Harri Laine

Johdatus sovellussuunnitteluun

osa 2

Helsingin yliopisto

Tietojenkäsittelytieteen laitos

Sisältö:

viimeinen sivu 39

4. Luokka- ja oliokaaviot

Luokkakaaviolla (class diagram) kuvataan

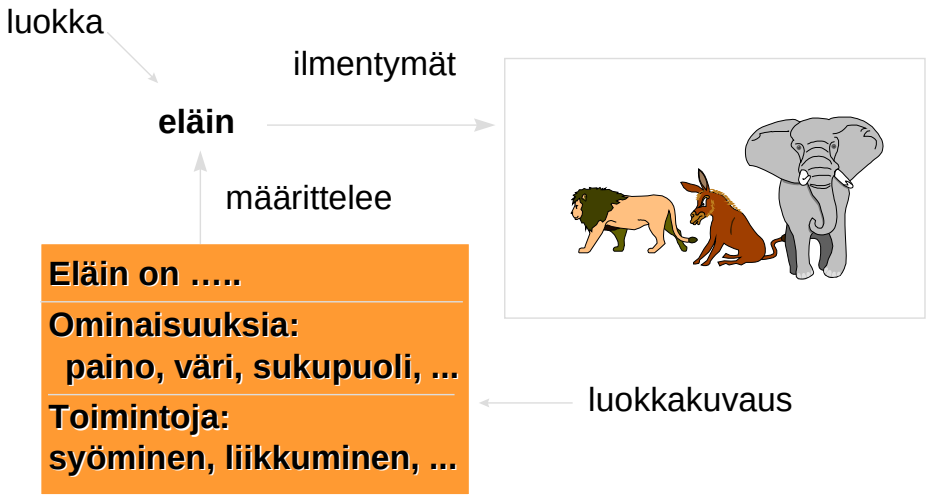
- ohjelmiston tai järjestelmän rakenne,
- olioiden tietosisältö,
- palvelut, joita oliot kykenevät suorittamaan sekä
- olioiden ja luokkien väliset yhteydet.

Oliokaaviolla (object diagram) havainnollistetaan luokkakaaviota, esimerkiksi olioiden välisiä yhteyksiä.

Oliolla (object) tarkoitetaan tiedon ja sen käsittelyyn liittyvien palveluiden muodostamaa kokonaisuutta. Olio on erotettavissa muista olioista. Sillä on identiteetti. Olio pitää sisällään tietoja, joita olion palvelut käsittelevät tai hyödyntävät. Puhdasoppisessa olio-ohjelmoinnissa olion tietoihin pääse käsiksi vain olion palveluiden avulla.

Tietojärjestelmää voidaan pitää oliona. Sillä on palveluja ja sillä on tietosisältö, joka tarkemmin tarkasteltuna jakautuu edelleen pienemmiksi olioiksi.

Oliomallintaminen perustuu olioiden luokitteluun. Jokainen olio kuuluu johonkin luokkaan ja on toiminnaltaan ja sisällöltään luokkakuvauksessa määritellyn mallin mukainen. Olioluokka (object class) on siis samankaltaisten olioden malli. Yksittäisiä olioita tarkastellaan tämän mallin ilmentyminä (instance) (Kuva 4.1).



Kuva 4.1: Luokka, luokkakuvaus ja luokan ilmentymät reaalimaailmassa

Luokkakuvaus esitetään jollakin kuvaustekniikalla joko graafisesti tai jäsenneltynä tekstiesityksenä. Olio-ohjelmoinnissa luokkakuvaus esitetään ohjelmointikielellä (kuva 4.2). Tämä esitys on tarkoitettu tietokoneelle. Lisäksi tarvitaan ihmisille tarkoitettu kuvaus. Kun järjestelmiä toteutettaessa edetään suunnittelusta toteutukseen on luonnollista, että ihmisille tarkoitetut kuvaukset eli suunnitelmat tuotetaan ensin ja vasta sitten koneelle tarkoitettu esitys.

```
public class elain {
   int elain_numero;
   String laji;
   Color vari;
   float paino;

   public elain(…){…}
   public setPaino(…){…}
   public float getPaino(){…}
   …
}

elain otus = new elain(….);
elain toinen = new elain(….);
```

ilmentymiä

Kuva 4.2: Luokkakuvaus ohjelmointikielellä

4.1 Luokkakuvaus

Luokkakuvauksessa nimetään luokka kuvaavalla nimellä. Tarvittaessa määritellään tekstikuvauksena luokan merkitys, eli selvitetään millaisia olioita luokkaan kuuluu. Luokkakuvauksessa määritellään myös luokan ilmentymiin sisältyvät tiedot ja palvelut (operaatiot), joita luokan ilmentymät kykenevät suorittamaan.

Luokan ilmentymiin sisältyvät tiedot kuvataan antamalla

- attribuutti (attribute), joka on tietoa kuvaava nimi
- arvomäärittely (value specification), joka esittää millaisten arvojen avulla tieto esitetään ja
- selvitys tiedon merkityksestä ja käytöstä .

Attribuutin arvona voi olla

- yksinkertainen arvo, esimerkiksi kokonaisluku
- rakenteinen arvo, esimerkiksi osoite, joka jakautuu postiosoitteeseen, postinumeroon ja lähiosoitteeseen
- kokoelma yksinkertaisia tai rakenteisia arvoja, esimerkiksi järjestämätön joukko värejä tai järjestetty joukko koordinaattipareja
    - kokoelmia voi olla rakenteeltaan erityyppisiä (taulukko, joukko, järjestetty joukko, lista, …)

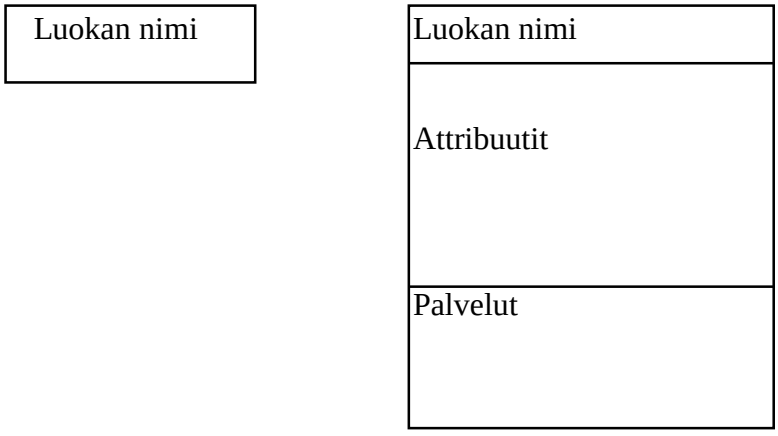
Attribuutin arvona voi olla myös viittaus olioon tai kokoelma viittauksia olioihin. Tällaista attribuuttia kutsutaan olioarvoiseksi. Olioarvoisen attribuutin olemassaolo tarkoittaa sitä, että on olemassa **yhteys** attribuutin sisältävän olion ja attribuutin arvona olevan olion välillä. Tällaiset yhteydet ovat usein hyvin oleellisia järjestelmän oliorakenteen ymmärtämisen kannalta. Niinpä niiden esittämiseen on olemassa oma tekniikkansa. Jotta samaa asiaa ei tarvitsi kuvata moneen kertaan jätetään olioarvoiset attribuutit tällöin pois. Olioarvoisia attribuutteja on kuitenkin syytä käyttää jos viitattu olio on perustietotyypin kaltainen. Tällaisia voisivat olla päiväys, kellonaika, henkilötunnus, väri, koordinaatti, jne.

Vaikka attribuuttinimen tulisi mahdollisimman hyvin kuvata sitä ilmiötä, joka attribuuttiin liittyvällä arvolla halutaan esittää, ei nimi yleensä riitä attribuutin määrit-

telyksi vaan tarvitaan erillinen selvitys attribuutin merkityksestä ja käytöstä. Selvityksessä voidaan käsitellä arvojen tulkintaa, arvojen tarkkuutta, arvojen mittaamista yms. Arvojen käyttöön ja käyttäytymiseen saattaa myös liittyä erityispiirteitä, joilla on merkitystä kehitettävälle ohjelmistolle. Eräs tällainen piirre on attribuutin käyttö olioiden ulkoiseen tunnistamiseen.. Olioilla on attribuuttien arvoista riippumaton identiteetti, mutta usein on tarve viitata olioon myös jonkin siitä tiedetyn ja attribuuttien avulla esitetyn ominaisuuden perusteella, esimerkiksi henkilö-oliot voitaisiin ulkoisesti tunnistaa nimen ja syntymäajan avulla.

Kiinteäarvoinen attribuutti on sellainen, jonka arvo säilyy samana koko olion eliniän, esimerkiksi henkilön syntymäaika on tällainen. Välttämätön attribuutti on sellainen, jonka arvo ei voi olla tuntematon tai puuttuva. Esimerkiksi henkilön nimi voisi olla välttämätön henkilön attribuutti, mutta osoite ei.

UML-luokkakaaviotekniikassa luokka kuvataan suorakaiteena. Se voidaan esittää suppeassa muodossa, jolloin suorakaiteen sisään kirjoitetaan vain luokan nimi, tai laveassa muodossa, jossa suorakaiteen sisällä näkyvät myös luokan attribuutit sekä mahdollisesti myös palvelut (kuva 4.3).



Kuva 4.3: Luokkasymboli suppeassa ja laveassa esitysmuodossa.

Attribuutista UML-luokkakaaviossa voidaan esittää

- Nimi (välttämätön)
- Tietotyyppi

Tietotyyppinä voi käyttää jonkin ohjelmointikielen mukaisia tietotyyppejä (int, char, real, boolean, jne). Tietotyypit voivat olla myös sovellusalue- tai tapauskohtaisia (rahasumma, prosenttiosuus, nimi, jne).

- Moniarvoisuus (multiplicity)

  Kokoelma-arvoisten attribuuttien kohdalla kokoelman koko voidaan ilmoittaa arvojen vähimmäis- ja enimmäismäärinä hakasulkeissa, esimerkiksi [0..5] tarkoittaa, että kokoelmassa on 0-5 arvoa, Tähti-merkkiä (*) voidaan käyttää tarkoittamaan 'monta, mutta täsmällistä ylärajaa ei voida antaa')

- Näkyvyys (visibility)

  Näkyvyys on eräissä olio-ohjelmointikielissä, esimerkiksi Java ja C++, tarjolla oleva tekniikka rajoittaa attribuutin (tai palvelun) käyttöä. Näissä kielissä on yleensä määritelty kolme näkyvyystyyppiä:

  - julkinen (public, UML:ssä '+') sallii attribuuttiin suoran käytön olion palvelujen ulkopuolelta
  - suojattu (protected, UML:ssä '#') sallii attribuutin käytön palveluissa, jotka on määritelty joko samassa luokkakuvauksessa kuin attribuutti tai tämän perivissä luokkakuvauksissa (perimistä tarkastellaan myöhemmin)
  - yksityinen (private, UML:ssä '-') sallii attribuutin käytön vain samassa luokkakuvauksessa määritellyissä palveluissa, jossa attribuuttikin on määritelty.

  Näkyvyyden määrittely tulee kyseeseen vain ohjelmointiläheisessä luokkakuvauksessa. Puhdasoppisen oliolähestymistavan mukaisesti näkyvyyden tulisi olla aina 'suojattu'.

- Oletusarvo
- Muita määreitä

  UML:ssä attribuutin arvo voidaan määritellä kiinteäarvoiseksi '*frozen*' tai muuttuvaksi '*changeable*'. Oletusarvoisesti attribuutin arvo on muuttuva. Kokoelma-arvoiseen attribuuttiin voidaan liittää määre '*addOnly*' estämään arvojen poiston kokoelmasta. UML ei tarjoa valmista määrittelytekniikkaa välttämättömille ja tunnistaville attribuuteille. Nämä ovat kuitenkin järjestelmää kuvattaessa vähintään yhtä tarpeellisia kuin pysyvyys ja muuttuvuus. UML sisältää kuitenkin laajennusmahdollisuuden, jota hyödyntämällä voimme päättää,

että välttämätön attribuutti ilmaistaan määreellä ‘*not null*’ ja tunnistava määreellä ‘*id*’.

UML:n attribuuttimäärittelyn rakenne on

```
[<näkyvyys>] <nimi> [<moniarvoisuus>] [: <tietotyyppi> ]
     [= <oletusarvo>] [ { <muut määreet> } ][1]
```

Esimerkkejä:

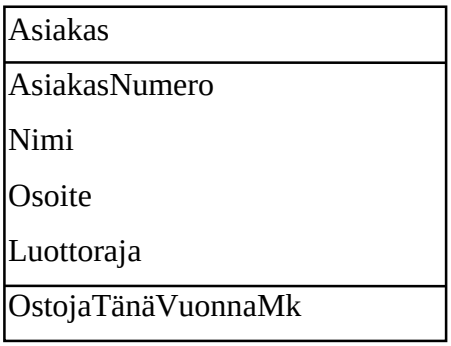
hetu : Henkilötunnus {frozed, id, not null}

Attribuutin hetu arvo on tyyppiä Henkilötunnus. Arvo on pysyvä ja välttämätön ja sitä käytetään olion ulkoiseen tunnistamiseen.

suosikkiruoka [*] : String {ordered}

Attribuutin suosikkiruoka arvona on järjestetty kokoelma merkkijonoja. ‘*ordered*’ on UML:n perusmääre, joka voidaan liittää kokoelmiin ja yhteyksiin. Siitä ei näe millä perusteella arvot on järjestetty, mutta voisimme vapaamuotoisessa tekstikuvauksessa täsmentää, että ruoat ovat kokoelmassa suosituimmuusjärjestyksessä.

