikaan myös Objectory- ja OOSE-menetelmien kehittäjä Ivar Jacobson siirtyi Rational Softwaren palvelukseen sen ostettua ruotsalaisen Objective Systemsin. Kirjan [Eriksson, s. 4] mukaan Rumbaugh, Booch ja Jacobson antoivat työlleen uudeksi nimeksi Unified Modeling Language, huomattuaan menetelmän sijasta itse asiassa kehittäneensä mallinnuskieltä.

2.3 UML:n standardointi

Vuonna 1996 Object Management Group (OMG) pyysi ehdotuksia oliomallinnusstandardiksi. Kirjassa [Fowler, s. 8] esitetään, että OMG:n kiinnostuksen taustalla oli Unified Metodin kiinteä kytkös Rational Software -yhtiöön, mikä aiheutti muissa CASE-välineiden valmistajissa huolta kilpailun vääristymisestä. OMG:n mukaantulon jälkeen Booch, Jacobson ja Rumbaugh alkoivat kehittää OMG:n jäsenistölle soveltuvaa menetelmää ja kuvauskieltä yhdessä eräiden muiden yhtiöiden edustajien kanssa.

Samaan aikaan kehitteillä oli myös useita kilpailevia standardiehdotuksia, mutta ne kaikki yhdistyivät lopulliseen UML-ehdotukseen, joka toimitettiin kirjan [Booch, s. xx] mukaan OMG:lle heinäkuussa 1997. Kirjassa [Fowler, s. 151] mainitaan ajankohdaksi syyskuu 1997, mutta se viitannee tällöin tapahtuneeseen OMG:n työryhmien hyväksyntään. OMG:n

jäsenet hyväksyivät UML:n standardiksi marraskuussa 1997, ja samalla OMG otti vastuulleen UML:n jatkokehityksen. Jo ennen standardin lopullista vahvistamista julkaistiin monia UML:n keskeisiä piirteitä käsitteleviä kirjoja, ja useat ohjelmistotoimittajat ilmoittivat tukevansa UML:a.

2.4 UML:n versiot

Ensimmäinen julkinen versio UML:sta oli lokakuussa 1995 julkaistu Unified Methodin versio 0.8. Vuonna 1996 julkaistiin versiot 0.9 ja 0.91. Tammikuussa 1997 OMG:lle toimitettu ehdotus oli UML:n versio 1.0. Lopullinen OMG:n marraskuussa 1997 hyväksymä versio oli UML 1.1, josta OMG kuitenkin päätti käyttää versionumeroa 1.0. Kirjan [Fowler, s. 151] mukaan standardoituun versioon viitataan kuitenkin käytännössä numerolla 1.1.

UML:n versio 1.2 julkaistiin vuonna 1998, versio 1.3 1999, versio 1.4 2001 ja versio 1.5 2002. Näiden versioiden muutokset kohdistuivat pääasiassa UML:n sisäisiin rakenteisiin, sekä versiossa 1.3 tehtiin näkyviä muutoksia erityisesti käyttötapaus- ja aktiviteettikaavioihin.

Vuonna 2003 julkaistiin tähän mennessä eniten uudistettu versio numerolla 2.0. Näkyvin muutos verrattuna aikaisempiin versioihin ovat uudet kaaviotyypit. UML 2.0:ssa on myös esimerkiksi parannettu rakennetta ja käyttäytymistä kuvaavien kaavioiden yhteyttä toisiinsa sekä olennaisesti laajennettu sekvenssikaavion ilmaisuvoimaa. Version 2.0 jälkeen on julkaistu versiot 2.1, 2.1.1, 2.1.2 ja 2.2, jotka sisältävät pieniksi luonnehdittuja korjauksia versioon 2.0.

3 UML-mallinnus

Suunnittelun kohteena olevaa ohjelmistoa ja siihen liittyvää ongelma-aluetta kuvataan ohjelmistokehityksessä yleisesti erilaisten mallien avulla. Mallissa kohteen tarkastelua pyritään yksinkertaistamaan kuvaamalla järjestelmästä vain valitun asiakokonaisuuden käsittelyn kannalta oleelliset osat ja poistamalla turhia yksityiskohtia.

Malleja voidaan käyttää paitsi ongelman ymmärtämisen ja rajaamisen apuna, myös spesifikaatioiden ja dokumenttien laatimiseen. Ohjelmistoa kehitettäessä ne toimivat ihmisten välisen kommunikaation tukena.

Luku 3.1 esittelee mallinnukseen ja UML:een liittyviä keskeisiä käsitteitä. Luvussa 3.2 käsitellään UML:n mallinnuselementtejä, joilla kuvataan UML-mallissa esitettävät käsitteet ja rakenteet. Luku 3.3 käsittelee yleisiä merkintätapoja, joiden avulla malliin voidaan liittää mallinnuselementtien perusominaisuuksiin kuulumatonta tietoa. Luku 3.4 käsittelee laajennusmekanismeja, joilla UML-mallinnuskieli voidaan sovittaa kulloiseenkin käyttötarkoitukseen muuttamatta itse kieltä. Luku 3.5 esittää periaatteen, joilla luvuissa 4 ja 5 esiteltävät UML:n kaaviotyypit on jaettu ryhmiin UML-standardin Superstructure-määrittelyssä [OMG 2009b, s. 686].

Luvuissa 3.2-3.4 noudatetaan kirjassa [Eriksson, s. 21-27] käytettyä jaottelua. Muita keskeisiä lähteitä ovat standardi [OMG 2009b, s. 58, 653, 683, 686 ja 695], kirja [Booch, s. 17-24, 27-30, 76-89] ja artikkeli [Koskimies 2010, s. 21-22, 30].

3.1 Mallinnuksen peruskäsitteitä

Mallien esittämiseen käytetään erilaisia kuvaustapoja, esimerkiksi graafisia kaavioita, tekstiä tai taulukoita. **Mallinnuskielestä** (engl. *modeling language*) voidaan puhua, kun kuvaustapa on täsmällisesti määritelty. Määrittelyn tulisi kuvata kielen rakenne ja rakenteiden merkitys.

UML on graafinen mallinnuskieli, joka kokoaa yhteen useita sitä edeltäneiden mallinnuskielten kuvaustapoja. UML:en sisältyviä osakieliä kutsutaan **kaaviotyypeiksi**. Toisaalta artikkelissa [Koskimies 2010, s. 21-22] todetaan, ettei UML-standardi selkeästi erottele eri

kaaviotyyppejä toisistaan. Lisäksi standardissa [OMG 2009b, s. 686] huomautetaan, että kaaviotyyppien rajat ovat joustavia. Tästä johtuen kaaviotyypit ovat enemmänkin tiettyjä vakiintuneita tapoja käyttää UML:a kuin täsmällisesti määriteltyjä osakieliä. UML:n 2.2-version 14 kaaviotyyppiä on kuvattu luvuissa 4 ja 5.

Artikkelissa [Koskimies 2010, s. 30] todetaan, että UML:ssa rakenteiden merkitystä ei ole tarkasti määritelty. Kirjassa [Fowler, s. 14] puolestaan mainitaan, ettei UML-mallin perusteella voida esimerkiksi yksikäsitteisesti sanoa, miltä mallia vastaava jollain tietyllä ohjelmointikielellä kirjoitettu ohjelmakoodi näyttää. **Mallin merkitys on tulkittava suhteessa mallinnettavaan kohteeseen ja tilanteeseen, johon se liittyy.** Esimerkiksi samaa kaaviotyyppiä voidaan käyttää sekä mallinnettavaan kohteeseen liittyvien käsitteiden että sovelluksen luokkien kuvaamiseen. Näin määrittely ei rajoita UML:n käyttöä sovelluskehityksen eri vaiheissa tai eri sovellusalueilla. Lisäksi sitä voidaan hyödyntää silloinkin, kun mallinnettavan järjestelmän kaikkia ominaisuuksia ja käsitteitä ei mallia laadittaessa tunneta.

UML-standardi määrittelee kielen rakenteen notaation ja metamallin (engl. *metamodel*) avulla. **Notaatio** on mallinnuskielen syntaksi, joka kuvaa malleissa käytettävän graafisen esitystavan. **Metamalli** kuvaa mallien sallitut abstraktit rakenteet ja määrittelee joukon nk. **hyvinmuodostuneisuussääntöjä** (engl. *well-formedness rules*), jotka edelleen rajoittavat sallittuja UML-malleja.

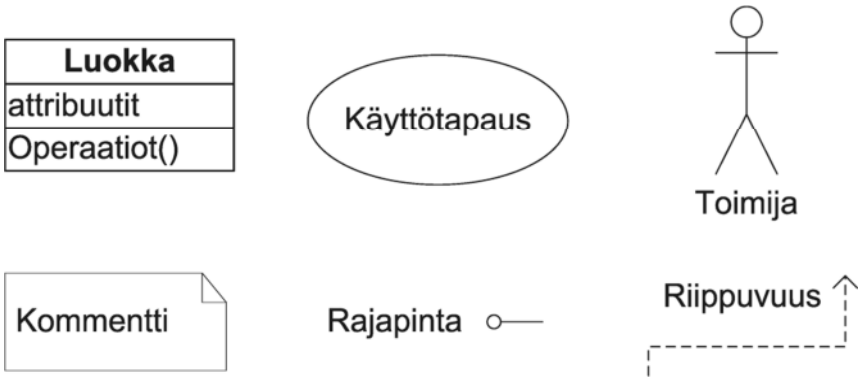
UML-malli koostuu **kaavioista** (engl. *diagram*), joilla kuvataan järjestelmän korkean tason toiminnallisuutta, staattista ja dynaamista rakennetta sekä toiminnan aikaista käyttäytymistä. Kukin kaavio noudattaa kaaviotyyppinsä määrittelemää rakennetta. Kaaviotyypit on esitelty luvuissa 4 ja 5.

3.2 Mallinnuselementit

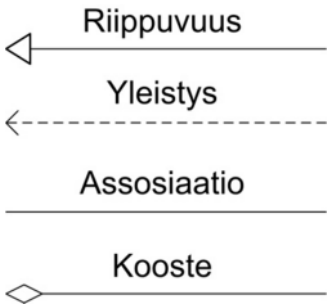
Standardissa [OMG 2009b, s. 683] todetaan UML:n kaavioiden koostuvan mallinnuselementeistä. Ne vastaavat yleisiä olio-ohjelmoinnin käsitteitä, kuten **luokka** (engl. *class*) ja **olio** (engl. *entity*) sekä niiden välisiä suhteita, kuten **assosiaatio** (engl. *association*) ja **yleistys** (engl. *generalization*) eli **periytyminen** (engl. *inheritance*). Samaa mallinnuselementtiä

voidaan käyttää useissa kaaviotyypeissä, mutta sen merkitys ja ulkoasu pysyvät muuttumattomina.

Jokaisella mallinnuselementillä on sitä vastaava graafinen symboli, jota käytetään kaavioissa. **Näkymäelementit** ovat geometrisiä kuvioita (kuten suorakaiteita), ja niiden väliset suhteet kuvataan erityyppisinä suhdeviivoina. Kuva 3.1 esittää muutamia mallinnuselementtejä ja kuva 3.2 niiden välisten suhteiden kuvaamiseen käytettyjä näkymäelementtejä Microsoft Office Visio -sovelluksella piirrettyinä.



Kuva 3.1:Yleisiä näkymäelementtejä.

