Eija Järvelä
Elina Puusaari

UML–käsikirja

Jyväskylän yliopisto
Chydenius-instituutti – Kokkolan yliopistokeskus
Kokkola 2005

OHRA-projekti
Ohjelmistoarkkitehtuurien integrointi ohjelmistotuotantoon

Chydenius-instituutin selvityksiä 1/2005

ISBN 951-39-2153-0 (pdf)

SISÄLLYSLUETTELO

KIRJAN RAKENNE JA ESIPUHE

Tämä kirja on jatkoa syksyllä 2003 ilmestyneeseen UML-käsikirjaan. Paikallisilta ohjelmistoyrityksiltä pyydetyn UML-käsikirjaa koskevan palautteen perusteella tehtiin jatko-osa, joka keskittyy UML -kaavioista käyttötapaus-, luokka- ja sekvenssikaavioihin.

Luvussa 1 on esitelty lyhyesti kaikki kaaviotyypit ja mallinnuselementit. Mallinnuselementtien graafiset symbolit esitellään luvussa 5. Käsikirjan lopussa on lisäksi tärkeimpien käsitteiden hakemisto (luku 6) ja sanastot (luvussa 7 suomi – englanti ja luvussa 8 englanti – suomi).

Käsikirja on suomenkielinen, mutta kaikki termit on kirjoitettu myös englanniksi joko suluissa mainittaessa ensimmäistä kertaa tai kirjan lopussa sanasto-osassa. UML -notaatioiden määrittelyissä englannin kieli on suositeltavaa hyödynnettäessä käsikirjaa yritysympäristössä.

UML:n kaikki elementit ja kaaviot perustuvat oliosuuntautuneeseen ajattelumalliin. Käsikirjassa käsitellään oliosuuntautuneita rakenteita. Oliosanastosta (luvut 9 ja 10) voi tarvittaessa käydä tarkistamassa sanojen merkityksen.

Käsikirja etenee käyttötapauskaavion esittelystä luokkakaavioon ja sekvenssikaavioon. Esimerkit ovat yksinkertaisia ja lyhyitä. Luokkakaavioesimerkeistä generoitu koodi on tehty Visual UML:n versiolla 4.1. Ohjelmointikielinä ovat C++ ja muutamassa esimerkissä Java. Koodissa luokat ja esimerkkiä kuvaavat kohdat on piirretty lihavoituina.

Laajana esimerkkitapauksena on työnjohtojärjestelmä, joka etenee käyttötapaus-, luokka- ja sekvenssikaavioiden esittelyinä kaaviokuvina sekä sanallisesti.

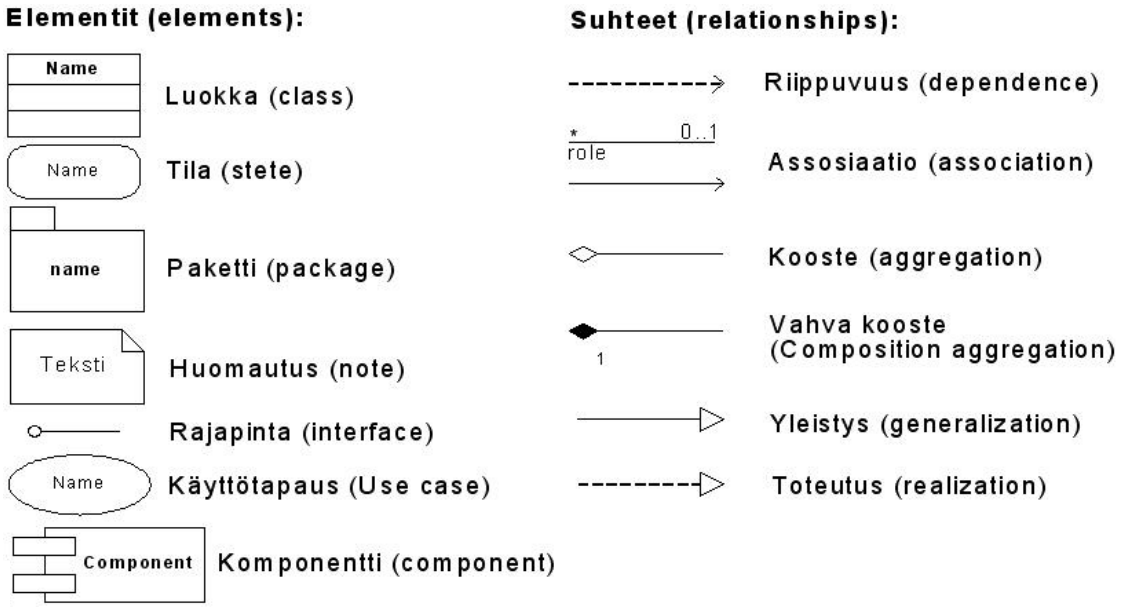
Kirja on kirjoitettu Microsoft Wordilla ja UML -kaaviot on piirretty Microsoft Visiolla.