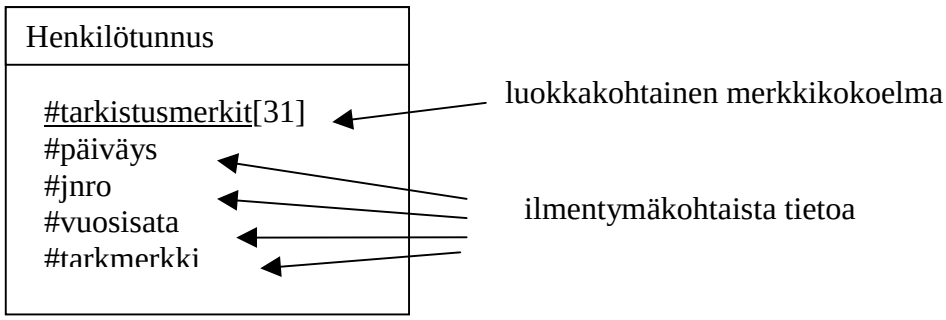
Kuvaesityksessä attribuuteista usein annetaan vain nimi. (Kuva 4.4) Tällöin tarvitaan kuvan täydennykseksi tekstimuotoinen määrittely. CASE-välineissä on yleensä mahdollisuus valita miten paljon yksityiskohtia luokkakuvauksessa näytetään.



Kuva 4.4: Asiakas -luokkaa kuvaava kaaviosymboli

[1] Hakasulut ilmaisevat valinnaisen elementin eli [a] tarkoittaa, että elementti a voi kuulua esitykseen tai puuttua siitä. Kulmasulut (<>) rajaavat syntaksisen elementin. Attribuutille ainoa välttämätön kuvailutieto on täten nimi.

Eräissä ohjelmointikielissä (esim. Java, C++) on mahdollista määritellä luokan ilmentymiin sisältyvien tietojen lisäksi luokkakohtaisia tietoja. UML:ssä luokkakohtaiset tiedot kuvataan alleviivaamalla attribuutin nimi (kuva 4.5).



Kuva 3.5: Luokka- ja ilmentymäkohtaiset attribuutit

Palvelun määrittely on UML:ssä muotoa

```
[<näkyvyys>] <nimi> [(<parametrit>)]
      [:<paluuarvon tyyppi>] [{<muut määreet>}]
```

Määrittelyn osat:

- Nimi (välttämätön)
- Parametrit

  Parametrin määrittely on muotoa:

  ```
  [<suunta>] <nimi>: <tyyppi> [= <oletusarvo> ]
  ```

  - Suunta ilmaisee onko kyseessä
    - syöteparametri (in), joka välittää tietoa palvelulle (jos parametrina on olio sen tila ei muutu)
    - tulosparametri (out), joka välittää tietoa palvelulta sen käyttäjälle (jos parametrina on olio sen tila voi muuttua)
    - yhdistetty syöte ja tulosparametri (inout), joka välittää tietoa kumpaankin suuntaan
- Näkyvyys

  Palvelujen näkyvyys voidaan määritellä samoin kuin attribuuttien näkyvyys.

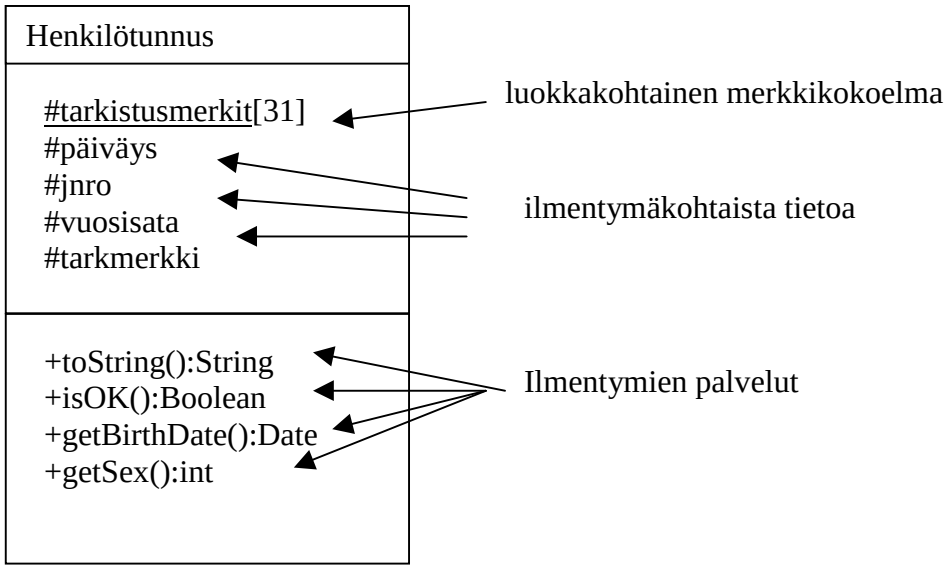
- julkinen palvelu (public, UML:ssä ‘+’) on muihin luokkiin kuuluvien olioidenkin pyydettävissä
- suojattu palvelu (protected, UML:ssä ‘#’) on käytettävissä palveluissa, jotka on määritelty samassa luokkakuvauksessa kuin käytettävä palvelu tai tämän luokkakuvauksen perivissä kuvauksissa
- yksityinen (private, UML:ssä ’-’) sallii palvelun käytön vain samassa luokkakuvauksessa määritellyissä palveluissa, jossa palvelu on määritelty.

- Paluuarvon tyyppi

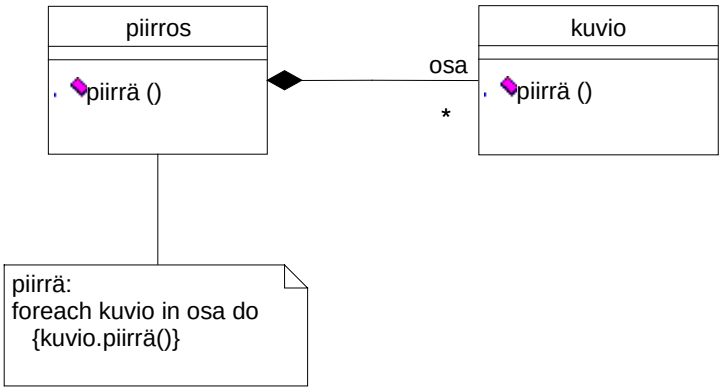
  Paluuarvon typpinä voi olla jokin perustietotyyppi. Paluuarvona voi olla myös olio, jolloin tyyppinä on olion luokka.
- Muut määreet

  Palveluun voidaan liittää muita määreitä. UML määrittely sisältää valmiina muutaman samanaikaisuuden hallintaan liittyvän määreen.

Kuvaesityksessä palvelusta saattaa olla järkevää esittää vain nimi ja mahdollisesti näkyvyys (kuva 4.6). Joskus on hyödyllistä ottaa kaavioon mukaan muistilappu, jossa voi kuvata jonkin keskeisen palvelun toimintaperiaatteen (kuva 4.7). Palvelujen täsmälliseen kuvaukseen tarvitaan aina lisäksi tekstimuotoinen selvitys siitä, mitä palvelussa tehdään.



Kuva 4.6: Luokka- ja ilmentymäkohtaiset attribuutit



Kuva 4.7: Palvelun täsmennys muistilapulla.

Luokan ilmentymä kuvataan UML:ssä samanlaisella symbolilla kuin luokkakin. Erona on se, että ilmentymäkuvauksessa luokkanimen tilalla ylimmässä lokerossa on ilmentymäviite. Tämän rakenne on seuraava

```
<tunnus> | [<tunnus>] : <luokkanimi>
```
[2]

Ilmentymäviite esitetään alleviivattuna. Ilmentymäviite voisi olla esimerkiksi

Kalle

Kalle:Henkilö

:Henkilö

Attribuuttiosassa voidaan antaa attribuutin nimen lisäksi attribuutin arvo. Arvo annetaan vain niiden attribuuttien osalta, joihin lukijan halutaan kiinnittävän huomiota. Ilmentymäsymboleita käytetään esimerkeissä haluttaessa havainnollistaa luokkakaavion määrittelemää rakennetta (kuva 4.8)

Roope:Asiakas
AsiakasNumero= 12345 Nimi=Roope Ankka Osoite=Ankkalinna 1 Luottoraja=99999999999999

Kuva 4.8: Asiakas –luokan ilmentymä

[2] Pystyyviiva (|) erottaa vaihtoehdot.

4.2 Olioiden väliset yhteydet

Olioiden (ilmentymien) välisiä rakenteellisia kytkentöjä kutsutaan yhteyksiksi (association). Yhteys kahden olion välillä on olemassa esimerkiksi silloin kun olio on toisen olion osa. Reaalimaailmassa tällaisia tilanteita olisivat esimerkiksi

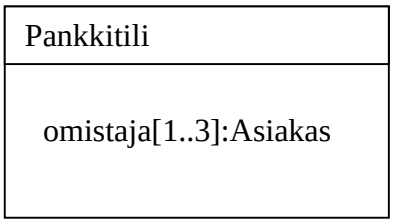
- luku on kirjan osa
- hytti on laivan osa

Olioiden välillä voi olla myös muunlaisia yhteyksiä, esimerkiksi:

- henkilö työskentelee yrityksessä
- opiskelija suorittaa kurssia
- kirja kuuluu kurssin oppimateriaaliin

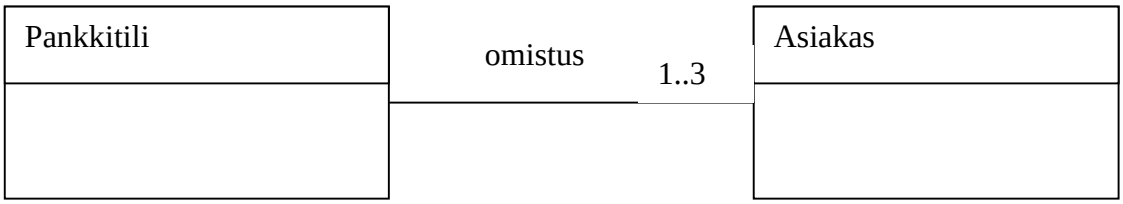
Yhteyden *'henkilö työskentelee yrityksessä'* ilmentymänä voisi olla *'Timo Alanko työskentelee Helsingin yliopistossa'*. Rakenteellisella kytkennällä tarkoitetaan tilannetta, jossa kytkentä olioiden välillä ei ole vain sattumanvaraista ja hetkellistä esimerkiksi *'Pekka käyttää Java Programming kirjaa näytön päällä istuvan ampiaisen hätistelyyn*'. Tässä Pekan ja kirjan välillä on selkeä kytkentä tuon hätistelyn ajan. Sensijaan kytkentä ei ole vallitseva tilanne, asiantila, jonka saisimme selville henkilöiden työhuoneita tai työympäristöä analysoimalla. Rakenteellinen kytkentä sensijaan voisi olla se, että *'henkilöllä on kirja lainassa*'. Sen ilmentymä voisi yllämainitussa tilanteessa olla vaikkapa *'Tiinalla on Java Programming kirja lainassa*'. Pekka sattui käyttämään hätistelyyn Tiinan lainaamaa kirjaa.

Olio-ohjelmassa rakenteellinen yhteys ilmenee siten, että oliolla on olioarvoinen attribuutti, joka sisältää viittauksen kytkettyyn olioon (kuva 4.9). Kytkentä voi olla toteutettuna myös jonkin epäsuoran viittauksen avulla.



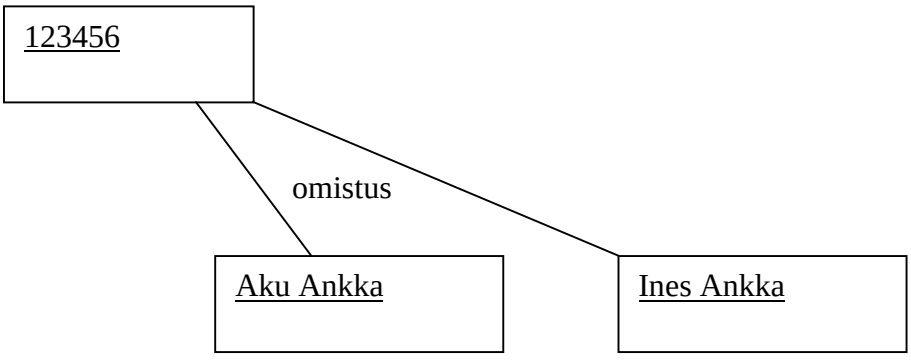
Pankkitili oliolla on moniarvoinen ja olioarvoinen attribuutti Omistaja

kuvataan yhteytenä



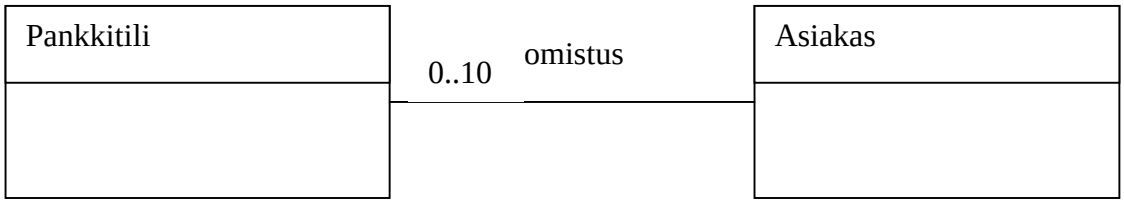
Kuva 4.9: Pankkitilin omistus yhteytenä

Kuvassa 4.9 Omistus on yhteydelle (yhteystyypille) annettu nimi. Lukupari ’1..3’ yhteysviivan päässä kuvaa osallistumisrajoitteen. Mikä tahansa pankkitili voi olla osallisena enintään kolmessa Omistus-yhteydessä, ts. tilillä voi olla enintään 3 omistajaa. Edelleen jokaisen pankkitilin on oltava osallisena ainakin yhdessä Omistus-yhteydessä, ts. tilillä on oltava ainakin yksi omistaja. Ilmentymätasolla tilanne voisi yhden pankkitilin kannalta tarkasteltuna olla kuvan 4.10 mukainen.