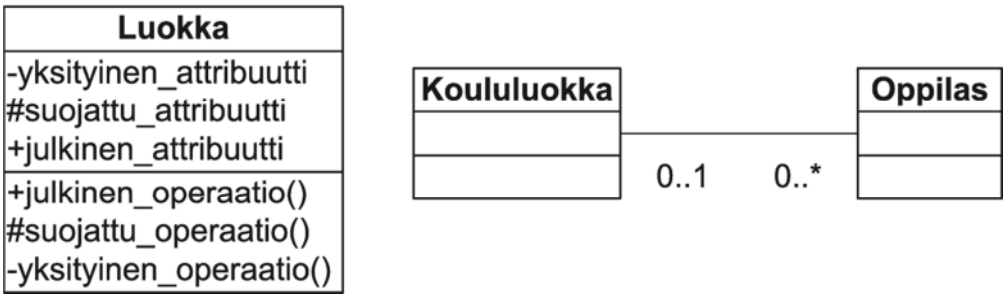


Kuva 3.2: Mallinnuselementtien suhteita kuvaavia näkymäelementtejä.

Kirjassa [Koskimies 2000, s. 126-127] huomautetaan, ettei UML tarkasti määrittele näkymäelementtien ja suhdeviivojen ulkoasua, vaan asia jää työkaluohjelmistojen valmistajien päätettäväksi. Ohjelmisto voi myös jättää näyttämättä mallista tiettyjä asioita käyttäjän pyynnöstä tai automaattisesti. Näin samasta mallista voi olla useita eri tarkoitukseen sopivia erilaisia näkymiä.

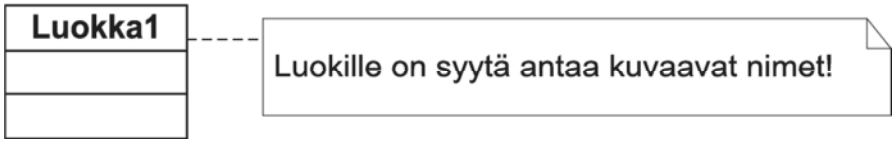
3.3 Lisäinformaation liittäminen mallinnuselementteihin

Yhteisillä merkinnöillä voi kaavioon liittää sellaista lisäinformaatiota, joka ei kuulu mallinnuselementtien perusominaisuuksiin. Kirjassa [Booch, s. 27-28] todetaan, että kaikkiin mallinnuselementteihin voidaan liittää **koristeita** (engl. *adornments*), jotka lisäävät näkymäelementtiin uutta informaatiota. Niiden avulla voidaan esimerkiksi luokkaa kuvaavassa näkymäelementissä ilmaista attribuuttien ja metodien näkyvyys tai liittää kerrannaisuuden määrittely assosiaatioon. Kuvan 3.3 vasemmassa kuviossa on koristeita käyttäen `Luokka`-luokkaan lisätty tieto attribuuttien ja operaatioiden näkyvyydestä, sekä oikealla on liitetty `Koululuokka`- ja `Oppilas`-luokkien väliseen assosiaatioon kerrannaisuutta kuvaavat koristeet.



Kuva 3.3: Esimerkki koristeiden käytöstä.

Huomautuksen (engl. *note*) avulla voidaan malliin lisätä tietoa, jota siinä ei voida muuten esittää. Se voidaan sijoittaa minne tahansa missä tahansa kaaviossa, ja se voi sisältää mitä tahansa tietoa, kuten kommentin tai viittauksen erilliseen dokumenttiin. Huomautus liitetään yleensä kuvan 3.4 tapaan katkoviivalla kyseiseen näkymäelementtiin.

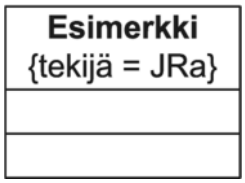


Kuva 3.4: `Luokka1`-näkymäelementtiin on liitetty huomautus.

Mallinnuselementtien **ominaisuuksien** (engl. *properties*) avulla voidaan esittää elementin tietoja. Ominaisuus voi olla esimerkiksi luokan tarkoituksen ja toiminnot kuvaava lyhyt vapaamuotoinen teksti. Käyttäjä voi lisäksi liittää mallinnuselementteihin tarvitsemiaan

lisätietomääreitä (nimetty arvo, engl. *tagged value*). Näin mallinnuselementtiin voidaan kiinnittää esimerkiksi koodigeneraattorin tai prosessiohjauksen tarvitsemia tietoja.

Ominaisuudet eivät tavallisesti näy kaaviossa, mutta ne on usein mahdollista saada esiin ohjelmistoissa esimerkiksi kaksoisnapsauttamalla näkymäelementtiä. Jos ominaisuudet ovat kaaviossa näkyvissä, ne merkitään kuvassa 3.5 esitetyllä tavalla aaltosulkuihin {} näkymäelementin nimen alapuolelle.



Kuva 3.5: Esimerkki luokkaan liitetystä lisätietomääreestä.

3.4 UML:n laajentaminen ja sovittaminen

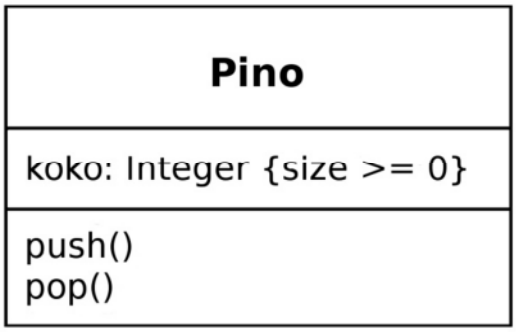
UML on yleiskäyttöinen mallinnuskieli. Kirjassa [Booch, s. 29] todetaan, että joissain tilanteissa se ei kuitenkaan sellaisenaan sovi käytettäväksi, vaan sitä on sovitettava esimerkiksi tiettyyn menetelmään tai tietylle organisaatiolle sopivaksi. Näitä tarpeita varten UML:ssa on joukko mekanismeja, joiden avulla sovittaminen voidaan tehdä muuttamatta itse mallinnuskieltä.

Laajentamiseen ja sovittamiseen voidaan käyttää lisätietomääreitä, rajoitteita (engl. *constraint*) ja stereotyyppejä (engl. *stereotype*). Lisätietomääreitä käsiteltiin luvussa 3.3 mallinnuselementtien ominaisuuksien yhteydessä.

Stereotyypit ovat mallinnuselementtien laajennuksia. Stereotyyppi laajentaa perustana olevan mallinnuselementin merkitystä, muttei muuta sen rakennetta. Kaaviossa stereotyyppi voidaan esittää omalla symbolillaan tai sijoittamalla näkymäelementissä sen nimi kaksoiskulmasuluissa «» elementin nimen eteen tai yläpuolelle, kuten luvun 4.4 kuvassa 4.6. UML-standardi [OMG 2009b, s. 695] määrittelee joukon valmiita käytössä vakiintuneita stereotyyppejä, joiden lisäksi on mahdollista määritellä omia stereotyyppejä.

Rajoite rajaa joko mallinnuselementin käyttöä tai sen merkitystä. Niiden avulla mallin laatijaa voidaan ohjata käyttämään mallinnuselementtiä tarkoitetulla tavalla. Kaavioissa

rajoitukset näytetään aaltosulkujen sisällä, kuten kuvassa 3.6. UML-standardissa on määritelty joukko valmiita rajoituksia, joiden lisäksi on mahdollista määrittää omia rajoituksia.



Kuva 3.6: Esimerkki luokan attribuuttiin liitetystä rajoitteesta.

Yhteenkuuluvat, esimerkiksi tiettyyn liiketoiminta-alueeseen liittyvät, UML:n metamallin laajennukset voidaan paketoida **profiiliksi** (engl. *profile*). Tällöin suunnittelija voi ottaa profiiliin sisältyvät laajennukset käyttöönsä yhtenä kokonaisuutena.

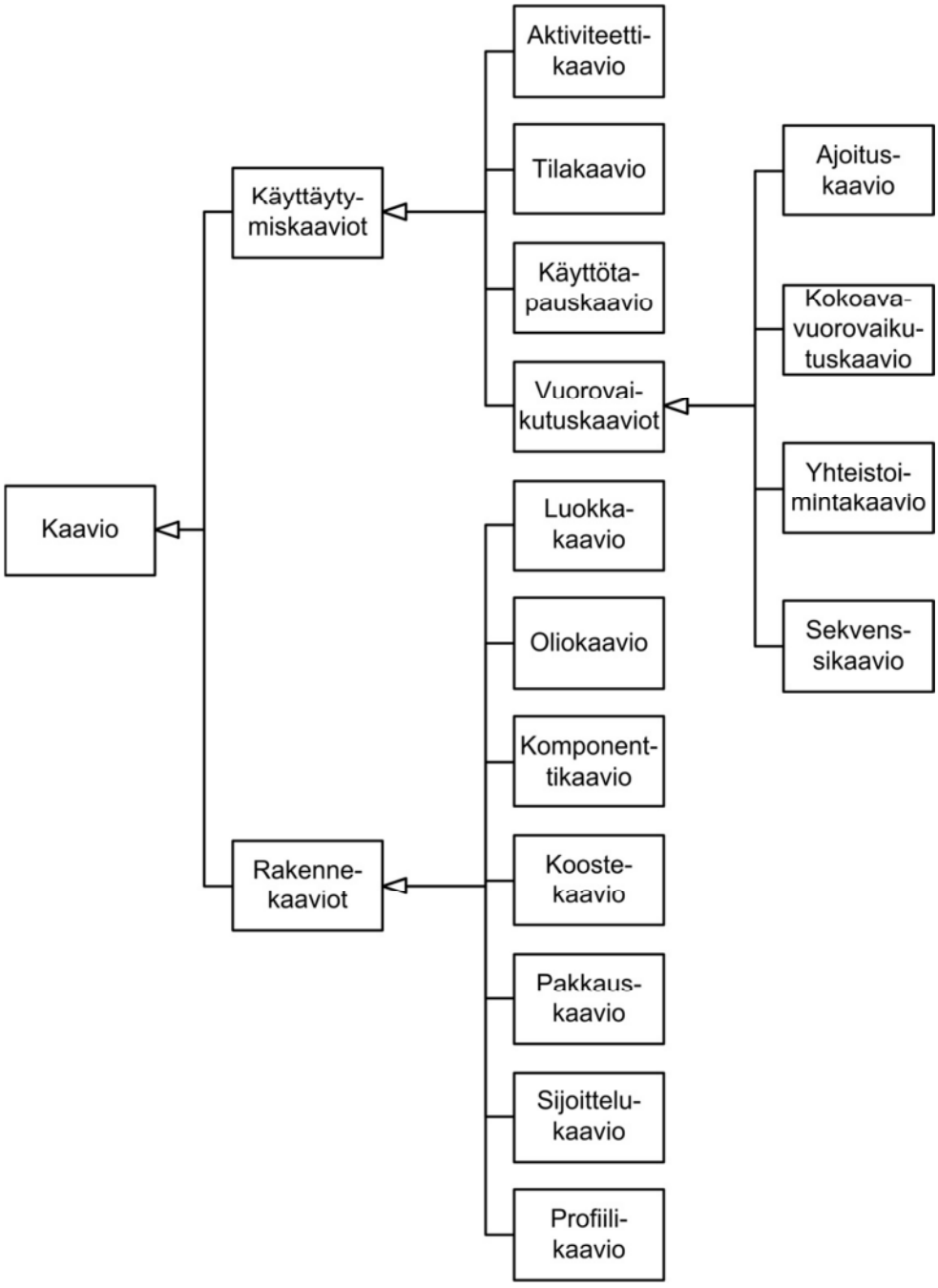
3.5 Kaaviotyypit

Tarkasteltavaa järjestelmää kuvaava malli esitetään UML:ssa kaavioiden avulla. Kaaviot ovat kuvioita, joissa mallinnuselementit on järjestetty kuvaamaan järjestelmää tietystä näkökulmasta. UML:n versiossa 2 on 13 kaaviotyyppiä, joita yhdistelemällä järjestelmän kokonaiskuva muodostetaan. UML-standardin [OMG 2009b, s. 686] versioon 2.2 on lisätty 14. kaaviotyypiksi profiilikaavio.