Lähdeluettelossa olevista lähteistä kirjassa on käytetty pääasiassa teoksia UML (Eriksson, Hans-Erik & Penker, Magnus. 2000) sekä Oliokirja (Koskimies, Kai. 2000).

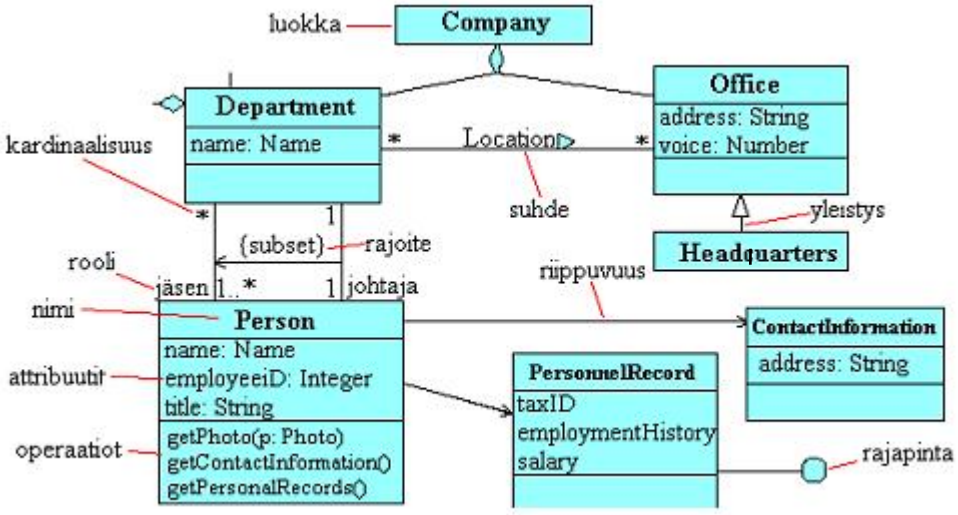
Käsikirja noudattaa rakenteeltaan ja osittain sisällöltään v. 2003 ilmestynyttä UML-käsikirjaa, jonka tekijöinä olivat Elina Puusaari ja Tommi Viitasaari. UML-käsikirjan jatko-osaa tekivät Eija Järvelän lisäksi Tommi Viitasaari ja Juha Himanka.

1. JOHDANTO

Yhtenäistetty mallinnuskieli, **Unified Modeling Language** (UML) v.1.0, julkaistiin v. 1997. **UML -kaaviot** (UML diagrams) ovat näkymän sisältöä kuvaavia kuvioita. Kaavioissa käytetään **mallinnuselementtejä** (model elements), jotka vastaavat yleisiä olio-ohjelmoinnin käsitteitä, kuten luokkia, olioita ja viestejä. Niiden väliset yhteydet ovat **assosiaatioita** (association), **riippuvuuksia** (dependency) ja **yleistyksiä** (generalization). Mallinnuselementille on aina määritelty *semantiikka*, eli muodollinen määritelmä, joka kertoo sen tarkan merkityksen yksikäsitteisesti. Mallinnuselementillä on sitä vastaava *näkymäelementti*, joka on se graafinen symboli, jota käytetään kaaviossa esittämässä mallinnuselementtiä (Kuva 1.1 ja Kuva 1.2).



Kuva 1.1 Mallinnuselementtejä.



Kuva 1.2 Esimerkki mallinnuselementeistä luokkakaaviossa.