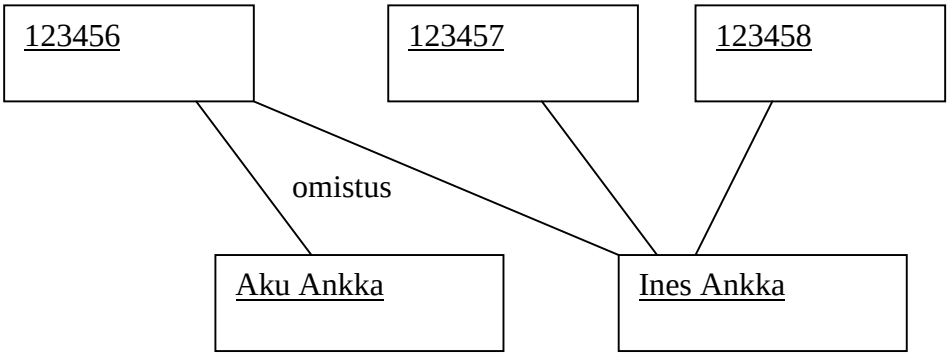
Kuva 4.10: Kahden omistajan yhteinen tili

Yllä on tarkasteltu yhteyttä pankkitilistä asiakkaaseen. Yhteyttä voidaan tarkastella myös vastakkaiseen suuntaan asiakkaasta pankkitiliin. Tällöin voisimme päätyä kuvan 4.11 mukaisiin rajoitteisiin.



Kuva 4.11: Omistus-yhteys ja rajoitteet, jotka tarkastelevat suhdetta asiakkaasta pankkitileihin

Kuvassa 4.11 on päädytty rajoitteeseen, jonka mukaan asiakkaalla voi olla enintään 10 pankkitiliä. Asiakkaalla ei tarvitse olla yhtään pankkitiliä. Jos laajennamme ilmentymäesimerkkiä ottamalla mukaan asiakkaiden kaikki tilit päädymme kuvan 4.12 mukaiseen tilanteeseen.

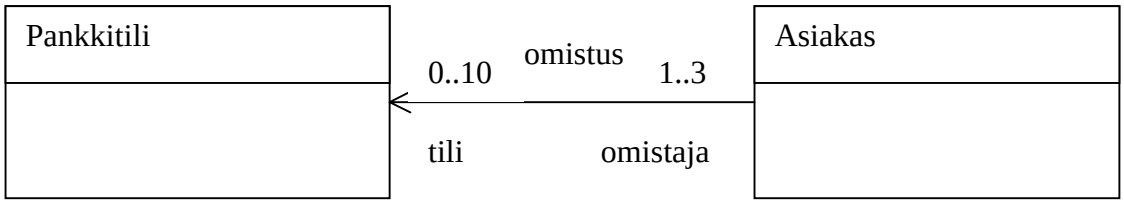


Kuva 4.12: Ilmentymä tilinomistuksista – Ines Ankka omistaa yksinään 2 tiliä ja yhdessä Aku Ankan kanssa yhden tilin

Java-ohjelmassa pankkitiliä ja asiakasta vastaavat luokat voisi olla määritelty seuraavasti

```
public class Pankkitili {
    TiliNumero tilinumero,
    int [] omistaja;  // arvona asiakasnumero
   jne
}
public class Asiakas {
    int asiakasnumero;
    Pankkitili [] tili;
   jne
}
```

Tässä ratkaisussa asiakkaasta on suorat olioviitteet asiakkaan pankkitileihin. Pankkitilistä puolestaan on epäsuora asiakasnumeron avulla hoidettu viite omistaja-asiakkaisiin. Tällöin asiakkaasta on ’suora pääsy’ pankkitiliin, mutta pankkitilistä ei ole suoraa pääsyä asiakkaaseen. UML:ssä tämä voidaan esittää navigointimääreen avulla. Navigointimääre esitetään liittämällä yhteysviivaan avoin nuolenkärki, joka osoittaa siihen luokkaan, jonne toisesta osapuolesta on suora pääsy. (kuva 4.13)



Kuva 4.13: Omistus yhteyden toteutustapa siten, että asiakkaasta on suora pääsy pankkitiliin.

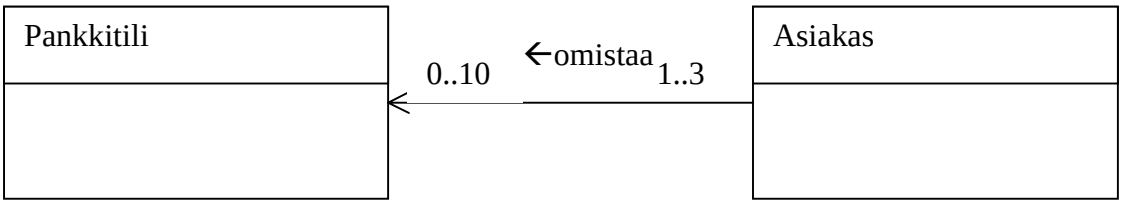
Suora pääsy on mahdollista määritellä kumpaankin suuntaan. Yleensä se toteutetaan ainakin toiseen suuntaan. Sen määrittely kuulu matalan tason ohjelmointiläheiseen kuvaukseen. Korkeamman abstraktiotason kuvauksissa se jätetään teknisen toteutuksen yksityiskohtana kuvaamatta.

Kuvassa 4.13 on yhteyden kuvaukseen otettu mukaan roolinimet ’tili’ ja ’omistaja’. Rooli kuvaa yhteyden osapuolen asemaa suhteessa toiseen osapuoleen, esimerkiksi asiakkaan asema suhteessa pankkitiliin on ’omistaja’. Kuvassa roolinimet ovat samat

kuin yllä olevassa Java-ohjelmassa yhteyden toteutukseen käytettyjen attribuuttien nimet. Tämä kuvastaakin varsin luontevaa tapaa edetä suunnitelmasta toteutukseen käyttämällä ohjelmassa suoraan niitä nimiä, jotka esiintyvät suunnitelmassa.

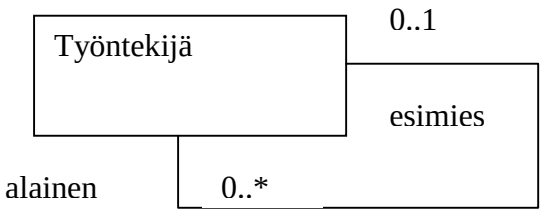
Olioiden väliset yhteydet esitetään luokkakaaviossa kytkemällä osapuolten luokat yhteen viivalla. Viivan päissä ilmoitetaan osallistumisrajoitteet ja roolinimet. Osallistumisrajoite esitetään *alaraja .. yläraja* –parina. Käytännössä eniten merkitystä on alarajoilla 0 ja 1. Alaraja 0 tarkoittaa, että osapuoliluokan olioiden ei tarvitse olla mukana yhdessäkään yhteydessä eli yhteys on niille valinnainen (esim. asiakkaalla ei tarvitse olla pankkitiliä). Alaraja 1 puolestaan merkitsee pakollista yhteyttä (esim. tilillä on oltava vähintään yksi omistaja). Ylärajoista eniten on merkitystä arvoilla 1 ja * (epämääräisen monta). Yläraja 1 määrittelee yhteyden funktionaaliseksi. Kullakin oliolla on enintään yksi kumppani (esim. kurssilla on vain yksi vastuuhenkilö). Usein tiedetään, että olio voi olla yhteydessä moneen olioon, mutta ei ole mahdollista antaa mitään kiinteää ylärajaa kumppaneiden määrälle. Tällöin käytetään ylärajaa * (monta), esimerkiksi tilanteessa ’*kurssilla on monta opiskelijaa*’.

UML:ssä yhteyden voi nimetä käyttäen joko suunnattua tai suuntaamatonta nimeä. Esimerkiksi ’omistus’ on suuntaamaton nimi ja ’omistaa’ on suunnattu. Jotta suunnattua nimeä käytettäessä ei syntyisi tulkintaongelmia siitä miten yhteysnimi pitäisi tulkita voidaan nimeen liittää lukusuunta (kuva 4.14). Myös roolinimien käyttö ehkäisee tulkintaongelmia.



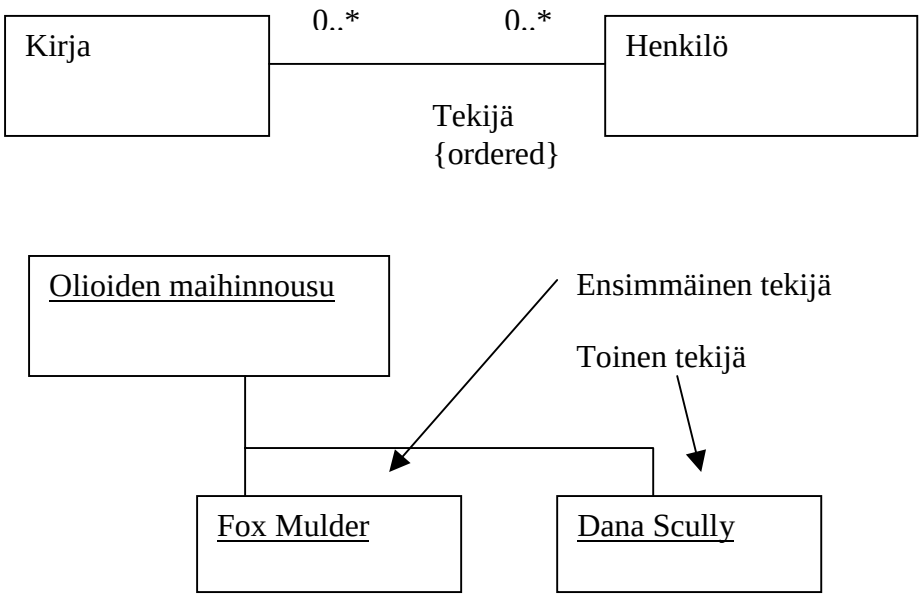
Kuva 4.14: Yhteysnimen lukusuunta: Asiakas omistaa pankkitilin.

Yhteydet molemmat osapuolet voivat kuulua samaan luokkaan. Tällöin roolinimien käyttö on välttämätöntä. Kuvan 4.15 esimerkissä kuvataan työnjohtosuhdetta.



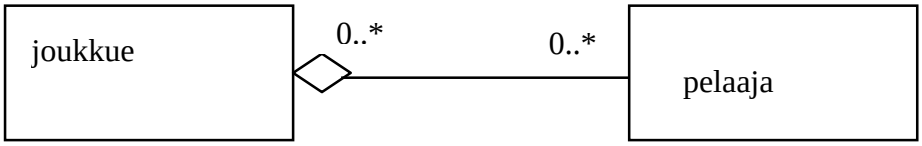
Kuva 4.15: Yhteys, jonka osapuolet kuuluvat samaan luokkaan.

Yhteyteen voidaan liittää järjestysmääre kuvaamaan sitä, että samaan olioon yhteydessä olevien olioiden joukko on jollain perusteella järjestetty. Kuvassa 4.16 on esimerkki tällaisesta yhteysmäärittelystä ja sen mukaisesta ilmentymästä.



Kuva 4.16: Järjestetty kumppanijoukko.

Luokkakaavioita laadittaessa joudutaan usein kuvaamaan kokonaisuuden ja siihen kuuluvan osan välisiä yhteyksiä. Tällaiselle yhteydelle on oma nimityksensä kooste (aggregate) ja UML:ssä oma merkintätapansa, salmiakkisymboli kokonaisuuden puoleisessa päässä. (kuva 4.17). Kuvan 4.17 esimerkissä joukkueeseen voi kuulua monta pelaaja ja pelaaja voi kuulua useaan joukkueeseen. Pelaaja on osa joukkuetta.



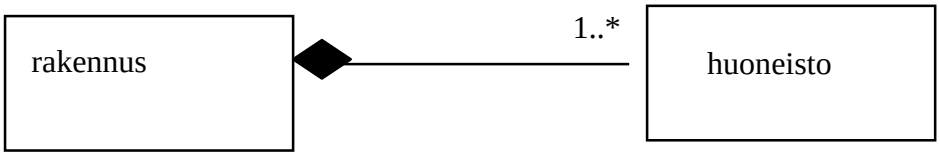
Kuva 4.17: Koosteyhteys.

Koosteen käyttö lisää mallin luettavuutta. Muuta merkitystä sillä ei ole. Se ei vaikuta millään tavalla navigointiin eikä osallistumisrajoitteisiin. Koosteen tapauksessa osan ja kokonaisuuden välinen kytkentä on löyhä. Osa voidaan irrottaa kokonaisuudesta ja liittää toiseen. Esimerkissä pelaaja voi vaihtaa joukkuetta ja kuulua moneen eri joukkueeseen. Joukkuen lopettaminen ei vaikuta mitään pelaajiin.

Koostetta merkittävämpi rakenne mallintamisen kannalta on kompositio (composition). Kompositioon liittyy tiukkoja rajoituksia:

- kompositiossa osa on olemassaoloriippuva kokonaisuudesta. Kokonaisuuden tuhoaminen hävittää myös sen osat.
- osa voi kuulua vain yhteen samantyyppiseen kompositioon
- osa on koko elinaikansa kytkettynä samaan kokonaisuuteen.

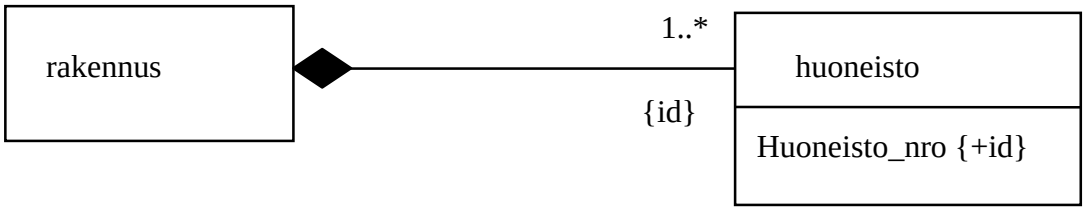
Kompositio kuvataan UML:ssä mustalla salmiakkisymbolilla kokonaisuuden puolessa päässä yhteysviivaa (kuva 4.18).