Kirjassa [Sommerville, s. 329] todetaan, että järjestelmää mallinnettaessa on yleensä huomioitava kaksi toisiaan täydentävää ja toisistaan riippuvaa näkökulmaa, jotka ovat järjestelmän rakenne ja sen toiminnan aikainen käyttäytyminen. UML-kaaviotyypit voidaan jakaa näiden näkökulmien perusteella rakennekaavioihin ja käyttäytymiskaavioihin. **Rakennekaaviot** kuvaavat järjestelmän staattista, ajasta riippumatonta, rakennetta ja **käyttäytymiskaaviot** järjestelmän ajonaikaista, ajasta riippuvaa, rakennetta. Käyttäytymiskaavioista voidaan edelleen erottaa vuorovaikutusta kuvaavat kaaviotyypit omaksi ryhmäkseen.

Kuvassa 3.7 on kuvattu OMG:n UML-standardin Superstructure-määrittelyssä [OMG 2009b, s. 686] esitetty jaottelu. Standardissa huomautetaan, että kaaviotyyppien rajat eivät

ole joustamattomia, vaan usein on sallittua käyttää kaaviossa toisen kaaviotyypin mallinnuselementtejä. Myös rakenne- ja käyttäytymiskaavioiden merkintöjä voidaan tarvittaessa käyttää samassa kaaviossa.



Kuva 3.7: UML:n kaaviotyyppien luokittelu.

4 Rakennekaaviot

Rakennekaaviot kuvaavat mallinnettavan järjestelmän staattista, ajasta riippumatonta, rakennetta. Kaaviotyypeistä luokka-, olio-, komponentti- ja koostekaavio liittyvät läheisesti toisiinsa, joten ne on esitelty luvussa 4.1. Kolme muuta rakennekaaviotyyppiä esitellään luvuissa 4.2-4.4. Luku perustuu pääasiassa standardiin [OMG 2009b, s. 140-214] sekä kirjoihin [Booch, s. 105-115, 195-201, 393-417] ja [Fowler, s. 35-52, 87-98, 135-142].

4.1 Luokka-, olio-, komponentti- ja koostekaaviot

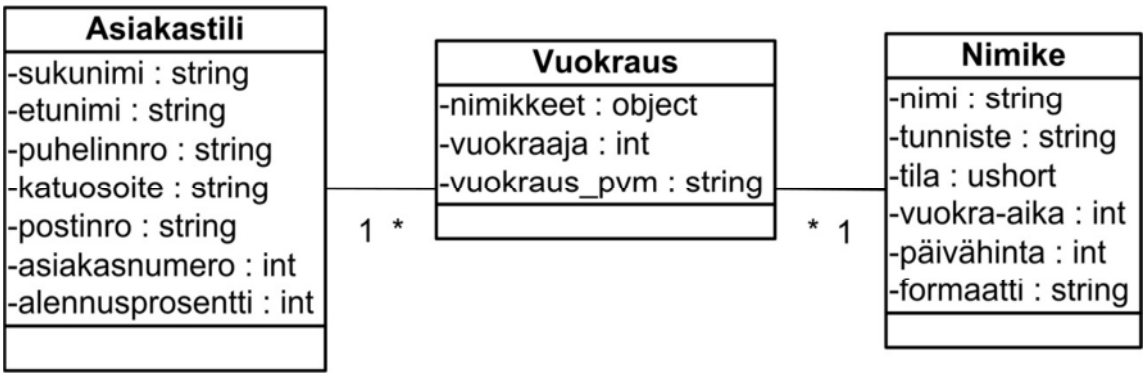
Artikkelissa [Koskimies 2010, s. 23] todetaan **luokkakaavion** (engl. *class diagram*) olevan tärkein oliomallinnuksessa käytetyistä rakennekaaviosta. Se kuvaa järjestelmän staattista rakennetta luokkien ja niiden välisten yhteyksien avulla. Kaaviossa voidaan esittää tiettyjä toiminnallisia elementtejä (kuten luokan toimintoja), mutta niiden dynaaminen toiminta kuvataan muiden kaavioiden avulla. Luokkakaaviota voidaan hyödyntää ohjelmiston luokkien esittämisen lisäksi myös mallinnettavan sovellusalueen käsitteistön kuvaamisessa.

Kirjan [Fowler, s. 35] mukaan luokka esitetään kaaviossa luvun 3.2 mukaisesti suorakaiteena, jonka sisällä ovat luokan nimi, luokan attribuutit ja metodien otsaketiedot. Luokan ominaisuuksista ja toiminnoista käytetään UML-terminologiassa yleisnimitystä **piirre** (engl. *feature*). Luokkien väliset yhteydet esitetään luvussa 3.2 määriteltyjen suhteita kuvaavien näkymäelementtien avulla. Kuvassa 3.7 on luokkakaavion merkinnöin esitetty UML:n kaaviotyyppien luokittelu, sekä luvun 7 esimerkkiin liittyvät luokkakaaviot on esitetty kuvissa 4.1, 7.1 ja 7.2.

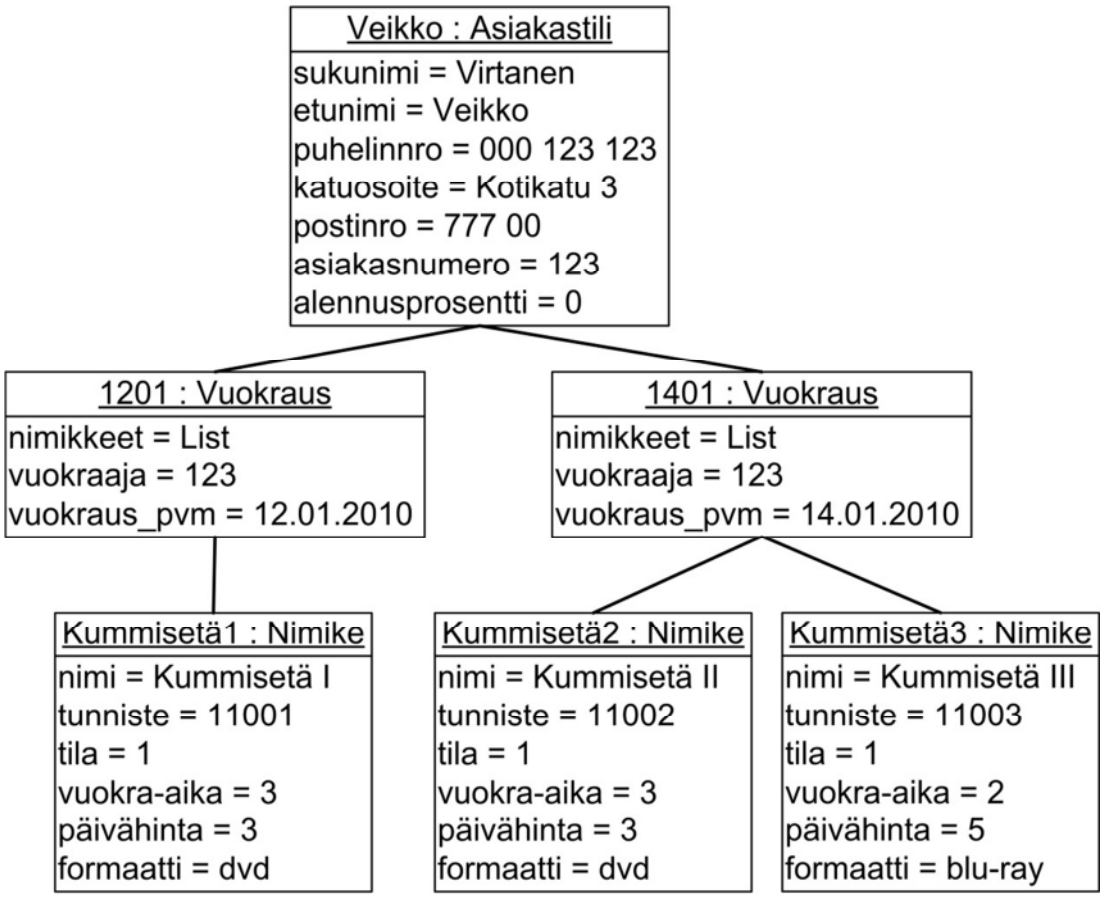
Kirjassa [Fowler, s. 87] todetaan, että **oliokaavio** (engl. *object diagram*) on kuva järjestelmästä tietyllä ajan hetkellä. Kaaviota kutsutaan oliokaavioksi, koska siihen sisältyy olioiden kuvauksia. Kaaviossa ei ole tarpeen kuvata olioita täydellisinä, vaan osa tiedoista voidaan kuvata epätäydellisesti esimerkiksi antamalla raja-arvot, joiden välillä ominaisuuden arvo voi vaihdella. Tarkan määrittelyn sijasta oliokaavion järjestelmästä antama kuva toimii esimerkkinä sen toiminnasta, jolloin oliokaavion avulla voidaan muun muassa havainnollistaa monimutkaisia tietorakenteita.

Oliokaaviossa käytetään samoja merkintätapoja ja suhteita kuin luokkakaaviossa. Olio kuvataan luokkamallinnuselementillä, jossa olion nimi on alleviivattu. Vaihtoehtoisesti voidaan käyttää luokan nimeä alleviivattuna tai olion ja luokan nimen yhdistelmää.

Kuvassa 4.1 on esitetty luvun 7 esimerkkiin liittyvä luokkakaavio ja kuvassa 4.2 sen perusteella laadittu oliokaavio. Luokkakaaviossa on kuvattu kolme luokkaa: `Asiakastili`, `Vuokraus` ja `Nimike`. Kaavion perusteella `Asiakastili`in voi liittyä useita `Vuokraus`-luokkia, joista kuhunkin voi liittyä useita `Nimike`-luokkia. Oliokaaviossa havainnollistetaan luokkakaavion luokkien välisiä yhteyksiä esimerkin avulla. Attribuuteille on asetettu arvot, esimerkiksi `Veikko:Asiakastili`-olion `sukunimi`-attribuutin arvo on `Virtanen`.

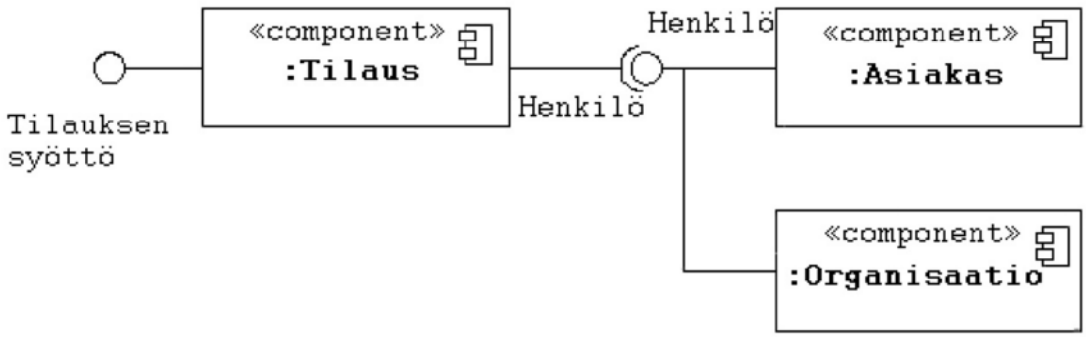


Kuva 4.1: Luokkakaavio.



Kuva 4.2: Kuvan 4.1 luokkakaaviota vastaava oliokaavio.

Komponentti on itsenäinen ohjelmayksikkö, joka toteuttaa tarkasti määritellyn rajapinnan. **Komponenttikaavio** (engl. *component diagram*) esittää järjestelmän komponentit ja niiden väliset yhteydet. Kirjan [Fowler, s. 141] mukaan sitä voidaan käyttää havainnollistamaan joko järjestelmän jakoa komponentteihin ja niiden vuorovaikutusta rajapintojen kautta tai haluttaessa paloitella komponentteja alemman tason rakenteiksi. Komponenttikaavion ja yleisen luokkakaavion raja ei ole tarkasti määritelty, mutta luokkakaaviosta poiketen komponenttikaaviossa kuvataan tyypillisesti arkkitehtuuritason rakenteita. Kuvassa 4.3 on UML:n version 2 näkymäelementein piirretty komponenttikaavio.