Kuva 4.18: Kompositio

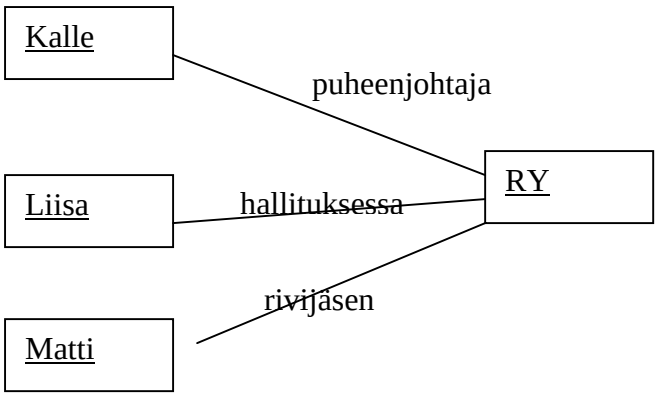
Kuvan 4.18 esimerkissä ’huoneisto’ on koko olemassaolonsa ajan osa samaa rakennusta. Jos rakennus puretaan häviää myös huoneisto. Olio-ohjelmassa komposition tehtävä on luoda ja tuhota osansa. Kompositiossa kokonaisuuteen liittyvien osien joukon ei tarvitse olla kiinteä (se voi toki olla sitä). Kompositioon voidaan lisätä osia

ja siitä voidaan poistaa niitä Kompositiota voidaan käyttää hyväksi myös sen osien ulkoisessa tunnistamisessa Esimerkiksi huoneisto identifioidaan yleisesti identifioimalla rakennus, johon huoneisto kuuluu. Rakennuksen sisällä huoneiston identifiointiin riittää sisäinen huoneistonumero. Tällainen tilanne on esitetty kuvassa 4.19. Perus UML:ssä ei ole mahdollista kuvata yhteyteen perustuvaa tunnistamista. Niinpä tekniikkaa on tässä laajennettu siten, että asia voidaan kuvata osan puoleiseen päähän yhteysviivaa sijoitettavalla {id} määreellä. Konposition sisäisesti käytettävä tunnistava attribuutti osoitetaan määreellä {+id}.



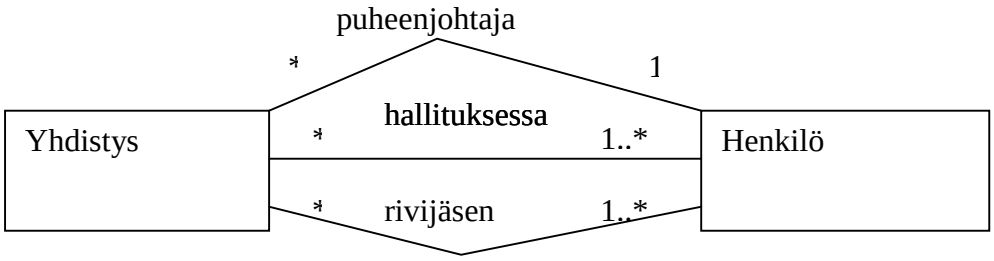
Kuva 4.19: Osan tunnistaminen komposition ja sen sisäisen tunnuksen avulla.

Joskus on tarpeen liittää yhteyteen täsmentävää tietoa siitä millainen yhteys on kyseessä. Tarkastellaan esimerkkinä jäsenyyttä yhdistyksessä. (kuva 4.20).



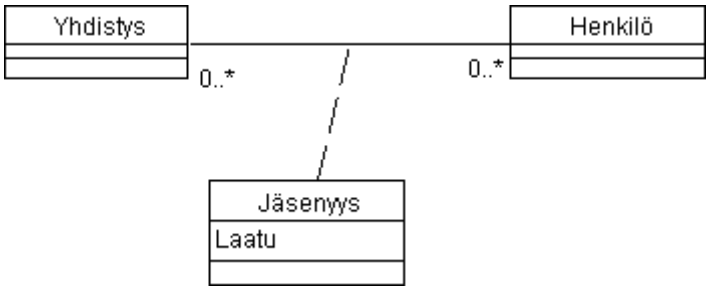
Kuva 4.20: Erilaisia jäsenyyksiä.

Jos vaihtoehtoja on vähän kukin niistä voitaisiin mallintaa omana yhteystyyppinään. (kuva 4.21).

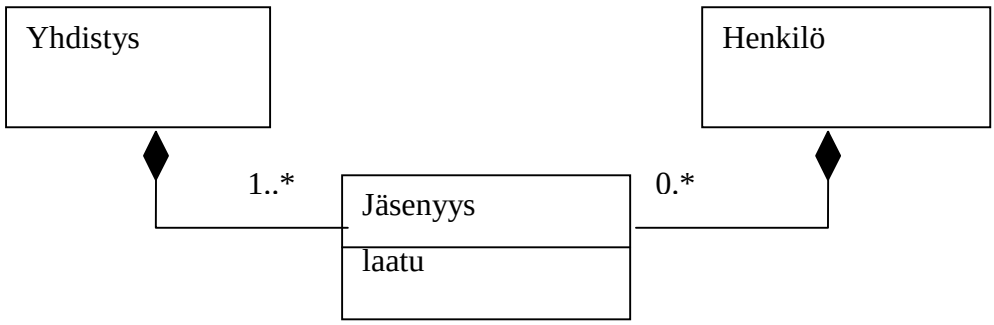


Kuva 4.21: Yhteyksien luonnehdinta käyttämällä yhteystyyppejä.

Jos vaihtoehtoja sen sijaan on runsaasti tarvitaan jokin muu tapa. UML:ssä on tähän tarkoitukseen tarjolla yhteysluokka (association class). (kuva 4.22). Kolmas mahdollinen tapa olisi määritellä erillinen luokka jäsenyys ja kytkeä se yhdistykseen ja henkilöön (kuva 4.23).



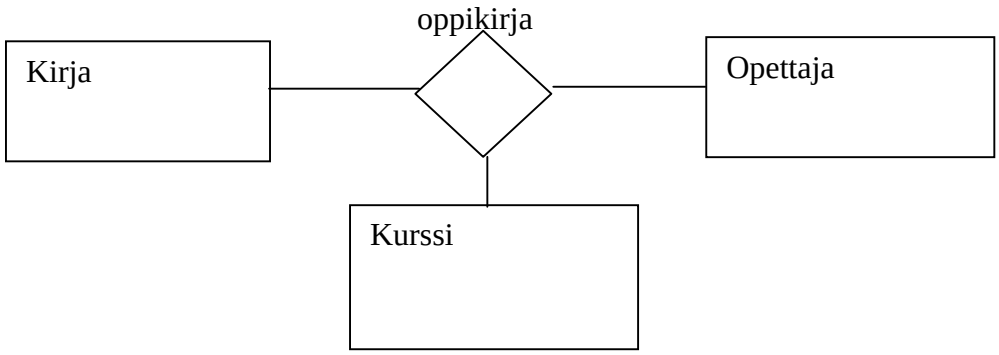
Kuva 4.22: Yhteyden luonnehdinta yhteysluokan avulla.



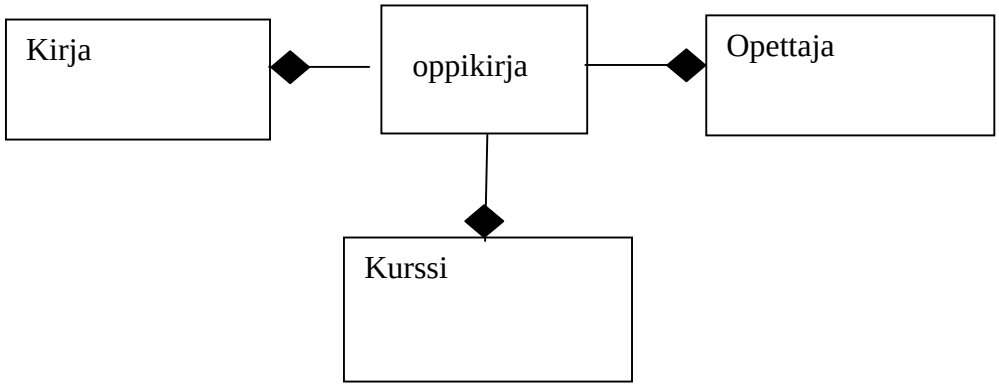
Kuva 4.23: Yhteyden luonnehdinta tekemällä siitä erillinen luokka

Kuvien 4.22 ja 4.23 kuvaamien vaihtoehtojen ohjelmatoteutus voi olla aivan samanlainen.

UML:ssä on tarjolla myös useamman kuin kahden osapuolen välinen yhteys. Tällaisesta on esimerkki kuvassa 4.24. Siinä eri opettajat käyttävät samalla kurssilla eri oppikirjaa. Tämä voidaan muuntaa kahdenvälisiin yhteyksiin perustuvaksi kuvan 4.25 mukaisesti.

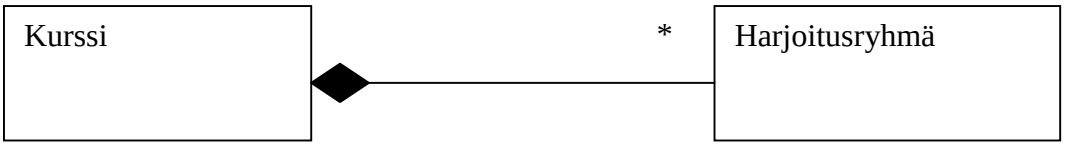


Kuva 4.24: Kolmenvälinen yhteys



Kuva 4.25: Kolmenvälisen yhteyden korvaaminen luokalla

Luokkakaavioita voidaan laatia ohjelmaläheisinä, jolloin niillä pyritään mahdollisimman tarkoin kuvaamaan ohjelman oliorakenne. Korkeammalla abstraktiotasolla laaditut mallit jättävät joitain yksityiskohtia esittämättä. Eräs tyypillinen ero korkeamman tason kaavion jä ohjelmaläheisen kaavion välillä liittyy olioiden välisiin yhteyksiin. Ohjelmaläheisessä kuvauksessa esitetään miten olioiden välinen yhteys teknisesti toteutetaan. Korkeamman abstratiotason kuvauksessa riittää kuvata keskeisten yhteyksien olemassaolo. Tarkastellaan esimerkkinä kurssin ja harjoitusryhmän välistä yhteyttä, joka loogisesti on kuvattu kuvassa 4.26.

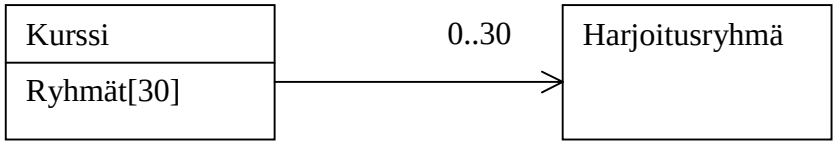


Kuva 4.26: Harjoitusryhmä on kurssin osa.

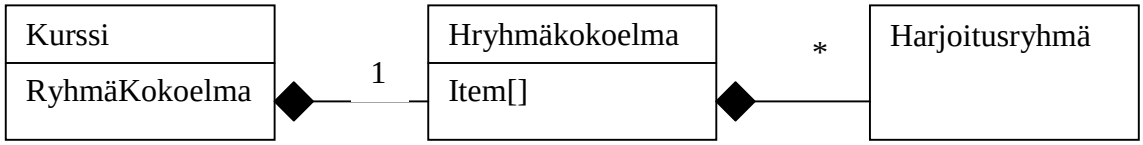
Tarkastellaan toteutusta, joka mahdollistaa pääsyn kurssista harjoitusryhmään. Vaihtoehtoisina tapoina voisivat tulla kyseeseen esimerkiksi

- taulukko
- jokin kokoelmarakenne (container) tai
- listarakenne.

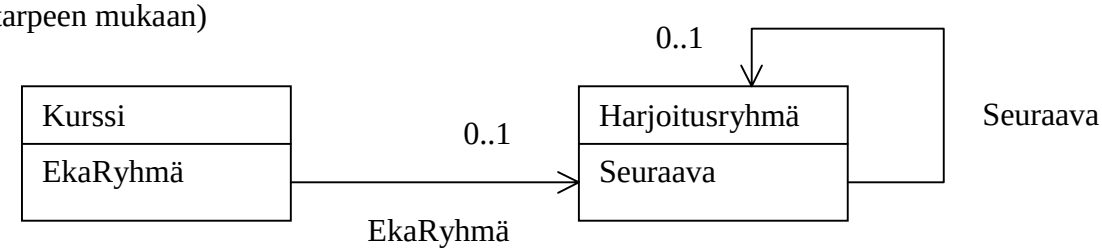
Kuvassa 4.27 on esitetty näitä vastaavat teknisen tason luokkakaaviot.



a) taulukkona



b) kokoelmaluokan avulla (kokoelma olisi tässä dynaaminen taulukko, joka laajenee tarpeen mukaan)

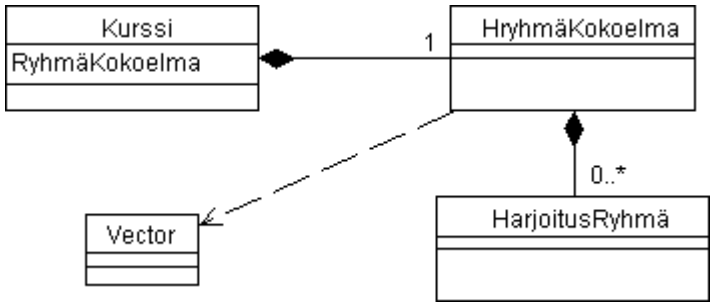


c) listarakenteena

Kuva 4.27: Kurssin ja harjoitusryhmän välisen yhteyden tekninen toteutus.

4.3 Luokkien väliset riippuvuudet

Kuvan 4.27 b-kohdan toteutuksessa käytettäisiin varmaankin hyväksi jotain valmista kokoelmaluokkaa (säiliöluokkaa). Javassa tällainen luokka on esimerkiksi kielen peruskomponentteihin (util-pakkaus) kuuluva Vector-luokka. Jos haluamme kaaviossa näkyvän, että HRyhmäKokoelma onkin Vector-luokan soveltaminen kyseiseen tilanteeseen, tämä voidaan kuvata luokan HryhmäKokoelma riippuvuutena luokasta Vector (Kuva 4.28).



Ohjelmakoodina:

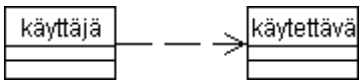
```
public class Kurssi {
    Vector RyhmäKokoelma;   //HRyhmäKokoelma

    public void  lisääRyhmä(HarjoitusRyhmä h) {
        RyhmäKokoelma.addItem(h);
     }
```

Kuva 4.28: Kokoelman toteutus yleiskäyttöisen kokoelmaluokan avulla.