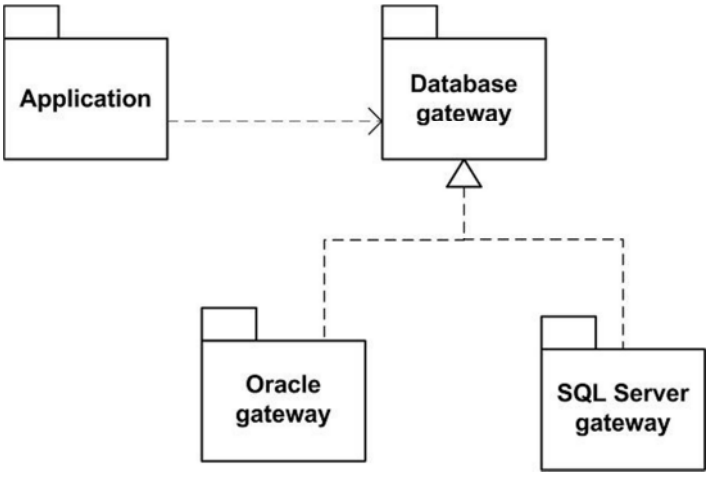
Kuva 4.3: Komponenttikaavio.

Koostekaavioiksi (engl. *composite structure diagram*) kutsutaan artikkelin [Koskimies 2010, s. 24] mukaan luokkakaaviota, jossa luokkien sisään on piirretty olioita tai komponentteja. Sen avulla voidaan esittää näiden osien suoritusaikainen ryhmittely.

4.2 Pakkauskaavio

Pakkaus (engl. *package*) on rakenne, jonka avulla mitä tahansa UML:n elementtejä voidaan ryhmitellä korkeamman tason yksiköiksi. **Pakkauskaavio** (engl. *package diagram*) on rakennekaavio, joka koostuu pääasiassa pakkauksista ja niiden välisistä yhteyksistä. Kirjan [Fowler, s. 89] mukaan pakkauskaaviota käytetään yleisimmin luokkien ryhmittelyyn, mutta sen avulla voidaan ryhmitellä myös muita mallinnuselementtejä suuremmiksi kokonaisuuksiksi.

Kuvassa 4.4 on esimerkki pakkauskaaviosta, jossa on kuvattu sovelluksen tietokantaliittymän toteutus korkealla tasolla. Kaavion perusteella nähdään, että `Database gateway` -pakkauksen toiminnot toteuttaa valitusta tietokannasta riippuen joko `Oracle gateway` tai `SQL Server gateway` -pakkaus.

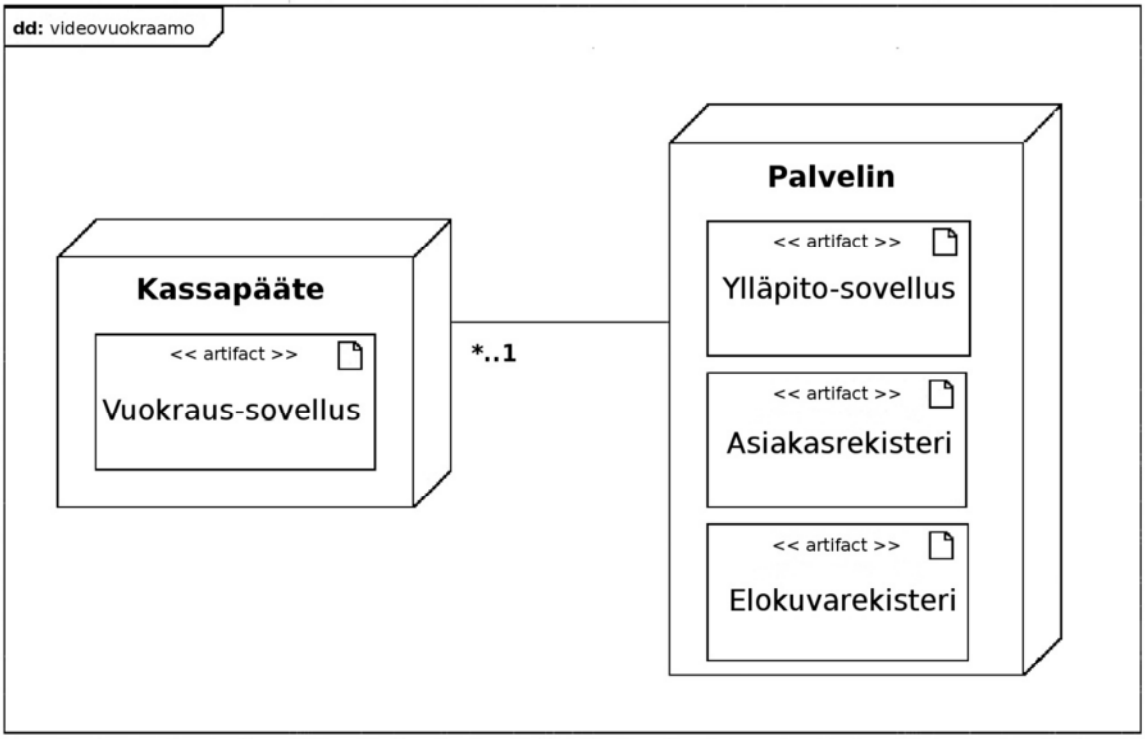


Kuva 4.4: Esimerkki pakkauskaaviosta.

4.3 Sijoittelukaavio

Sijoittelukaavio (engl. *deployment diagram*) tunnetaan myös nimellä **käyttöönottokaavio**. Se kuvaa järjestelmän laitearkkitehtuurin ja ilmaisee, miten ohjelmiston osien suoritus jakautuu laitteiden välillä.

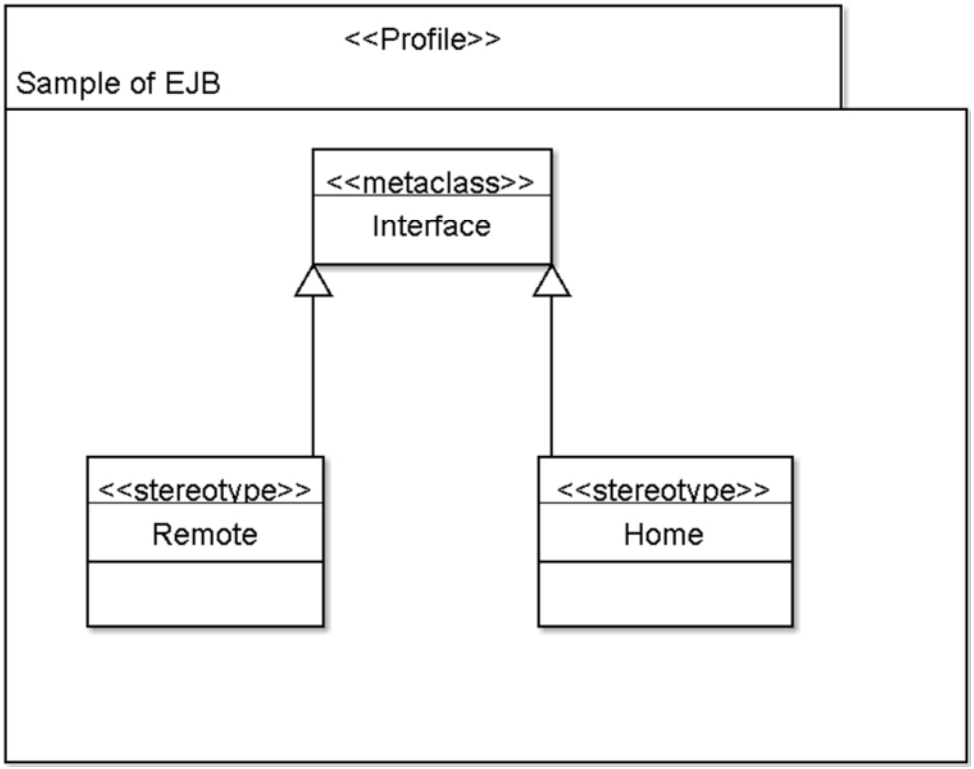
Kuvassa 4.5 on luvun 7 videovuokraamojärjestelmään liittyvä sijoittelukaavio Poseidon For UML -ohjelmistolla piirrettynä. Kaavio esittää sovelluksen osien jakautumisen eri laitteille. Laitteiston muodostavat palvelin ja yksi tai useampi kassapääte.



Kuva 4.5: Videovuokraamojärjestelmään liittyvä sijoittelukaavio.

4.4 Profiilikaavio

Profiilikaavio (engl. *profile diagram*) on määritelty UML-standardin versiossa 2.0 rakennekaavioksi, jossa kuvataan luvussa 3.4 esiteltyjä profiileja, stereotyyppejä ja niihin liittyviä metaluokkia. Se on lisätty kaavioiden jaottelua kuvaavaan liitteeseen versiossa 2.2. Kuvassa 4.6 on havainnollistettu profiilikaaviossa käytettäviä merkintätapoja.



Kuva 4.6: Yksinkertainen profiilikaavio.

5 Käyttäytymiskaaviot

Käyttäytymiskaavioita käytetään järjestelmän ajonaikaisen, ajasta riippuvan, rakenteen kuvaamiseen. Käyttäytymiskaavioista voidaan edelleen erottaa vuorovaikutusta kuvaavat kaaviot omaksi ryhmäkseen. Luku perustuu pääasiassa lähteisiin [OMG 2009b, s. 215-605] ja [Fowler, s. 53-62, 99-133, 143-150].