Kuvan 4.28 ohjelmakoodissa luokkaa Hryhmäkokoelma ei näyttäisi olevan olemassa; on vain luokan Vector ilmentymä. Yleiskäyttöisten kokoelmaluokkien kukin ilmentymä käyttäytyy kuitenkin eri tavoin ja käyttäytymissäännöt voidaan ilmaista vain luokkatasolla. Esimerkiksi tässä tapauksessa ilmentymän pitäisi hävitä ja hävittää myös siihen kytketyt harjoitusryhmät kun kurssi hävitetään. Näinollen luokka on loogisesti olemassa..

Luokka toisen luokan ilmentymänä on yksi esimerkki luokkien välisistä riippuvuuksista. Yleisesti riippuvuudella tarkoitetaan tilannetta, jossa luokan määrittelyissä tapahtuvalla muutoksella voi olla vaikutuksia toisen luokan toimintaan. Riippuvuus esitetään kaaviossa katkoviivalla, jonka päässä oleva nuolenkärki osoittaa siihen luokkaan, josta toinen luokka on riippuva. (Kuva 4.29) Katkoviivaan voidaan liittää tekstinä riippuvuuden tyyppi, yllä se olisi ollut ’*instance of*’.



Kuva 4.29: Luokkien välinen riippuvuus Käyttäjä on riippuvainen Käytettävästä.

Usein esiintyvä riippuvuus on palvelun parametrin aiheuttama riippuvuus, jossa palvelun tarjoava luokka tulee riippuvaksi parametrin luokasta. Alla oleva ohjelmakoodi aiheuttaisi tällöin kuvan 4.29 mukaisen riippuvuuden. Riippuvuuden tyyppi olisi tällöin ’*use*’.

```
class Käyttäjä {
     public omaPalvelu(Käytettävä k) {
        k.vierasPalvelu();
   }
}
```

4.4 Luokkakaavion laatiminen

Järjestelmän oliperustaista toteutusta voidaan ajatella simulointimallin muodostamisena jostakin reaalimaailman ilmiöstä. Tällöin ohjelmassa toimivat oliot vastaavat pitkälti reaalimaailman ilmiössä osallisena olevia olioita. Lisäksi tarvitaan olioita, joiden kautta voidaan tarkastella simulointimallin tilaa ja välittää siihen liittyvää ohjaustietoa (eli käyttöliittymä). Keskeistä simulointimallin muodostamisessa on löytää ne reaalimaailman kohteet, joita vastaavia luokkia ohjelmaan tulisi ottaa mukaan.

Näiden luokkien etsimistä voidaan kutsua oliomallinnukseksi tai käsiteanalyysiksi (conceptual modeling). Tehtävänä on löytää oleellinen rakenne simuloitavalle reaalimaailman ilmiölle.

Mallin laatiminen voisi tapahtua seuraavien vaiheiden mukaisesti:

1. Kartoita luokkaehdokkaita.

   Laadi luettelo tarkasteltavan ilmiön kannalta keskeisistä kohteista tai ilmiöistä, jotka voisivat tulla kyseeseen luokkina tai olioina. Tällaisia voisivat olla toimintaan osallistujat, toiminnan kohteet, toimintaan liittyvät tapahtumat, materiaalit, tuotteet ja välituotteet, toiminnalle edellytyksiä luovat asiat..

   Kartoituksen pohjana voi käyttää vapaamuotoista tekstikuvausta tarkasteltavasta ilmiöstä, kutsutaan sitä jatkossa kohdealueeksi. Tästä kuvauksesta alleviivataan luokkaehdokkaita ja kerätään ne luetteloon. Luokkaehdokkaat esiintyvät kuvauksessa usein substantiiveina. Verbit voivat ilmaista yhteyksinä. Alustavaa karsintaa voi tehdä sen perusteella onko asia lainkaan oleellinen mallinnettavan ilmiön kannalta.

2. Karsi ehdokkaita

   Luetteloon saadut ehdokkaat käydään läpi ja arvioidaan voisiko ehdokas tulla kyseeseen luokkana. Arvioinnissa tulisi tarkastella
   - Liittyykö ilmiöön tietosisältöä, joka on välttämätöntä järjestelmän kannalta
   - Onko asia riittävän tärkeä kohdealueen kannalta.

   Karsintaa ja ehdokkaiden kartoitusta voidaan joutua tekemään iteratiivisesti. (toistuvasti uudelleen) Ensimmäinen karsintakierros ei välttämättä tuota lopullista tulosta.

3. Tunnista olioiden väliset yhteydet

Yhteyksiä voi etsiä vapaamuotoisesta kuvauksesta (verbit, genetiivit, muut ilmaukset jotka kuvaavat kytkentää). Yhteyksienkin suhteen tulisi miettiä onko yhteys oleellinen tarkasteltavan ilmiön kannalta sekä onko se rakenteellinen.

4. Täsmennä luokkakuvauksia määrittelemällä attribuutit

.

Attribuutteja saattaa löytyä vapaamuotoisesta kuvauksesta, mutta yleensä niiden löytäminen edellyttää lisäselvityksiä kohdealueesta, esimerkiksi toiminnan osapuolten haastatteluja. Attribuuttien kohdalla pitäisi myös selvittää mihin niitä tarvitaan.

5. Määrittele yhteyksiin liittyvät osallistumisrajoitteet.

Osallistumisrajoitteiden avulla ilmaistaan rakenteellisia sääntöjä. Ne eivät välttämättä tule esiin vapaamuotoisessa kuvauksessa vaan edellyttävät tarkempaa kohdealueen analysointia.

6. Liitä luokkiin palvelut.
Karkeassa alustavassa mallissa palveluja ei välttämättä määritellä.

7. Varmista palvelujen ja tietosisällön yhteensopivuus

4.4.1 Esimerkki

Tarkasteltavana ilmiönä on elokuvalipun varaaminen. Lippu oikeuttaa paikkaan tietyssä näytöksessä. Näytöksellä tarkoitetaan elokuvan esittämistä tietyssä teatterissa tiettyyn aikaan. Samaa elokuvaa voidaan esittää useissa teattereissa useina aikoina Teatterin ohjelmakartta määrittelee, mitä elokuvia missäkin näytöksessä esitetään. Asiakas voi samassa varauksessa varata useita lippuja..

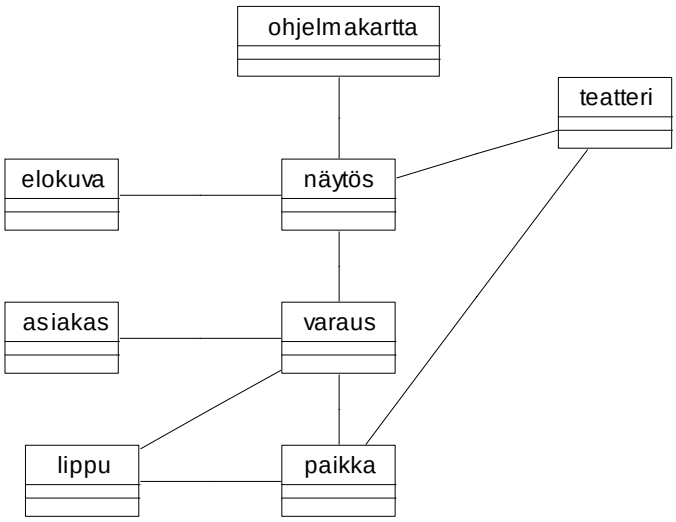
Edellistä kappaletta voidaan pitää ilmiön vapaamuotoisena kuvauksena. Otetaan se jatkokäsittelyyn ja alleviivataan luokkaehdokkaita.

Tarkasteltavana ilmiönä on elokuvalipun varaaminen. Lippu oikeuttaa paikkaan tietyssä näytöksessä. Näytöksellä tarkoitetaan elokuvan esittämistä tietyssä teatterissa tiettyyn aikaan. Samaa elokuvaa voidaan esittää useissa teattereissa useina aikoina Teatterin ohjelmakartta määrittelee, mitä elokuvia missäkin näytöksessä esitetään. Asiakas voi samassa varauksessa varata useita lippuja.

Saatiin luettelo

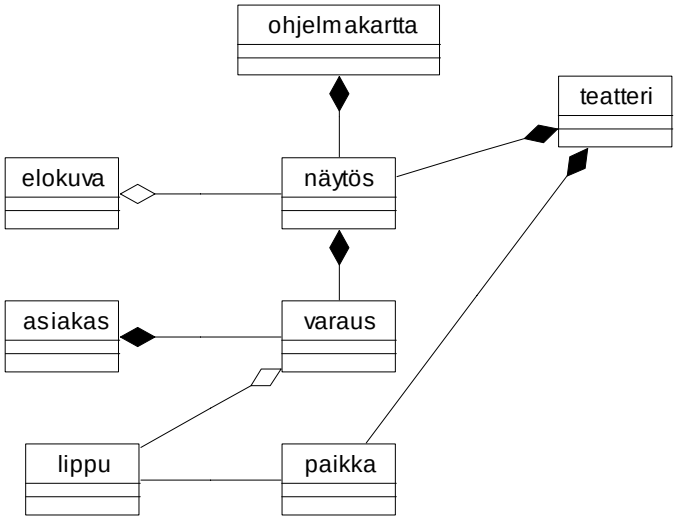
Ehdokas	**Analyysi**
~~elokuvalippu~~	Tämä on lipun synonyymi karsitaan pois.
lippu	oikeus paikkaan, asia, mahdollinen, onko tietosisältöä?
paikka	istuin teatterissa, varauksen kohde, mahdollinen, tunnus
näytös	varauksen kohde, keskeinen, aika
elokuva	näytöksen sisältö, nimi, yms.
teatteri	isompi paikka
asiakas	varauksen tekijä, oleellinen
ohjelmakartta	ohjelmatarjonta, kytkennät ajan ja paikan ja elokuvan välillä
varaus	tuote

Päätetään jättää muut ehdokkaat paitsi elokuvalippu, joka on lipun synonyymi. Tarve ottaa lippu mukaan on kyseenalaista. Sen kohtalo jää ratkaistavaksi myöhemmin attribuuttien selvityksen yhteydessä. Kuvassa 4.30 on ensimmäinen luonnos luokkakaaviosta. Yhteyksiä on luonnosteltu, mutta niihin ei vielä liity rajoitteita eikä edes nimiä.



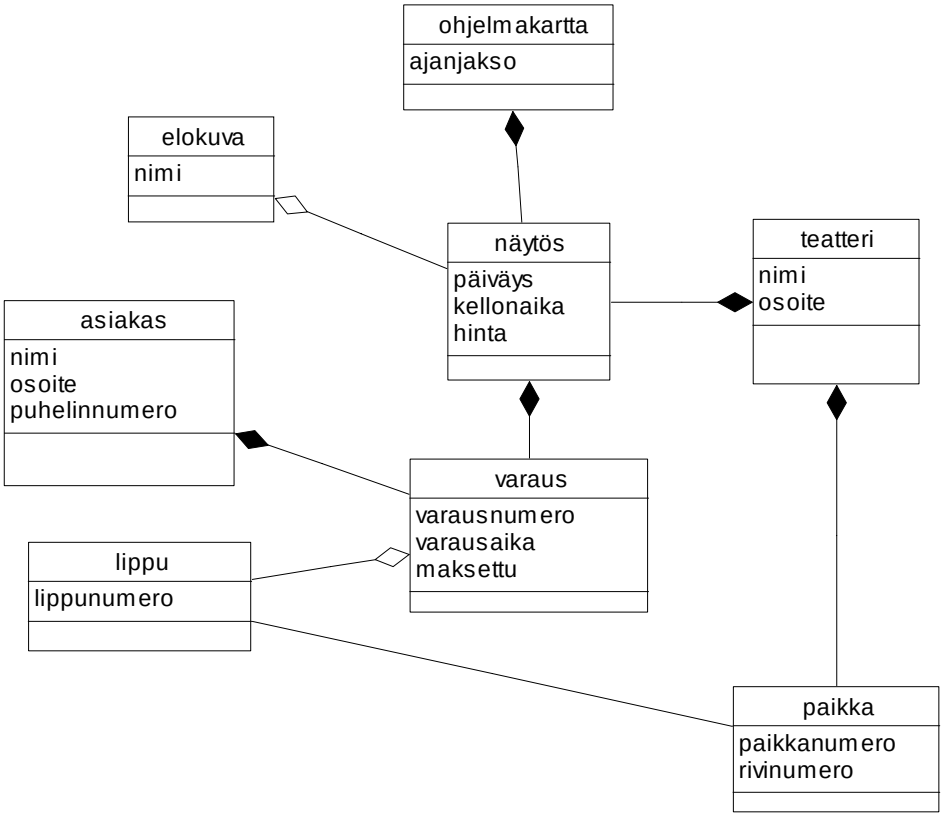
Kuva 4.30: Luokkakaavion ensimmäinen luonnos.

Seuraavassa versiossa on tunnistettu kompositiot ja koosteet (kuva 4.31). Lippu on päätetty säilyttää. Koska varauksen ja paikan välinen yhteys on pääteltävissä varauksen ja lipun sekä lipun ja paikan välisistä yhteyksistä, se karsitaan redundanttina (toisteisena) pois.



Kuva 4.31: Luokkakaavion seuraava versio.

Seuraavaksi kaaviota täydennetään keskeisillä attribuuteilla (kuva 4.32).