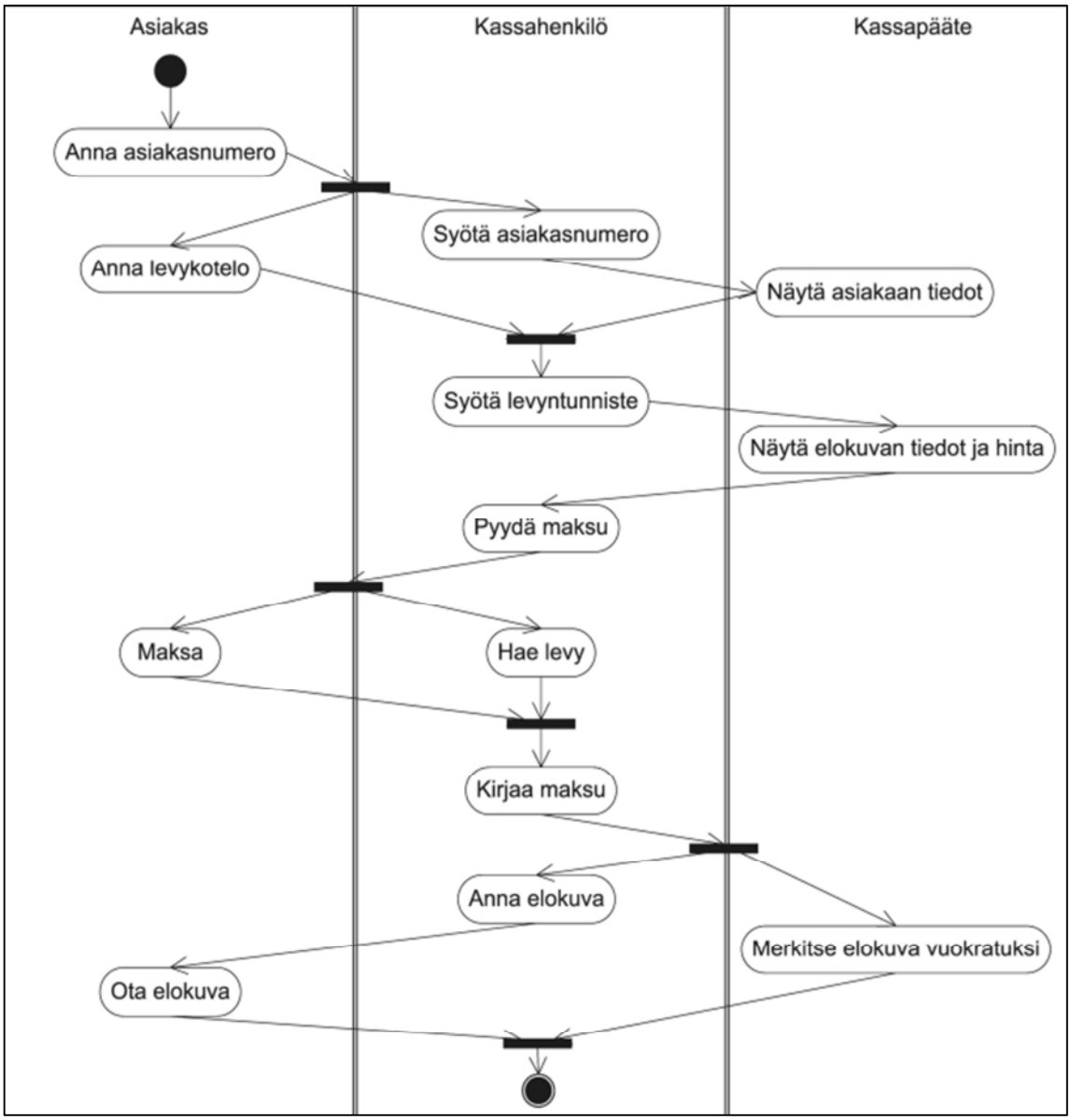
5.1 Aktiviteettikaavio

Kirjan [Fowler, s. 117] mukaan aktiviteettikaavion (engl. *activity diagram*) eli **toiminto-kaavion** avulla voidaan kuvata proseduurin toimintaa, liiketoimintaprosessia tai työn kulkua. Kaavio koostuu aktiviteeteista ja niiden välisistä siirtymistä. Aktiviteetti kuvataan kulmista pyöristettynä suorakaiteena kuvan 5.1 tapaan. Kaaviossa voidaan esittää myös rinnakkaisia tai vaihtoehtoisesti suoritettavia aktiviteetteja ja olioita.

Useissa UML-versioissa aktiviteettikaavioon on tehty merkittäviä muutoksia. Myös UML-versiossa 2.0 sitä on muutettu ja laajennettu merkittävästi. Kirjassa [Fowler, s. 117] nostetaan muutoksista esille aktiviteettikaavion ja luvussa 5.2 esiteltävän tilakaavion välisen sidoksen poistaminen. Aiemmissa UML:n versiossa aktiviteettikaavion katsottiin nimittäin olevan tilakaavion erikoistapaus. Yhteys tilakaavioon on koettu ongelmalliseksi etenkin, kun aktiviteettikaaviota on käytetty työn kulun kuvaamiseen.

Aktiviteettikaaviossa voidaan osoittaa aktiviteettien suorittajat jakamalla kaavio vyöhykkeisiin (engl. *partition* tai *swim lane*). UML-versiossa 1.x vyöhykkeet jakoivat kaavion pysty- tai vaakasuunnassa. UML:n versiossa 2.x voidaan käyttää myös kaksiulotteista ruudukkojakoa.

Kuvassa 5.1 on luvun 7 esimerkkiin liittyvä aktiviteettikaavio Microsoft Office Visio -sovelluksella piirrettynä. Kaaviossa esitetään käyttötapauksen `Levyn vuokraaminen` eteneminen. Kaavio on jaettu vyöhykkeisiin, jotka erottavat toimijat `Asiakas` ja `Kassa-henkilö` sekä `Kassapääte`-järjestelmän aktiviteetit toisistaan.



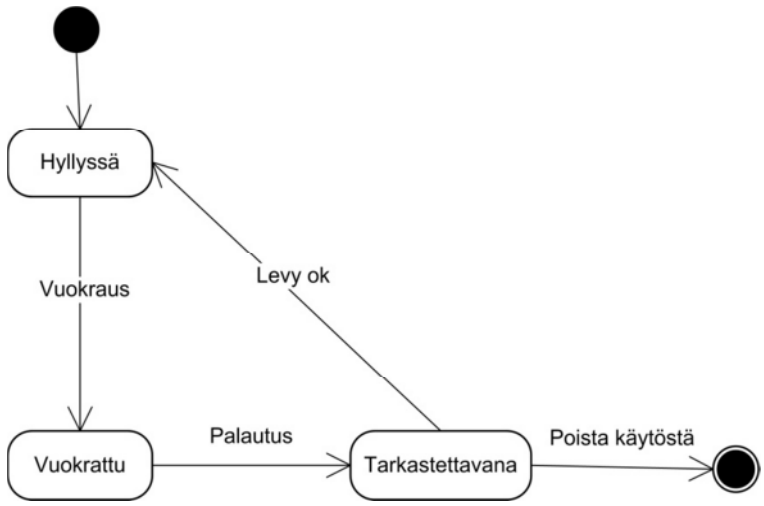
Kuva 5.1: Käyttötapauksen `Levyn vuokraaminen` aktiviteettikaavio.

5.2 Tilakaavio

Tilakaavio (engl. *state diagram* tai *state machine diagram*) kuvaa kirjan [Fowler, s. 107] mukaan oliosuuntautuneessa lähestymistavassa yhden olion suoritusaikaisen toiminnan tilakoneen avulla. Kaavio koostuu **tiloista** (engl. *state*), niiden välisistä siirtymistä ja koostetiloista. Tila kuvataan kaaviossa kulmistaan pyöristettynä suorakaiteena.

Oliosuuntautuneessa lähestymistavassa tilakaaviolla kuvataan tyypillisesti luokan yhden olion toiminta. Kaavio esittää kaikki oliolle mahdolliset tilat ja siirtymisen tilojen välillä aiheuttavat **tapahtumat** (engl. *event*).

Kuvassa 5.2 on luvun 7 esimerkkiin liittyvä tilakaavio Microsoft Office Visio -sovelluksella piirrettynä. Kaaviossa esitetään `Levy`-luokan olion tilan muuttuminen, kun levy vuokrataan, palautetaan, tarkastetaan palautuksen jälkeen ja siirretään hyllyyn tai poistetaan käytöstä tarkastuksen tuloksen perusteella.



Kuva 5.2: `Levy`-luokan olion tilan muuttumista kuvaava tilakaavio.

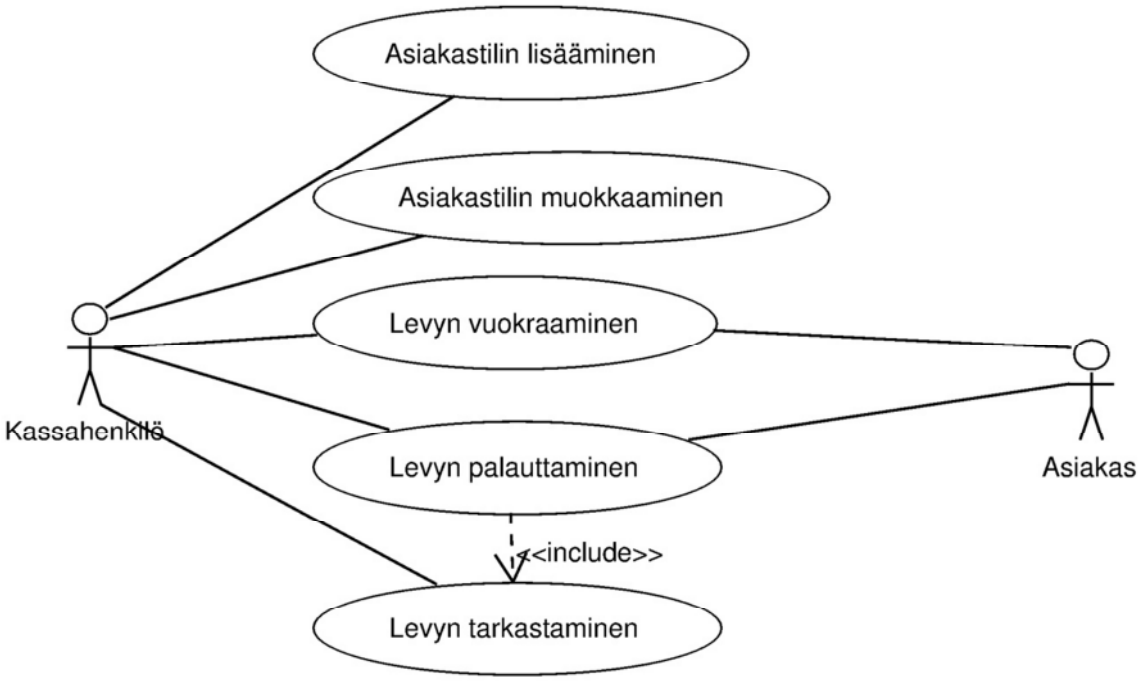
5.3 Käyttötapauskaavio

Kirjan [Fowler, s. 102] mukaan käyttötapauskaavion (engl. *use case diagram*) avulla listataan järjestelmän **käyttötapaukset** (engl. *use case*), niihin liittyvät toimijat (engl. *actor*) ja näiden suhteet. Käyttötapausten avulla voidaan kartoittaa järjestelmän toiminnalliset vaatimukset. Kukin käyttötapaus kuvaa järjestelmän ja sitä käyttävän toimijan välistä vuorovaikutusta tietyn tuloksen saavuttamiseksi. Käyttötapaus esitetään kaaviossa ellipsinä kuvan 5.3 tapaan.

Toimija voi olla ihminen tai toinen järjestelmä, joka käyttää kuvattavan järjestelmän palveluita. Toimijaa kuvaa kaaviossa tikku-ukkoa muistuttava näkymäelementti. Sama fyysinen käyttäjä voi esiintyä mallissa useampana kuin yhtenä toimijana. Kirjassa [Fowler, s. 100] todetaankin, että englanninkielisen termin *actor* sijasta termi *role* olisi kuvaavampi.

UML-standardi ei millään tavoin määritä tapaa, jolla käyttötapausten sisältö tulisi kuvata. Koska käyttötapausten keskeisin anti on niiden sisällön kuvaus, pelkästä kaaviosta saatava hyöty on varsin rajoitettu. Käyttötapausten toiminta kuvataan toiminnallisten näkymien avulla. Usein toiminnan kuvaamiseen käytetään luvussa 5.4 kuvattavaa vuorovaikutuskaavioita.

Kuvassa 5.3 on luvun 7 esimerkkiin liittyvä käyttötapauskaavio, jossa on kuvattu mallinnettavan järjestelmän käyttötapaukset ja niihin liittyvät toimijat.



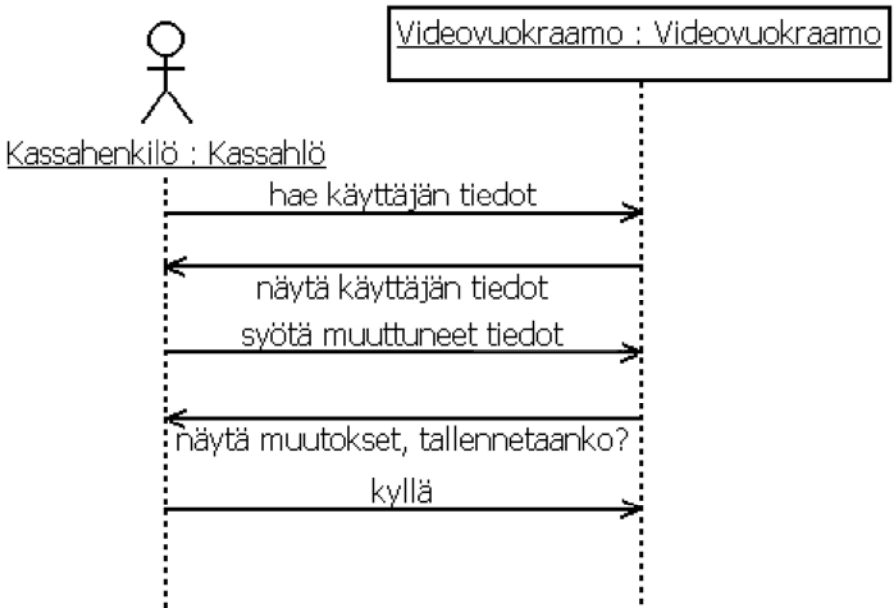
Kuva 5.3: Videovuokraamon kassapäätteen käyttötapauskaavio.

5.4 Vuorovaikutuskaaviot

Sekvenssikaavio (engl. *sequence diagram*) tunnetaan myös nimellä **viestiyhteyskaavio**. Se kuvaa artikkelin [Koskimies 2010, s. 27] ja kirjan [Fowler, s. 53] mukaan olioiden keskinäistä vuorovaikutusta ajan kuluessa ja niiden välillä kulkevia viestejä. Useimmista muista kaaviotyypeistä poiketen sekvenssikaavio ei ole verkkomainen esitys, vaan osallistujat kuvataan aikaulottuvuuden suuntaisina viivoina kuvan 5.4 tapaan. UML:n versiossa 2.0 sekvenssikaavion ilmaisuvoimaa on parannettu lisäämällä siihen esimerkiksi rakenteiset haarautumisilmaisut.

Sekvenssikaavioita käytetään usein, kun käyttötapauksiin halutaan liittää lisäinformaatiota. Tällöin sekvenssikaavio kuvaa, kuinka käyttötapaukseen liittyvät toimijat ovat vuorovaikutuksessa järjestelmän kanssa.

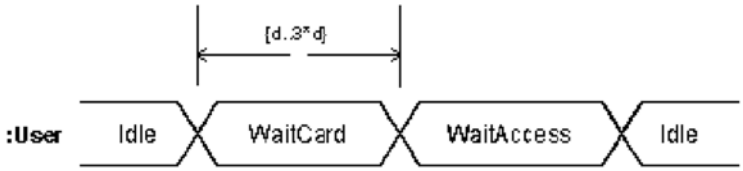
Kuvassa 5.4 on luvun 7 esimerkkiin liittyvä sekvenssikaavio StarUML-sovelluksella piirrettynä. Se kuvaa `Kassahenkilö`-toimijan ja `Videovuokraamo`-järjestelmän välistä vuorovaikutusta `Asiakastilin muokkaaminen` -käyttötapauksessa.



Kuva 5.4: `Asiakastilin muokkaaminen` -sekvenssikaavio.

Kokoava vuorovaikutuskaavio (engl. *interaction overview diagram*) on aktiviteettikaavion muunnos, jossa erilliset sekvenssikaaviot yhdistetään toisiinsa luvussa 5.1 esitellyn aktiviteettikaavion rakentein.

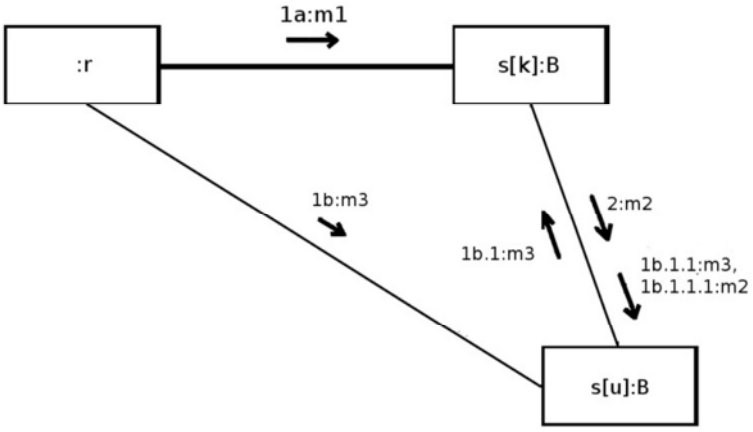
Ajoituskaavio (engl. *timing diagram*) on UML:n versioon 2.0 lisätty kaaviotyyppi, jota on ajateltu käytettävän sekvenssikaavioiden sijasta reaaliaikaisten sovellusten mallintamiseen. Se kuvaa yhden tai useamman olion käyttäytymisen ajan suhteen. Kuvassa 5.5 on standardin [OMG 2009b, s. 524] esimerkki ajoituskaaviosta, jossa on kuvattu `user`-olion tilan muuttuminen ajan kuluessa. Kuvan merkintä *{d.3*d}* ilmaisee tilaan liittyvän aikarajoituksen, jossa merkintä *d* vastaa yhden signaalin kestoa. Kyseinen aikarajoitus on siis väli yhden signaalin kestosta kolmen signaalin kestoon.



Kuva 5.5: Esimerkki ajoituskaaviosta.

Yhteistoimintakaaviosta (engl. *communication diagram*) käytetään joissain lähteissä myös nimeä **kommunikointikaavio**. Se kuvaa kirjan [Fowler, s. 131] mukaan vuorovaikutustilannetta painottaen vuorovaikutuksen osapuolten välisiä viestiyhteyksiä. Sekvenssikaaviosta poiketen yhteistoimintakaaviossa ei ajan kulumista esitetä erillisenä dimensiona, vaan viestien järjestys ilmaistaan järjestysnumeroin kuvan 5.6 tapaan. Yhteistoimintakaaviosta käytettiin UML-versiossa 1.x nimeä **yhteistyökaavio** (engl. *collaboration diagram*).

Kuvassa 5.6 on standardin [OMG 2009b, s. 517] esimerkki yhteistoimintakaaviosta. Viestit `m1` ja `m3` lähtevät samanaikaisesti luokan `r` ilmentymästä luokan `B` kahteen ilmentymään `s[k]` ja `s[u]`. UML-standardia noudattavan monitasoisen järjestysnumeroinnin mukaan esimerkissä viesti `1b.1` seuraa viestiä `1b` ja `1b.1.1` seuraa `1b.1`:tä.



Kuva 5.6: Esimerkki yhteistoimintakaaviosta.

6 UML:n hyödyntäminen ohjelmistojen kehittämisessä

UML ei ole sidoksissa mihinkään tiettyyn suunnittelumenetelmään, vaan sitä voidaan hyödyntää erilaisiin suunnittelumenetelmiin liittyvien ohjelmistokehitysprosessien yhteydessä. Sitoutumattomuudesta johtuen, UML-standardi ei ohjaa tapaa, jolla UML:a tulisi hyödyntää. Käytännössä UML:n hyödyntämistä eri tilanteissa ohjaavat sovitut tai vakiintuneet tavat.

Luvuissa 4 ja 5 esiteltyjen kaaviotyyppien suhdetta ohjelmistokehitysprosessimalleissa esiintyviin toimintoihin tarkastellaan luvuissa 6.1 - 6.5. Luvussa 6.6 esitellään UML:n erilaisia käyttötapoja ja niiden vaikutusta mallin laatimiseen käytettävän työkalun valintaan. Luku 6.7 käsittelee UML:n erilaisten käyttötapojen luokittelua. Luku perustuu pääosin lähteisiin [Fowler, s. 2, 13, 29-32] ja [Sommerville, s. 74-82].

6.1 UML:n suhde ohjelmistokehitysprosesseihin

Mallinnuskielet itsessään eivät välttämättä ota kantaa ohjelmistojen kehittämisessä käytettävään menetelmään tai prosessiin, vaan valittu **suunnittelumenetelmä** ja sovitut käytänteet määrittävät, kuinka mallinnuskieliä sovelletaan. Usein suunnittelumenetelmä kuvaa prosessin, jonka eri vaiheissa mallinnuskieliä käytetään menetelmän määrittämällä tavalla.

UML pyrkii standardin [OMG 2009a, s. 9] mukaan olemaan yhteensopiva useimpien olemassa olevien olio-suuntautuneiden ohjelmistokehitysprosessien kanssa. UML on siis sidoksissa olioparadigmaan, mutta ei mihinkään tiettyyn suunnittelumenetelmään. Artikkelissa [Koskimies 2010, s. 21] todetaan, ettei sidos oliolähestymistapaan estä käyttämästä UML:a muidenkin lähestymistapojen yhteydessä.

Vaikka **UML on riippumaton käytettävästä prosessimallista**, vaikuttaa valittu suunnittelumenetelmä UML:n käyttötapaan. UML:a suunniteltaessa on sitä ajateltu käytettävän käyttötapauksiin pohjautuvassa inkrementaalisessa kehitysprosessissa. UML:n yhteydessä mainitaan usein Unified Process -niminen prosessimalli.

Kaikkiin tarkoituksiin sopivaa ideaalista ohjelmistokehitysprosessia on tuskin mahdollista kehittää. Käytössä onkin useita toisistaan poikkeavia suunnittelumenetelmiä ja niihin liit-

tyviä prosessimalleja. Niissä esiintyy kuitenkin toisiaan vastaavia toimintoja. Kirjassa [Sommerville, s. 74] esitetään neljä ohjelmistokehitysprosessimalleille yhteistä toimintoa:

1. ohjelmiston määrittely,
2. ohjelmiston suunnittelu,
3. ohjelmiston toteutus ja validointi, sekä
4. ohjelmiston evoluutio.

UML:n hyödyntämistä kyseisissä toiminnoissa kuvataan luvuissa 6.2 - 6.5.

6.2 UML:n hyödyntäminen määrittelyssä

Ohjelmiston määrittelyyn sisältyvät kirjan [Sommerville, s. 75] mukaan esitutkimus, vaatimusten kartoitus ja analysointi, vaatimusten täsmentäminen sekä vaatimusten validointi. Määrittelyn tavoitteena on saavuttaa selkeä käsitys siitä, mitä järjestelmää ollaan toteuttamassa.

Esitutkimuksessa (engl. *feasibility study*) selvitetään, kannattaako järjestelmää ylipäätään toteuttaa. Tutkimukseen ei liity erityistä olionäkökulmaa, eikä UML-kaaviotyyppiä.

Vaatimusten kartoituksen ja analyysin (engl. *requirements elicitation and analysis*) aikana kerätään vaatimukset, mahdollisesti mallinnetaan järjestelmää löydettyjen vaatimusten perusteella ja johdetaan malleista uusia vaatimuksia. **Vaatimusten täsmennys** (engl. *requirements specification*) on työvaihe, jossa kerätty informaatio muunnetaan vaatimuslistoiksi ja määrittelykuvauksiksi. **Vaatimusten validoinnin** (engl. *requirements validation*) tehtävä on varmistaa, että löydetyt vaatimukset todella määrittelevät asiakkaan haluaman järjestelmän.

UML:a hyödynnettäessä määrittelyn tuloksena on joukko luvussa 5.3 esiteltyjen käyttötapauksien kuvauksia ja järjestelmän käsitemalli luokkakaaviona (katso luku 4.1). Käyttötapausten pohjalta voidaan laatia luvussa 5.4 kuvattuja sekvenssikaavioita, joiden perusteella saadaan täsmennettyä järjestelmältä odotettavat yksittäiset palvelut.

6.3 UML:n hyödyntäminen suunnittelussa

Suunnittelutoiminnossa määrittelyn tuloksena saatua mallia laajennetaan ja sitä muutetaan vastaamaan toteutuksessa käytettävän teknisen alustan vaatimuksia. Määrittelyssä laadittavaa luokkakaaviota täydennetään toteutukseen liittyvillä luokilla. Suunnittelutoimintoon sisältyvät kirjan [Sommerville, s. 77] mukaan arkkitehtuurisuunnittelu, abstrakti määrittely, sekä rajapinta-, komponentti-, tietorakenne- ja algoritmisuunnittelu. Jokaisessa vaiheessa täsmennetään edellisen vaiheen tuloksia ja lopputuloksena saadaan toteutettavien algoritmien ja tietorakenteiden yksityiskohtainen määrittely.