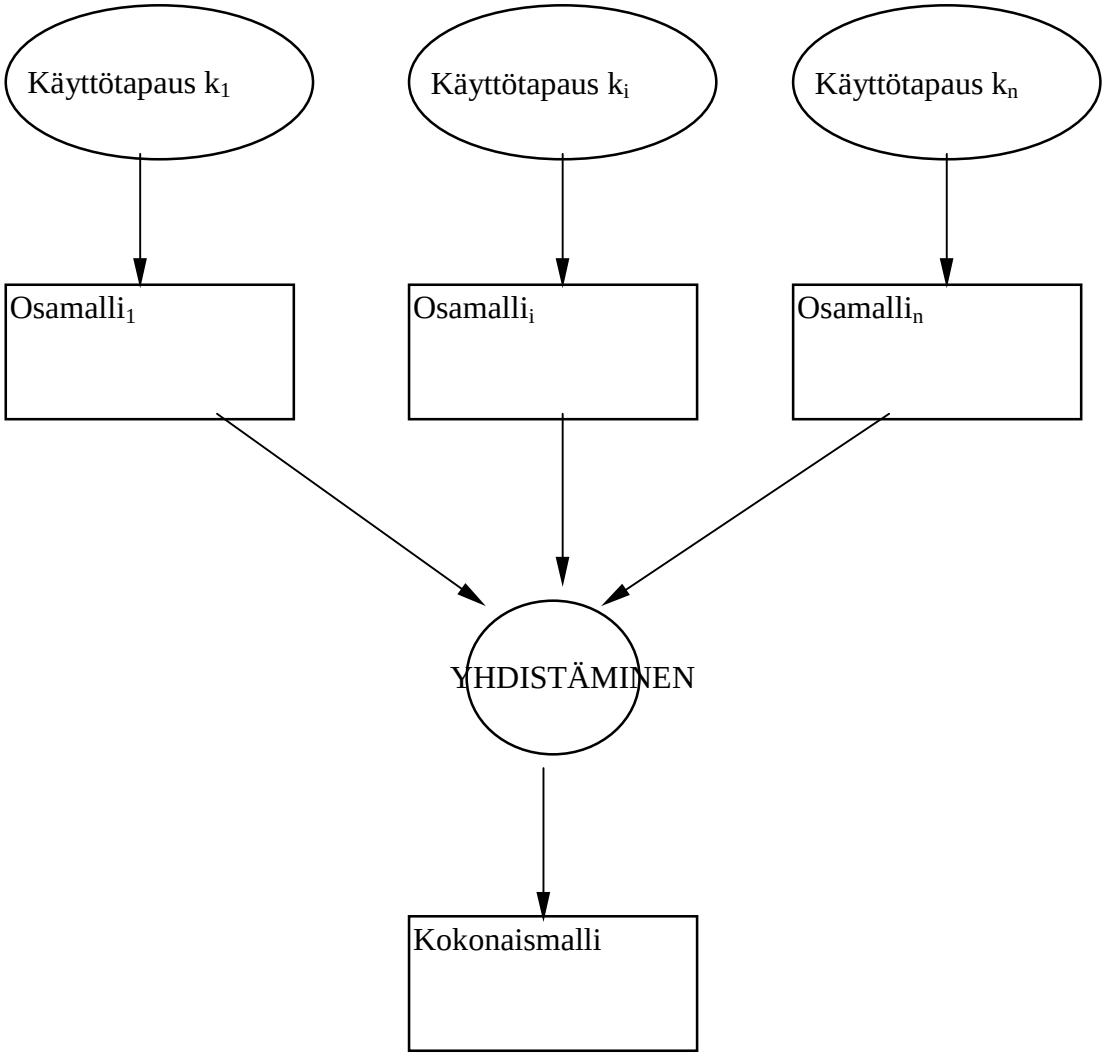


Kuva 4.32: Luokkakaavion seuraava versio

Yhteyksiä ja attribuutteja voidaan vielä joutua määrittelemään uudelleen kun luokkiin liitetään palvelut. Palvelujen määrittelyä tarkastellaan yhteistyökaavioiden käsittelyn jälkeen.

4.5 Käyttötapauspohjainen luokkakaavion laatiminen

Käyttötapaukset voivat toimia myös perustana järjestelmän sisältöluokkien määrittelyssä. Tähän perustuu ns. käyttötapauspohjainen (näkemyspohjainen) määrittely. Tässä lähestymistavassa laaditaan aluksi käyttötapauskohtaisia osamalleja, jotka sitten yhdistetään kokonaisvaltaiseksi malliksi (kuva 4.33).

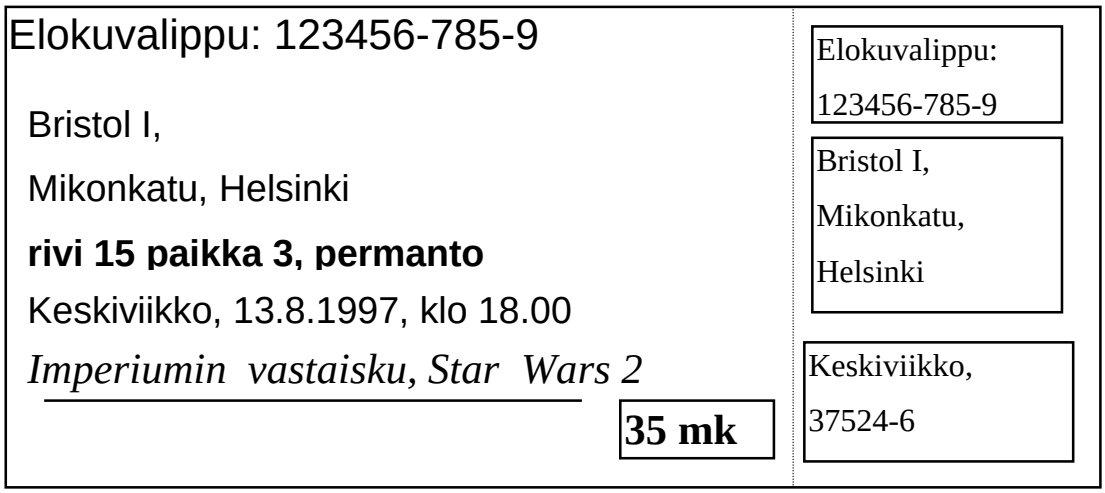


Kuva 4.33: Käyttötapauspohjainen tietosisällön määritys

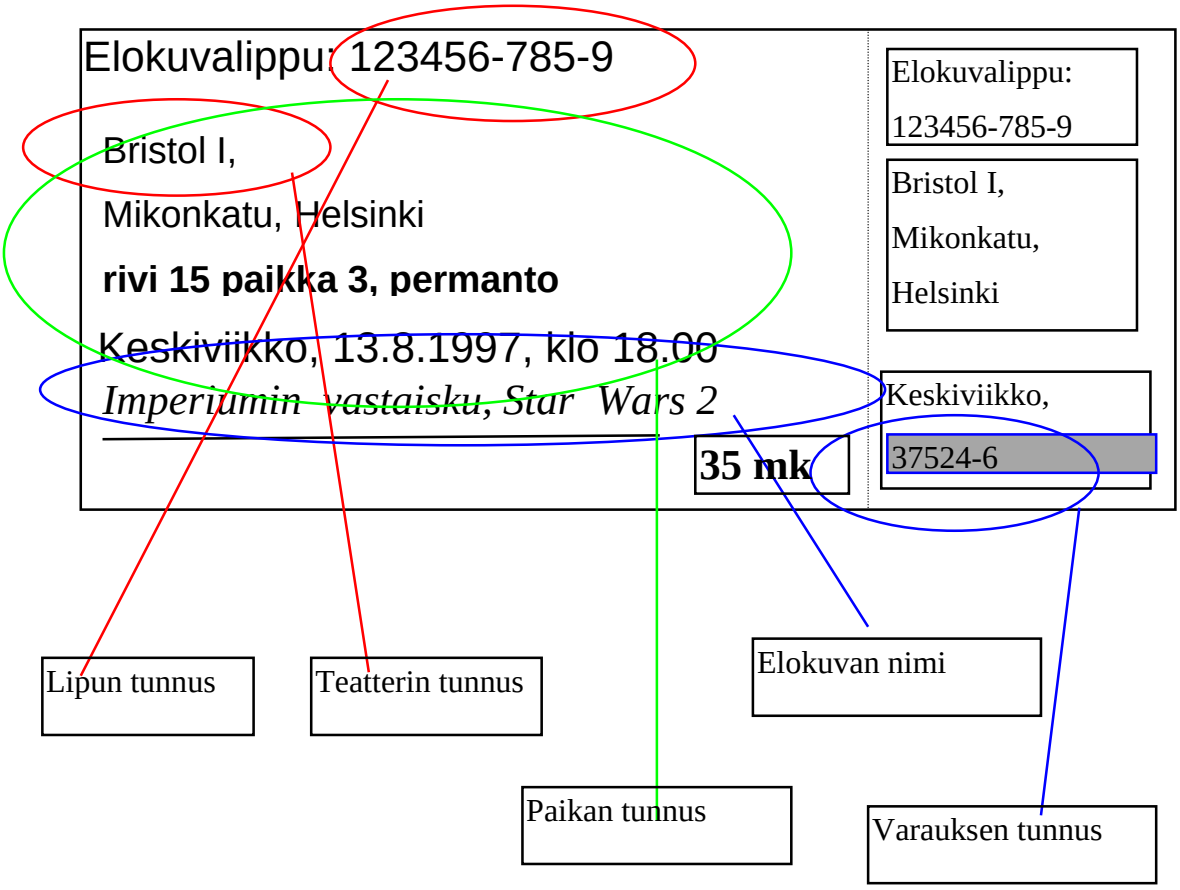
Kukin käyttötapauskohtainen osamalli määrittelee sisältöluokat vain käyttötapauksen edellyttämässä laajuudessa. Syntyvät mallit ovat siten kokonaismallia suppeampia ja näin helpommin aikaansaatavissa. Lähtökohdaksi mallinnukseen voidaan ottaa käyttötapauksen kuvaus, mahdollinen käyttötapaukseen liitetty luonnos käyttöliittymästä, raporttimalli tai lomake, joka liittyy käyttötapaukseen. Jos lähtökohtana on tekstikuvaus, mallinnus voi lähteä liikkeelle siten, että aluksi alleviivataan tekstistä luokkaehdokkaat ja edetään sitten kuten edellä luvussa 4.4 esitettiin.

Lomakkeita tai raportteja analysoitaessa ei näistä löydy välttämättä suoraan luokka-ehdokkaiden nimiä. Sensijaan löytyy tunnistetietoja, jotka epäsuorasti vihjaavat

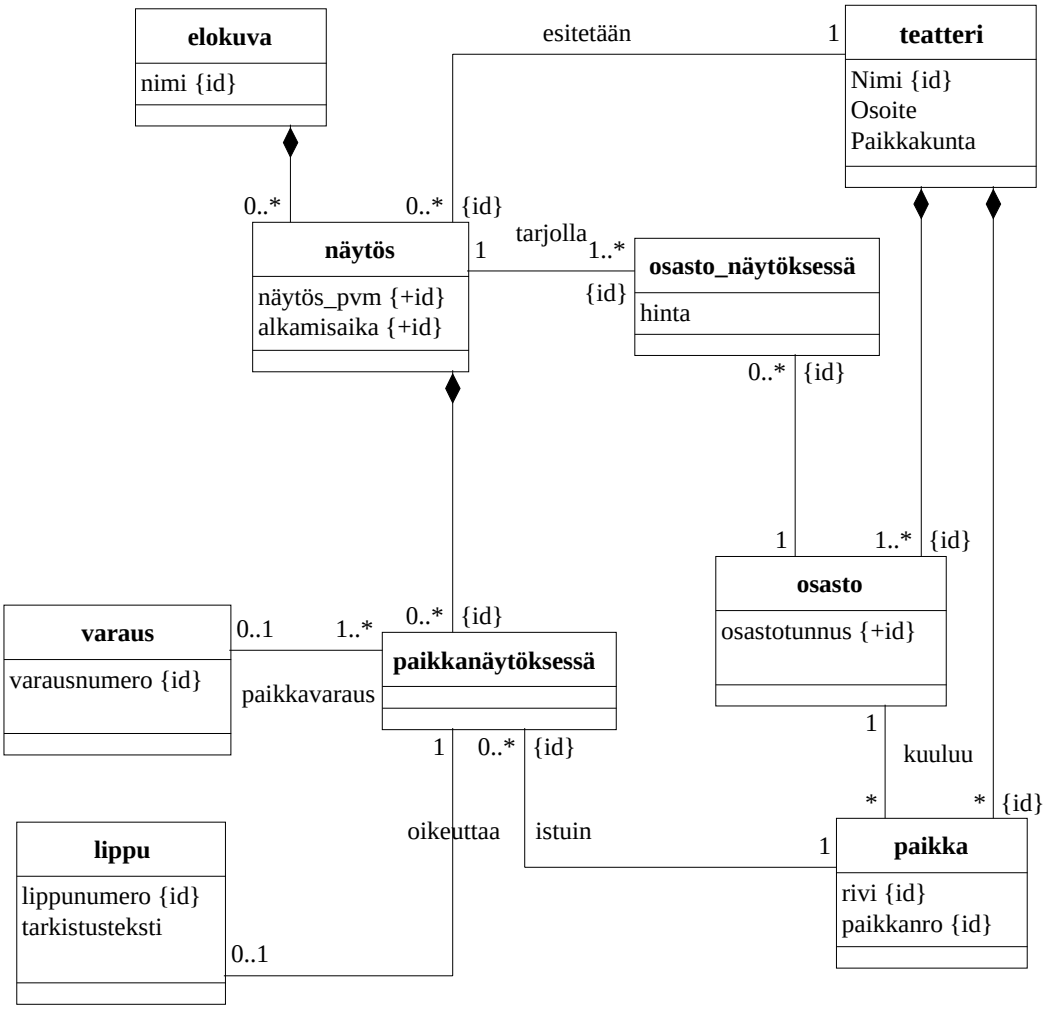
luokan olemassaolosta. Kuvissa 4.34 ja 4.35 on esimerkkinä raporttiin pohjautuvasta analyysista tarkasteltu lippuvarausjärjestelmältä tulokseksi haluttua elokuvalippua. Analyysin tuottama käyttötapauskohtainen osamalli on kuvassa 4.36.



Kuva 4.34: Elokuvalippu analysoinnin kohteena



Kuva 4.35: Elokuvalippu analysoituna



Kuva 4.36: Käyttötapauksen 'lipun tuottaminen' näkemys tietosisällöstä.

Kuvan 4.36 mallin lähtökohtana on tuotettu elokuvalippu. Mallia ei ole kuitenkaan konstruoitu pelkästään lippua tarkastelemalla, vaan sen aikaansaamiseksi on täytynyt analysoida ja selvittää laajemmin esimerkiksi sitä, miten lipun hinta määräytyy. Tätä varten on voitu esimerkiksi haastatella teatterin johtoa. Lopulta on päädytty erilaiseen ratkaisuun kuin aiemmin kuvan 4.32 kaaviossa. Tässä esitettävä näkemys perustuu siihen, että teatteri on jaettu osastoihin, joihin paikat kiinteästi liittyvät. Kaikilla osaston paikoilla on näytöskohtaisesti sama hinta.

Kuvien 4.32 ja 4.36 sisältömallit kuvaavat samaa sisältöä, mutta ne ovat erilaisia. Emme voi välttämättä tuomita kumpaakaan malleista virheelliseksi. Ne ovat erilaisia

näkemyksiä asiasta. Näkemysten yhdistämistehtävän tarkoituksena on tuottaa yhteinen kokonaisnäkemys tietosisällöstä. Yhdistämisen yhteydessä joudutaan osanäkemyksiä vertailemaan ja analysoimaan. Yhdistäminen ei ole helppo tehtävä. Ongelmia tuottavat synonyymit (sama asia nimetty eri tavoin), homonyymit (eri asioista käytetään yhtä nimeä), erilaiset rakenteet ja rajoitteet, jotka vaativat yhteensovitusta. Yhdistämistehtävässä joudutaan usein nostamaan mallin abstraktiotasoa. Näkemysten konkreettiset käsitteet korvataan abstrakteilla käsitteillä. Esimerkiksi lippujärjestelmässä voitaisiin välittää lippuja myös autokilpailuihin, jolloin 'teatteri' ei ehkä ole yhdistetyssä mallissa hyvä nimitys tilaisuuden pitopaikkalle.

Kuvassa 4.33 yhdistely on kuvattu yhtenä tehtävänä, jonka syötteinä on useita käyttötapauskohtaisia osamalleja. Tämä ei tarkoita sitä, että kaikki osamallit tulisi yhdistää kerralla. Osamalleja voidaan tuottaa erilaisin aikatauluin ja niitä voidaan myös yhdistellä eriaikaisesti. Voidaan esimerkiksi toimia siten, että otetaan jokin näkemys yhdistetyn kokonaismallin pohjaksi ja yhdistetään yksi kerrallaan muut näkemykset tähän yhdistemalliin.

4.5.1 Käyttötapausten ja sisältöluokkien yhteensopivuus

Järjestelmän tietosisältö muotoutuu käyttötapausten tuloksena. Käyttötapaukset puolestaan muokkaavat ja hyödyntävät tietosisältöä. Kuvaukset ovat siis riippuvuus-suhteessa toisiinsa. Kuvausten yhteensopivuuden varmistamiseksi voi käyttää riippu-vuusmatriisia. Matriisissa (kuva 4.37) riveinä ovat sisältöluokat ja yhteydet ja sarakkeina käyttötapaukset. Sarakkeen arvoksi tietylle riville merkitään miten käyttötapaus suhtautuu kyseisellä rivillä olevaan luokkaan tai yhteyteen. Sarake jätetään tyhjäksi, jos luokalla ja käyttötapauksella ei ole mitään tekemistä toistensa kanssa. Sarakkeeseen merkitään L (luo), jos käyttötapaus luo luokalle uusia ilmentymiä. Sarakkeeseen merkitään P (poisto), jos käyttötapaus hävittää luokan ilmentymiä. Sarakkeen arvona on M (muuttaa), jos käyttötapaus aiheuttaa muutoksia luokan olioiden attribuutteihin ja K (käyttää) , jos käyttötapaus käyttää hyväksi luokan olioiden tietoja.

	Käyttötapaukset																			
Olioluokat	Uusi artikkeli	Uusi artikkeliversio	Tiedustelu artikkelin tilasta	Lausunnonantajan valinta	Lausunnon saapuminen	Muistutus lausunnosta	Puuttumaan jäänyt lausunt	Valituksen käsittely	Julkaisupäätöksen kirjaus	Palautus korjattavaksi	Julkaistavaksi hyväksymine	Hylkääminen	Viimeistellyn saapuminen	Oikovedoksen lähettäminen	Korjausten vastaanotto	Eripainosten tilaus	Henkilötietojen rekisteröinti	Sisällysluettelo	Vuosikertatiedot	www-sivut	Raportit
Article	L	M	K	K	K	K	K	K	M	M	M	M	M	M	M	K		K	K	K	K
Article version	L	L	K	M	K	K	K	M	M	M	M	M									K
Person	X	X		X		K	K			K	K	K	K	K	K		X				K
Reference			K	L	M	M	M	M													K
Journal																		L	K	K	K
K= Käyttää, L = Luo, M= Muuttaa, P= Poistaa, X=Luo tai muuttaa																					

Kuva 4.37: Riippuvuusmatriisi

Matriisin käyttö kuvausten yhteensopivuuden tarkastukseen perustuu siihen, että jokaiselle sisältöluokalle täytyy löytyä ainakin yksi käyttötapaus, joka luo luokalle ilmentymiä.

- jokaiselle sisältöluokalle täytyy löytyä ainakin yksi käyttötapaus, joka käyttää hyväksi luokan olioiden tietoja
- jos luokan ilmentymäjoukko ei ole jatkuvasti kasvava, pitää löytyä käyttötapaus, jolla ilmentymiä voidaan hävittää (esimerkiksi ylläolevassa matriisissa ei ole lainkaan hävittämisen aikaansaavia käyttötapauksia)
- jos ilmentymän tiedot voivat muuttua, täytyy löytyä käyttötapaus, jolla muutos saadaan aikaan.
- jokaisen käyttötapauksen täytyy jotenkin liittyä vähintään yhteen sisältöluokkaan.

4.6 Yleistyshierarkia

Luokkakaavion laatiminen on kohdealueen jäsentämistä. Sitä tehtäessä luodaan tai kirjataan luokitusjärjestelmä kohdealueen ilmiöille. Luokitusjärjestelmälle voidaan asettaa erilaisia vaatimuksia. Joissakin menetelmissä edellytetään, että ilmiöt on luokiteltava siten, että kaikki luokat ovat erillisiä, ts. niillä ei ole yhteisiä ilmentymiä. Toiset menetelmät puolestaan sallivat ainakin jonkinasteisen luokkien päällekkäisyyden.

Luonnollisessa kielessä luokittelu on vapaata. Käsitteet ja niitä vastaavat luokat ovat limittäisiä tai päällekkäisiä. Kun tarkastellaan miten luokan ilmentymien joukko suhtautuu toisen luokan ilmentymien joukkoon voidaan erottaa tapaukset:

- ilmentymäjoukot ovat aina pistevieraat (esim auto ja henkilö)
- ilmentymäjoukot voivat sisältää yhteisiä ilmentymiä (esim. nainen ja johtaja)
- ilmentymäjoukko sisältyy aina toisen luokan ilmentymäjoukkoon (esim. nainen ja henkilö).

Jos luokan A ilmentymäjoukko sisältää aina luokan B ilmentymäjoukon on luokkien välillä *yleistys-erikoistus* (generalization - specialization) suhde. Käsite B on erikoistapaus käsitteestä A (nainen on henkilön erikoistapaus) tai päinvastoin käsite A on yleistys käsitteestä B (henkilö on yleiskäsite, johon nainen sisältyy). Kun luokkien suhdetta tarkastellaan yleistys-erikoistus –suhteeseen perustuen kutsutaan yleisempää luokkaa yläluokaksi (super class) ja erikoistuneempaa alaluokaksi (subclass). Yläluokka on yleistyshierarkiassa (luokkahierarkiassa) ylemmällä tasolla , siis yleisempi, kuin alaluokka.

Esimerkiksi voitaisiin määritellä

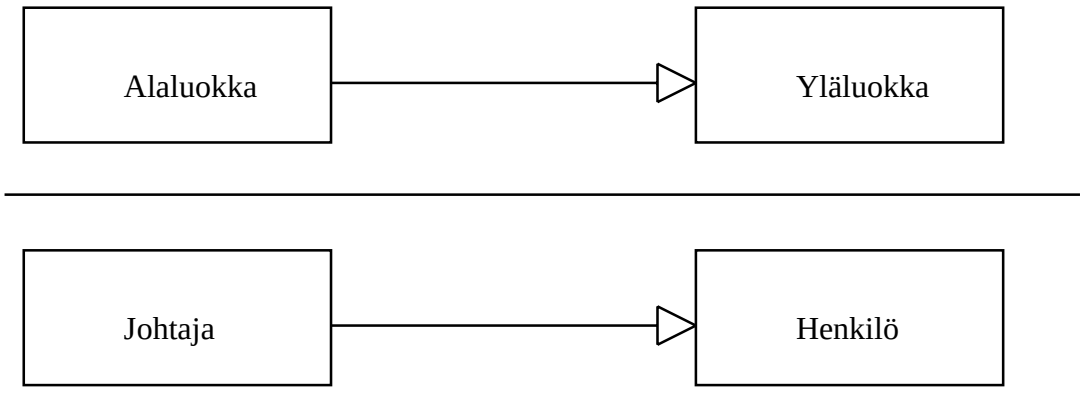
- *luokka nainen on luokan henkilö alaluokka,*
- *luokka johtaja on luokan henkilö alaluokka,*
- *luokat auto, laiva ja lentokone ovat luokan kulkuväline alaluokkia.*

Siitä, että luokka B on luokan A alaluokka seuraa, että jokainen B:n ilmentymä on myös A:n ilmentymä, eli jokainen johtaja on myös henkilö. Tästä taas seuraa, että kaikki yhteydet ja attribuutit, jotka ovat mahdollisia luokan A ilmentymille ovat mahdollisia myös B:n ilmentymille. Jos siis henkilölle on määritelty attribuutti Nimi

ja johtaja on henkilön alaluokka, niin myös johtajalla on automaattisesti Nimi-attribuutti. Jos henkilö on määritelty osapuoleksi työsuhde-yhteyteen, myös johtajan ilmentymä voi olla osapuolena työsuhde yhteydessä. Tätä ilmiötä kutsutaan periytymiseksi (inheritance).

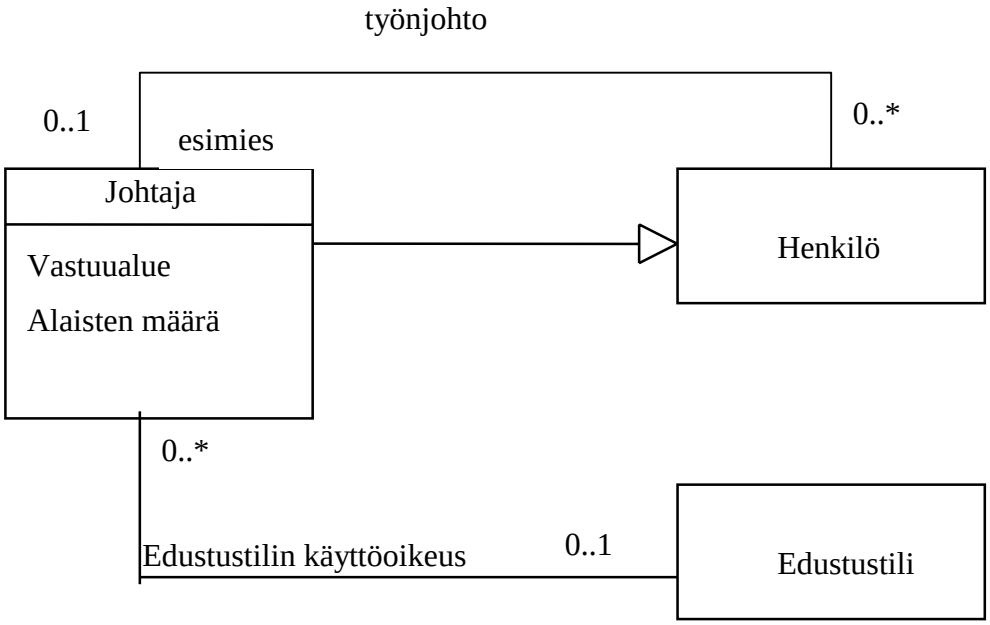
Periytymisessä yläluokkaan liitetyt attribuutit, palvelut ja yhteydet periytyvät alaluokalle, eli ovat voimassa myös alaluokan ilmentymille. On syytä pitää mielessä, että periytyminen on luokkien välistä määrittelyjen periytymistä - ilmentymät eivät peri mitään toisilta ilmentymiltä. Luokkakaavion laatimisen kannalta periytyminen tarkoittaa sitä, että piirrettä, joka on liitetty yläluokkaan ei tarvitse erikseen liittää alaluokkaan, vaan se liittyy sinne automaattisesti .

Alaluokan ja yläluokan välinen riippuvuus kuvataan luokkakaaviossa alaluokasta yläluokkaan osoittavalla nuolella, jossa on iso kolmionmuotoinen kärki (kuva 4.38).



Kuva 4.38: Yleistyshierarkian esittäminen.

Alaluokkaan voidaan liittää yläluokalta perittyjen attribuuttien, palvelujen ja yhteyksien lisäksi omia attribuutteja, yhteyksiä ja palveluja. Esimerkiksi johtajalle voitaisiin lisätä attribuutit ‘vastuualue‘ ja ‘alaisten määrä‘, joita ei ole henkilöillä yleisesti. Johtaja voitaisiin määritellä myös rooliin käyttäjä 'edustustilin käyttöoikeus' -yhteyteen ja rooliin esimies työnjohto-yhteydessä (kuva 4.39).



Kuva 4.39: Johtajan lisäattribuutit ja -yhteydet

Alaluokkaan johtaja voidaan liittää myös sellaisia palveluita, joita henkilöllä ei yleisesti ole. Tällaisia voisivat olla esimerkiksi ’*maksa edustustililtä*’, ’*laadi luettelo alaisista*’. Alaluokassa voidaan myös syrjäyttää (override) yläluokassa määritelty palvelu. Syrjäyttämisellä tarkoitetaan palvelun sisällön tai toteutustavan uudelleen-määrittelyä, nimen säilyessä kuitenkin ennallaan. Esimerkiksi Henkilö-luokalle voisi olla määritelty palvelu ’*Viikkoraportti*’, jonka sisältönä olisi

’*kerro ajankäyttö työtehtäviin*’.