Arkkitehtuurisuunnittelussa (engl. *architectural design*) tunnistetaan ja dokumentoidaan järjestelmään toteutettavat osajärjestelmät, eli itsenäiset toisistaan riippumattomat kokonaisuudet ja niiden väliset suhteet. Arkkitehtuurisuunnittelua varten on kehitetty erityisesti sitä tukevia kuvauskieliä, mutta arkkitehtuuria voidaan kuvata myös luvun 4 rakennekaavioiden avulla.

Abstraktin määrittelyn (engl. *abstract specification*) tuloksena saadaan arkkitehtuurisuunnittelussa löydettyjen alijärjestelmien palvelut ja toiminnan rajat määrittelevä kuvaus. Alijärjestelmä voidaan UML-kaaviossa esittää pakkausmallinnuselementillä ja alijärjestelmien suhteet luvussa 4.2 esiteltyä pakkauskaaviota käyttäen.

Rajapintasuunnittelussa (engl. *interface design*) suunnitellaan ja dokumentoidaan kunkin alijärjestelmän muille alijärjestelmille tarjoamat rajapinnat. Osajärjestelmän rajapinnan tulee olla sellainen, että osajärjestelmää voidaan käyttää ilman tietoa sen sisäisestä toteutuksesta. Dokumentoinnissa voidaan käyttää luvun 4.1 komponenttikaaviota.

Komponenttisuunnittelussa (engl. *component design*) osajärjestelmän palvelut ryhmitellään palvelut toteuttaviin komponentteihin ja komponenttien rajapinnat kuvataan. Komponenttien ja niiden rajapintojen esittämistä varten UML:ssa on käytössä luvun 4.1 komponenttikaavio.

Kun järjestelmä on jaettu osajärjestelmiin ja edelleen komponentteihin, **tietorakennesuunnittelussa** (engl. *data structure design*) järjestelmän toteutuksessa tarvittavat tietorakenteet suunnitellaan ja kuvataan yksityiskohtaisesti. Kuvaus voidaan esittää luvun 4.1

luokkakaaviona, jota havainnollistetaan tarvittaessa luvussa 4.1 esitellyn oliokaavion avulla.

Algoritmisuunnittelussa (engl. *algorithm design*) suunnitellaan ja kuvataan yksityiskohtaisesti ne palveluiden toteuttamisessa tarvittavat algoritmit, joiden toiminta ei riittävästi selviä muusta kuvauksesta. Algoritmien kuvaamiseen voidaan käyttää tarpeen mukaan useaa UML-kaaviotyyppiä, kuten luvussa 5.1 esiteltyä aktiviteettikaaviota tai luvun 5.4 sekvenssikaaviota.

6.4 UML:n hyödyntäminen ohjelmiston toteutuksessa ja validoinnissa

Toteutuksessa suunnittelun tuloksena saadut mallit ohjelmoidaan valitulla ohjelmointikielellä, sekä tehdään malliin käytännön toteutuksen vaatimat lisäykset ja ohjelmointiympäristön mahdollisesti vaatimat täsmennykset. Esimerkiksi luvun 4.1 luokkakaaviossa kuvattava luokkien välistä yhteenkuuluvuutta ilmaiseva koostesuhde toteutetaan eri tavoin eri ohjelmointikielissä.

Kirjassa [Sommerville, s. 80] todetaan, että ohjelmistoa **validoitaessa** kehitettyä järjestelmää arvioidaan yleensä yksikkö-, integrointi-, järjestelmä- ja hyväksymistesteillä. Eri tasoilla käytetään testauksen pohjana UML-mallin eri kaavioita. Yksikkötesteissä testauksen pohjana ovat luvun 4.1 luokkakaaviot ja -määritykset, integrointitesteissä tavallisesti luvun 4.1 komponenttikaaviot ja luvun 5.4 yhteistoimintakaaviot sekä järjestelmätesteissä luvun 5.3 käyttötapauskaaviot.

6.5 UML:n hyödyntäminen ohjelmiston evoluutiossa

Toteutuksen jälkeen ohjelmistoa on ylläpidettävä sen elinkaaren ajan. Kirjassa [Sommerville, s. 82] käytetään termiä **evoluutio** ylläpito-toiminnon sijasta, koska entistä useammin ylläpito liittyy saumattomasti järjestelmän kehittämiseen, eikä tiukka erottelu näiden välillä vastaa todellisuutta.

UML:n näkökulmasta siirtyminen perinteisestä ylläpidosta kohti jatkuvaa evoluutiota on haaste dokumentaationa käytettyjen mallien ylläpidolle. Kirjassa [Fowler, s. 31-32] esitetään, että jokaista sovelluksen yksityiskohtaa ei kannata mallintaa dokumentaatioon. Sen

sijaan mallin avulla voidaan havainnollistaa suunnittelun kannalta järjestelmän tärkeimpiä osia. Esimerkiksi luvun 4.2 pakkauskaavion avulla voidaan dokumentaatiossa antaa yleiskuva järjestelmän jakaantumisesta alijärjestelmiin.

UML-mallinnusta voidaan käyttää apuna myös selvitettäessä jälkikäteen ylläpidettävän järjestelmän rakennetta. Järjestelmän keskeisistä osista voidaan joko laatia UML-malli rakennetta tutkittaessa tai malli voidaan generoida automaattisesti ohjelmakoodista jotain työkalua käyttäen.

6.6 UML:n hyödyntäminen yrityksissä

UML:n käyttötapoja ei ole standardissa rajattu, joten sitä voidaan hyödyntää eri tavoin tarpeesta ja käyttäjistä riippuen. UML:a voidaan käyttää ohjelmistojen kuvaamisen lisäksi esimerkiksi myös jonkin tarkasteltavan aihealueen käsitteiden kuvaamiseen.

Artikkelissa [Koskimies 2010, s. 21] todetaan, että UML-standardin laajuuden ja monimutkaisuuden vuoksi yritykset käyttävät tavallisesti UML:sta vain itselleen tarpeellisia osajoukkoja. Kirjassa [Fowler, s. 13] huomautetaan, että UML-kaavion tulkitseminen vaatii standardin ymmärtämisen lisäksi mallin laatijoiden omaksumien erilaisten, niin yleisten kuin projektikohtaistenkin, käytäntöjen tuntemista.

Kirjassa [Haikala, s. 364] esitetään tutkimuksiin viittaamatta havainto, että **suomalaisissa yrityksissä** on omaksuttu luvun 4.1 luokkakaavioiden käyttö ja lisäksi hyödynnetään luvun 5.4 sekvenssikaaviota. Muiden kaavioiden käytön arvioidaan käytännössä olevan vähäisempää.

Kirjassa [Booch, s. 7] mainitaan useiden tutkimusten (joita ei kuitenkaan ole yksilöity) viittaavan siihen, että useimmat ohjelmistoja kehittävät yritykset käyttävät muodollisia mallinnusmenetelmiä vain vähäisessä määrin tai ei lainkaan. **Mallin tarkkuus ja laajuus** kehotetaankin valitsemaan suhteessa projektin monimutkaisuuteen. Kirjassa [Koskimies 2000, s. 125] puolestaan esitetään suositus, ettei UML:a kannata pyrkiä soveltamaan kokonaisuudessaan. Sen sijaan kehitysympäristön tarpeet huomioiden tulisi valita UML:sta vain ne osat, jotka kyseiset tarpeet parhaiten täyttävät.

UML koostuu useista erilaisista kaaviotyypeistä, joista tapauskohtaisesti voidaan valita kulloiseenkin tarkoitukseen parhaiten soveltuvat kaaviot. Toisaalta on syytä huomata, että UML ei suinkaan sisällä kaikkia käyttökelpoisia kaaviotyyppejä. Kirjassa [Fowler, s. 15 ja 62] mainitaan kolme UML:en kuulumatonta kaaviotyyppiä: päätöstaulu, CRC-kortit ja sovelluksen näyttöjen välillä siirtymistä kuvaava epämuodollinen kaavio. Näistä päätöstaulu voidaan yksinkertaisissa tapauksissa korvata UML:en kuuluvalla luvun 5.1 aktiviteettikaaviolla, mutta monimutkaisemmissa tapauksissa päätöstaulu on sekä tiiviimpi että selkeämpi.

6.7 UML:n käyttötapojen luokitteluja

Kirjassa [Fowler, s. 2] esitetään kolme UML:n toisistaan eroavaa käyttötapaa: luonnostelu, yksityiskohtainen suunnittelu ja käyttö ohjelmointikielenä. Artikkelissa [Koskimies 2010, s. 35-39] käyttötapoja erotellaan sen mukaan, onko kyseessä ihmisten välinen vai ihmisen ja koneen välinen kommunikointi. Luonnostelu kuuluu tämän jaottelun mukaan ihmisten väliseen kommunikointiin. Ihmisen ja koneen välistä kommunikaatiota ovat käyttö ohjelmointikielenä, sekä yksityiskohtaisen suunnittelu, jonka tavoitteena on tuottaa tietokoneelle laadittavien mallien pohjana käytettäviä kaavioita. Lisäksi artikkelissa mainitaan dokumentointi ihmisten välisen kommunikaation muotona.

7 Esimerkki UML:n käytöstä

Tarkastellaan UML-mallin laatimista esimerkin avulla. Esimerkissä luodaan malli yksinkertaistetusta videovuokraamon kassapäätejärjestelmästä, jonka vapaamuotoinen sanallinen kuvaus on esitetty luvussa 7.1. Luvuissa 7.2 - 7.5 esitellään mallin laatimisen vaiheet, sekä luvussa 7.6 pohditaan mallissa hyödynnettyjen kaaviotyyppien valintaa ja tarpeellisuutta mallin kannalta.

Kuvissa 4.5, 5.1, 5.2, 5.3, 5.4, 7.1 ja 7.2 on esitetty videovuokraamon tietojärjestelmän kassapäätteen toteutukseen liittyvän UML-mallin muodostavat kaaviot. Lisäksi kuvassa 4.1 on vaihtoehtoinen esitys kuvan 7.2 luokkakaaviosta ja kuvassa 4.2 siihen liittyvä oliokaavio. Kukin kaavio kuvaa järjestelmää omasta näkökulmastaan korostaen jotain malliin sisältyvää informaatiota.

7.1 Järjestelmän vapaamuotoinen kuvaus

Malli perustuu toteutettavasta järjestelmästä ja sen toiminnoista työn tilaajan kanssa luotuun kuvaukseen. **Esimerkkijärjestelmän vapaamuotoinen kuvaus** on seuraava:

- Kassapääte on yhteydessä vuokraamon palvelimeen, ja sen avulla palvelimella sijaitsevaan tietokantaan syötetään tieto asiakkaiden vuokraamista elokuvista.
- Asiakas valitsee elokuvan tuomalla hyllystä levykotelon kassahenkilölle.
- Kassahenkilö syöttää päätteelle asiakasnumeron ja levykotelossa olevan tunnisteen. Kassahenkilö voi tarvittaessa myös muokata asiakastilin tietoja.
- Jos asiakkaalla ei ole asiakasnumeroa, on hänelle luotava asiakastili ennen levyn vuokraamista. Asiakastiliin kirjataan asiakkaan yhteystiedot.
- Pääte ilmoittaa vuokrahinnan ja -ajan. Vuokrahinnan laskennassa huomioidaan asiakkaalle mahdollisesti määritelty kampanjakohtainen alennusprosentti.
- Kassahenkilö noutaa levyn varastosta ja antaa kotelon levyineen asiakkaalle tämän maksettua vuokrahinnan. Levy voi olla DVD- tai Blu-ray-formaatissa.
- Asiakas palauttaa levyn vuokra-ajan päätyttyä, ja kassahenkilö kirjaa sen palautetuksi.

- Palautuksen jälkeen levy tarkastetaan. Tarkastuksen tuloksen perusteella levy joko palautetaan hyllyyn tai poistetaan käytöstä.

7.2 Järjestelmän käyttötapaukset

Järjestelmän käyttötapauksia kuvaava kaavio on esitetty luvun 5.3 kuvassa 5.3. Toimijoiksi kuvauksen perusteella tunnistetaan `Kassahenkilö` ja `Asiakas`. Käyttötapauksia ovat `levyn vuokraaminen`, `levyn palauttaminen`, `levyn tarkastaminen`, `asiakastilin lisääminen` ja `asiakastilin muokkaaminen`. Sanallisen kuvauksen perusteella voidaan päätellä, että on mahdollisesti olemassa myös kassapäätteen ja palvelimen väliseen toimintaan liittyviä käyttötapauksia. Kuvaus rajataan kuitenkin asiakkaan ja kassahenkilön näkökulmaan ja muut käyttötapaukset käsitellään tarvittaessa erikseen.

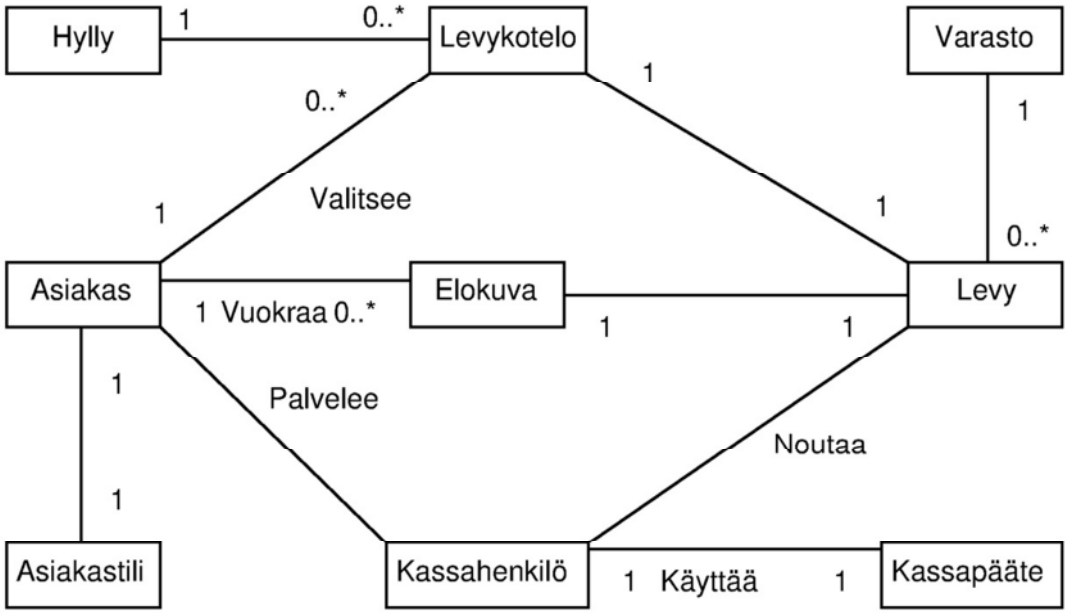
Käyttötapauskaavion lisäksi kukin käyttötapaus tulee kuvata myös joko sanallisesti tai muiden kaavioiden avulla. `Levyn vuokraaminen` -käyttötapaus on kuvattu aktiviteettikaaviona luvun 5.1 kuvassa 5.1.

7.3 Arkkitehtuurin kuvaus

Kassapäätteeseen ei esimerkissä liity arkkitehtuurisuunnittelun näkökulmasta erityisiä mallin kannalta oleellisia seikkoja, lukuun ottamatta sen yhteyttä videovuokraamon muuhun tietojärjestelmään. Luvun 4.3 kuvassa 4.5 on esitetty **kassapäätteen suhde kokonaisuuteen sijoittelukaavion avulla**. Palvelimen ja kassapäätteen suhdetta osoittavan assosiaation lukumääräsuhde osoittaa, että yhteen palvelimeen voidaan liittää yksi tai useampia kassapäätteitä. Sijoittelukaaviossa on kuvattu myös vuokraussovelluksen käyttämien asiakas- ja elokuvarekisterin sijoittuminen palvelimelle.

7.4 Määrittelytoiminnon käsitemalli

Luvussa 7.1 esitellyn järjestelmän sanallisen kuvauksen perusteella laadittu **käsitemalli** on esitetty yksinkertaistettuna luokkakaaviona (katso luku 4.1) kuvassa 7.1.

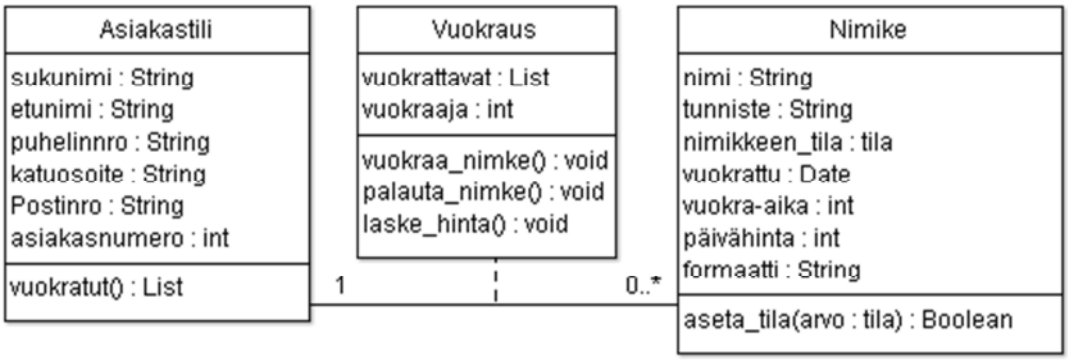


Kuva 7.1: Esimerkkijärjestelmään liittyvät käsitteet esitettynä luokkakaaviona.

7.5 Toteutusvaiheessa hyödynnettävä luokkakaavio

Käsitemallia kuvaavan luvun 7.4 luokkakaavion pohjalta laadittu **ohjelmiston toteutusvaiheessa käytettävä** luokkakaavio on esitetty kuvassa 7.2. Siinä näkökulma on teknisempi kuin käsitemallia kuvaavassa kuvan 7.1 kaaviossa. Luokkakaaviossa ei esitetä niitä käsitekaavion luokkia, joilla ei ole ohjelmiston kannalta merkitystä tai joiden sisältämä informaatio voidaan liittää attribuuttina johonkin toiseen luokkaan. Esimerkiksi luvun 5.3 kuvan 5.3 käyttötapauskaaviossa esiintyvää `Asiakas`-toimijaa ja käsitekaavion `Asiakas`-luokkaa vastaa kuvan 7.2 kaaviossa `Asiakastili`-luokka.

UML sallii kaavion sisältämän informaation tarkastelun eri tarkkuuksilla. Toteutusvaiheessa käytettävässä kuvan 7.2 luokkakaaviossa on kuvattu myös sovelluksessa käytettävät luokkien piirteet, kun taas kuvassa 7.1 on painotettu käsitteiden välisiä yhteyksiä, ja luokkien piirteet on piilotettu näkymäelementeistä.



Kuva 7.2: Esimerkkijärjestelmän toteutusta varten laadittu luokkakaavio.

Mallia voidaan tarvittaessa täydentää laatimalla luokkakaavion tueksi muita kaavioita, jotka kuvaavat järjestelmän osien toimintaa käytön aikana. Esimerkissä `Nimike`-luokalla on kolme mahdollista tilaa: `hyllyssä`, `vuokrattu` ja `tarkastettavana`. Tilojen välisten siirtymien havainnollistamiseksi piirretty tilakaavio on esitetty luvun 5.2 kuvassa 5.2.

7.6 Mallissa käytetyt kaaviotyypit

Kassapäätemalli koostuu kuvien 5.3, 5.4, 5.1, 7.1, 7.2 ja 4.5 käyttötapaus-, sekvenssi-, aktiviteetti-, luokka- ja sijoittelukaavioista. Käytettyjä kaaviotyyppejä olisi voitu karsia käyttämällä käyttötapausten kuvaamiseen johdonmukaisesti luvussa 5.1 esiteltyä aktiviteettitai luvussa 5.4 esiteltyä sekvenssikaaviota. Myös sijoittelukaavion tarpeellisuus näin yksinkertaisessa mallissa voidaan kyseenalaistaa. Toisaalta voidaan ajatella, että esimerkiksi luvussa 4.2 esitellyn pakkauskaavion laatiminen olisi saattanut olla tarpeen, mikäli se olisi selkeyttänyt mallia. Luvussa 5.4 esitellyn ajoituskaavion käyttämiselle mallissa on sen sijaan vaikea keksiä perusteluja.

8 Yhteenveto

UML on OMG:n standardoima laajasti tunnettu ja monipuolinen tietojärjestelmien mallinnuskieli, joka on kehitetty useiden sitä edeltäneiden mallinnuskielten pohjalta. Ensimmäinen julkinen versio UML:sta julkaistiin 1995, ja OMG hyväksyi standardiksi version 1.1 vuonna 1997. Viimeisin merkittäviä uudistuksia sisältänyt versio 2.0 julkaistiin vuonna 2003.

Tarkasteltavaa tietojärjestelmää kuvaava malli esitetään UML:ssa kaavioiden avulla. Standardin versiossa 2.2 määritellään seitsemän rakennetta kuvaavaa ja seitsemän järjestelmän käyttäytymistä kuvaavaa kaaviotyyppiä. Rakennekaavioista käytetyin on luokkakaavio ja käyttäytymiskaavioista hyödynnetään useimmin sekvenssikaaviota.

Kaaviot koostuvat mallinnuselementeistä ja niiden välisistä suhteista. Samaa mallinnuselementtiä voidaan käyttää useissa kaavioissa sen merkityksen ja ulkoasun pysyessä muuttumattomana. Jokaista mallinnuselementtiä vastaa kaavion graafisessa esityksessä näkymäelementti.

Edeltäjistään poiketen UML ei ole sidottu mihinkään tiettyyn prosessiin, eikä menetelmään, vaan sitä voidaan soveltaa mallinnettavan sovellusalueen ja menetelmän tarpeita vastaavalla tavalla. UML:a voidaan tarvittaessa myös muokata standardiin kuuluvien laajennusmekanismien avulla, jos se ei sellaisenaan sisällä kaikkia sovellusalueen ja menetelmän vaatimia käsitteitä.

Lähteet

Booch Grady, Rumbaugh James and Jacobson Ivar, "The Unified Modeling Language User Guide", Addison-Wesley, 1999.

Eriksson Hans-Erik and Penker Magnus, "UML", Oy Edita Ab, Jyväskylä, 2000.

Fowler Martin, "UML Distilled: A Brief Guide to the Standard Object Modeling Language", Third Edition, Addison-Wesley, 2004.

Haikala Ilkka ja Märijärvi Jukka, "Ohjelmistotuotanto", Talentum Media Oy, 2005.

Koskimies Kai, "Oliokirja", 2. muuttumaton painos, Satku – Kauppakaari, 2000.

Koskimies Kai, Koskinen Johannes, Maunumaa Mika, Peltonen Jari, Selonen Petri, Siikarla Mika ja Systä Tarja, *UML työvälineenä ja tutkimuskohteena*, saatavilla pdf-muodossa <URL: `http://www.tkts.fi/lehti/a21/uml.pdf`>, SYTYKE ry, viitattu 30.4.2010.

OMG, "OMG Unified Modeling Language (OMG UML), Superstructure 2.2", saatavilla pdf-muodossa <URL: `http://www.omg.org/spec/UML/2.2/Superstructure`>, February 2009.

OMG, "OMG Unified Modeling Language (OMG UML), Infrastructure 2.2", saatavilla pdf-muodossa <URL: `http://www.omg.org/spec/UML/2.2/Infrastructure`>, February 2009.

Sommerville Ian, "Software Engineering", Eighth Edition, Addison-Wesley, 2007.