Tämä voitaisiin syrjäyttää Johtaja-luokassa siten, että johtajan ’*Viikkoraportti*’ – palvelun sisältö olisikin

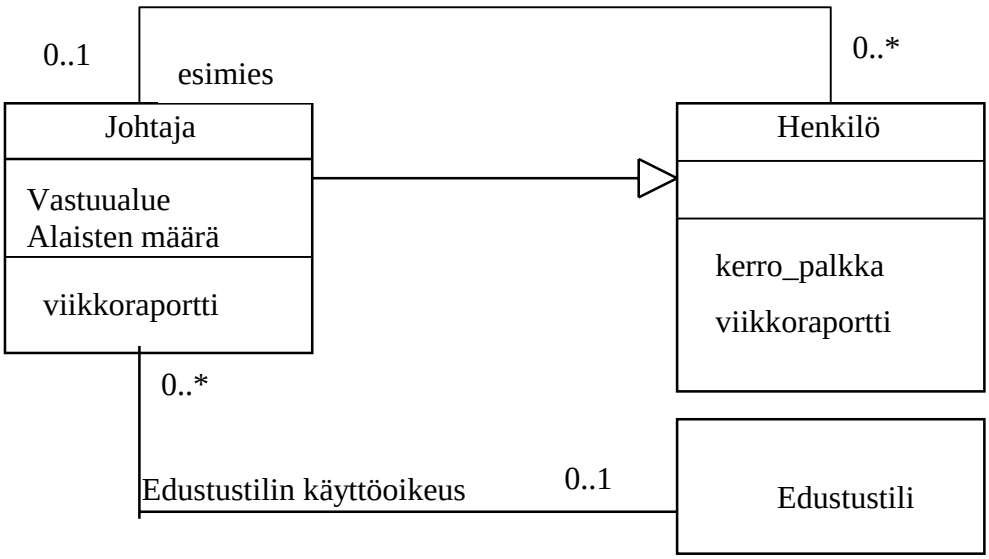
’kerro ajankäyttö työtehtäviin’,

’laadi yhteenveto alaisten viikkoraporteista’

’raportoi edustustilin käyttö’.

Olio-ohjelmoinnissa olio tietää oman luokkansa ja suorittaa palvelunsa oman luokkansa mukaisina. Niinpä olio-ohjelmassa miltä tahansa luokan henkilö tai sen alaluokan oliolta voidaan pyytää Viikkoraportti-palvelua. Jos olio ei ole johtaja, vaan tavallinen henkilö, hän kertoo oman ajankäyttönsä, mutta jos olio onkin johtaja, hän toimii johtajan Viikkoraportti-palvelun mukaisesti ja laatii lisäksi yhteenvedon alaisten viikkoraporteista ja raportoi edustustilin käyttönsä.

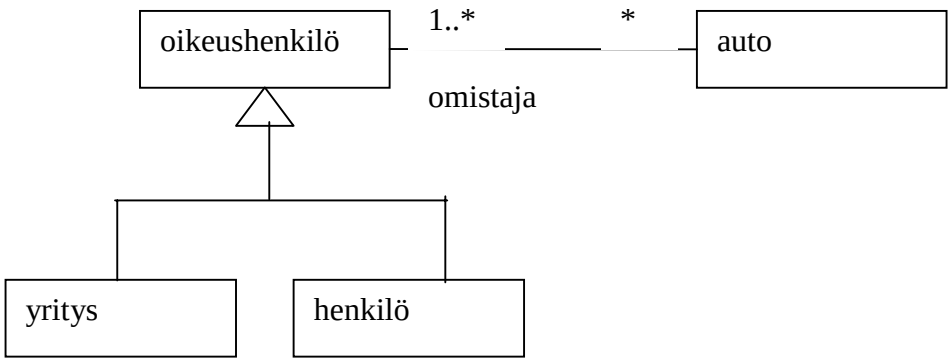
Mahdollisuus syrjäyttää palvelumäärityksiä on olio-ohjelmoinnin tärkeimpiä etuja perinteisiin ohjelmointikieliin verrattuna. Se lisää joustavuutta ohjelmakirjastojen käytössä, edistää ohjelmistokomponenttien uudelleenkäytettävyyttä ja muodostaa perustan erilaisten ohjelmistokehysten rakentamiselle (kehykset ovat eräänlaisia sovellusrunkoja, joissa toiminnan yksityiskohdat on jätetty esimerkiksi syrjäyttämisen avulla täsmennettäväksi). Syrjäyttämismahdollisuus onkin olio-ohjelmoinnin kannalta merkittävin syy yleistyshierarkian käyttöön. Sen hyväksikäyttömahdollisuudet tulevat kuitenkin usein esiin vasta laadittaessa teknisen tason suunnitelmaa ohjelmiston luokkarakenteesta. Luokkakaaviossa syrjäytetty palvelu kuvataan toistamalla palvelun nimi alaluokassa. Kuva 4.40 on henkilö-luokan palvelu viikkoraportti syrjäytetty luokassa johtaja. Sen sijaan palvelu kerro_palkka periytyy samansisältöisenä luokalta henkilö luokalle johtaja.



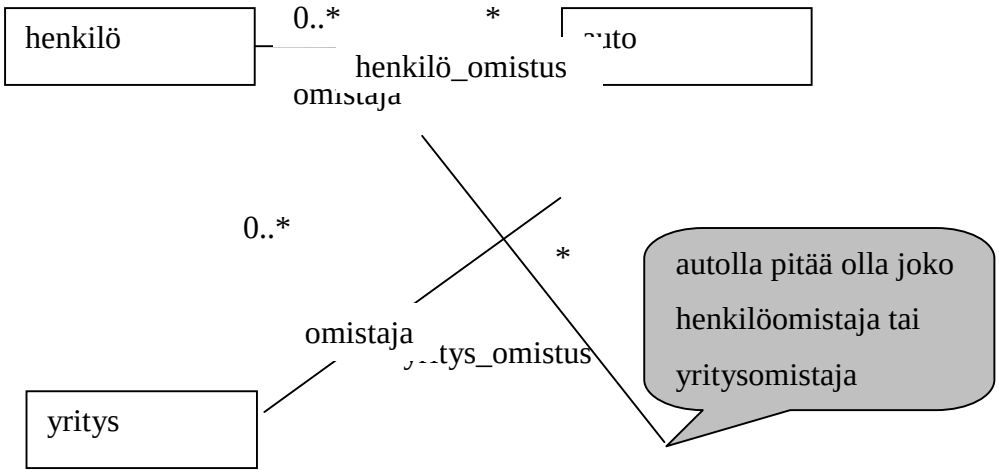
Kuva 4.40: Syrjäytetty palvelu viikkoraportti

Kohdealuetta mallinnettaessa yleistyshierarkian tärkein käyttötilanne on kuvausvaivan säästäminen ja erilaisten sääntöjen täsmällinen ilmaiseminen. Jos usealla luokalla on samoja attribuutteja ja luokille voidaan määritellä yhteinen yläluokka, voidaan yhteiset attribuutit liittää yläluokkaan ja määritellä näin vain kertaalleen. Sääntöjen täsmällisemmästä esittämisestä voidaan tarkastella esimerkkinä tilannetta, jossa on kuvattava auton omistusta. Auton omistajana voi olla henkilö tai yritys. Jokaisella

autolla täytyy olla omistaja. Yleistyshierarkiaa käyttäen asia voidaan esittää kuvan 4.41 mukaisesti. Tällöin sääntö saadaan esitettyä. Jos asia yritetään esittää ilman yleistyshierarkiaa, päädytään esimerkiksi kuvan 4.42 mukaiseen malliin. Siinä sääntö ei näy kaaviossa, vaan se on esitettävä erikseen tekstikuvauksessa (oheisesa kuvassa puhekuplana), jolloin se jää vähemmälle huomiolle.

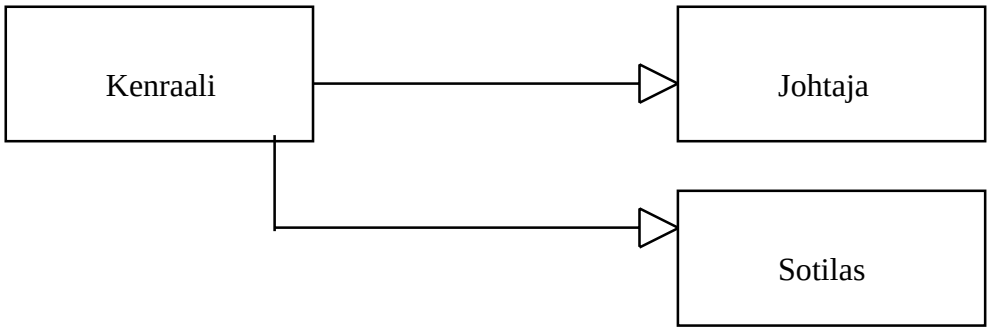


Kuva 4.41: Auton omistus yleistyshierarkiaa käyttäen, pakollinen omistus näkyy.



Kuva 4.42: Auton omistus ilman yleistyshierarkiaa, omistuksen pakollisuus on esitettävä erillisenä sääntönä.

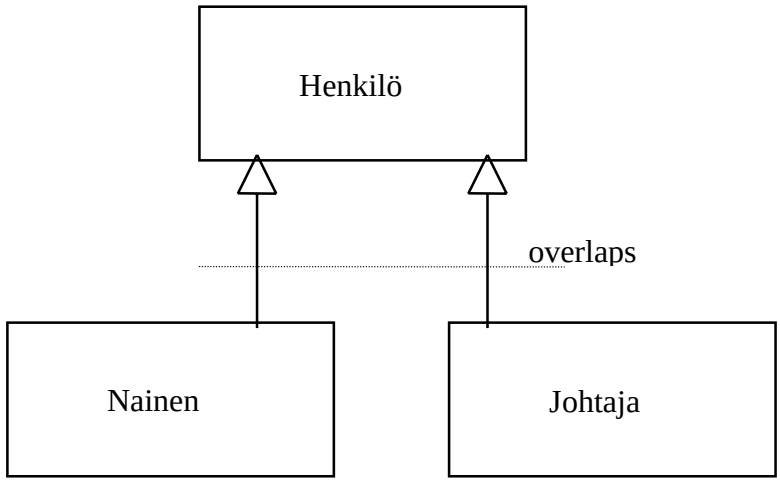
Tilannetta, jossa luokka on useamman kuin yhden luokan välitön alaluokka kutsutaan moniperiytymiseksi (multiple inheritance) (kuva 4.43).



Kuva 4.43: Moniperiytyminen.

Moniperiytyminen voi olla kätevä tapa kohdealueen mallinnuksessa. Kaikki olio-ohjelmointikielet, esimerkiksi Java, eivät kuitenkaan tue moniperintää,

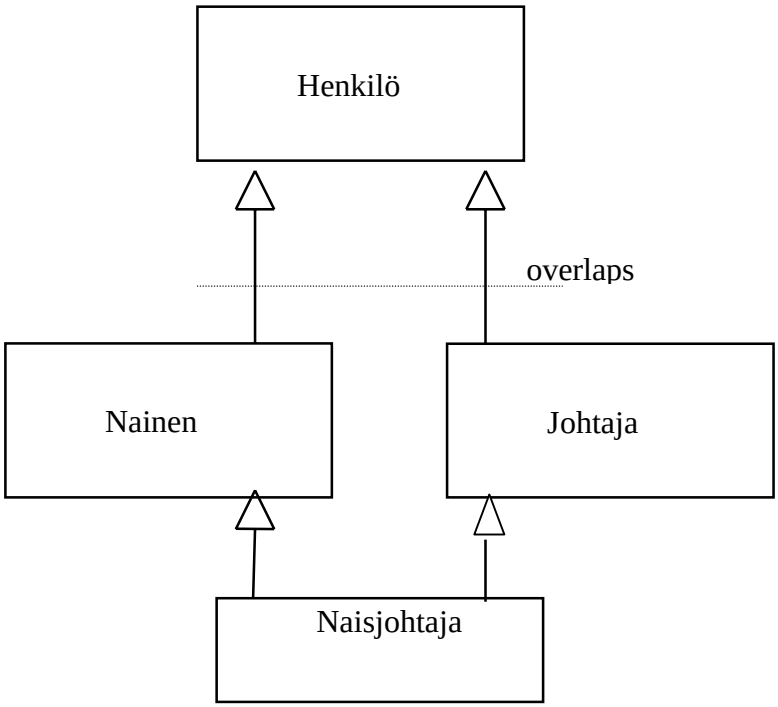
UML-kuvaustekniikassa voidaan antaa myös alaluokkajakoon liittyviä selvennyksiä ja rajoitteita. Esimerkkinä voidaan tarkastella henkilön alaluokkia nainen ja johtaja. Jos oletamme (toisin kuin olio-ohjelmointikielissä), että olio voi kuulua samanaikaisesti useaan alimman tason luokkaan, voitaisiin luokat nainen ja johtaja määritellä osittain päällekkäisiksi kuvan 4.44 mukaisesti.



Kuva 4.44: Osittain päällekkäiset alaluokat

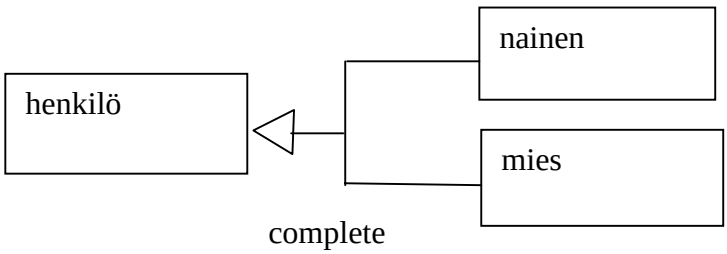
Kuvan 4.39 rakenteesta on syytä huomata, että se ei ole mahdollinen useimmissa olio-ohjelmointikielissä. Niissä olio voi tyypillisesti kuulua vain yhteen yleistys-hierarkian alimman tason luokkaan. Yllä oleva tilanne olisi tällöin hoidettava

lisäluokalla naisjohtaja (kuva 4.45). Tällainen rakenne taas on mahdollinen vain, jos moniperintä on mahdollista.



Kuva 4.45: Päällekkäisten luokkien hyväksikäyttö olio-ohjelmointikielissä.

UML:ssä toisensa poissulkevat samaan luokittelukriteeriin perustuvat alaluokat kuvataan yhteisellä kolmiokärjellä (kuva 4.46)



Kuva 4.46: Henkilön jako luokkiin nainen ja mies (lisämääre complete ilmaisee että henkilö on joko mies tai nainen eikä muuta).