Yleisten merkintöjen (general mechanisms) avulla mallinnuselementtiin voidaan liittää kommentteja, lisätietoja tai **lisämerkityksiä** (semantics). Niiden avulla UML:ää voidaan laajentaa sopimaan tietyille työmenetelmille, toimintaohjeille, yrityksille tai käyttäjille. Tyypillisimpiä yleisiä merkintöjä ovat **koristeet** (adornments) ja **huomautukset** (notes). Koristeilla voidaan erottaa tyyppi ilmentymästä (vahvennettu/alleviivattu) tai määritellä **kerrannaisuus** (multiplicity) assosiaatiosta. Huomautuksilla voidaan lisätä tietoa, jota ei muuten voida esittää UML -kielessä. Huomautuksilla voi olla **stereotyyppejä** (stereotypes), jotka kuvaavat huomautuksen tyypin.

1.1 Kaaviot

Kaaviot ovat kuvioita, joissa mallinnuselementit on järjestetty kuvaamaan tiettyä järjestelmän piirrettä. Kaavio kuuluu tiettyyn näkymään, johon se liitetään yleensä piirrosvaiheessa. Jotkut kaaviot voivat kuulua useaankin näkymään. UML:ssä on määritelty seuraavat kaaviot:

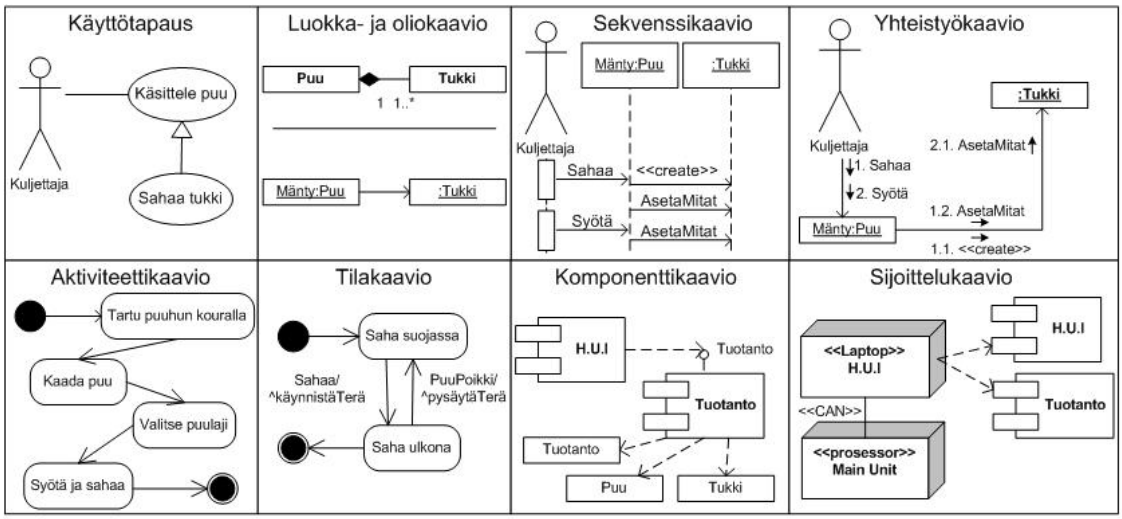
1. *Käyttötapauskaavio* (use case diagram)
   - näyttää järjestelmän toimijat, käyttötapaukset ja niiden väliset suhteet kuvaamalla toimijoiden ja järjestelmän vuorovaikutusta.
2. *Luokkakaavio* (class diagram)
   - kuvaa järjestelmän pysyvät rakenteet luokkien ja niiden välisten suhteiden avulla ja määrittää perustan muille kaavioille.
3. *Oliokaavio* (object diagram)
   - näyttää yksittäisten luokkien ilmentymien ja niiden välisten yhteyksien tilanteen jollakin hetkellä kuvaten otoksen kuvattavan järjestelmän yhdestä tilasta.
4. *Tilakaavio* (state diagram)
   - kuvaa olioiden, järjestelmien ja alijärjestelmien elinkaaria, sekä kertoo olioiden mahdolliset tilat ja erilaisten tapahtumien vaikutukset näihin tiloihin ajan kuluessa.
5. S*ekvenssikaavio* (sequence diagram)
   - kuvaa olioita ja niiden välisiä suhteita, oliojoukon viestintää, tietyssä tilanteessa, sekä mitä avustavia olioita tarvitaan tilanteen suorittamiseen ja mitä ja missä järjestyksessä olioiden palveluita käytetään.
6. *Yhteistyökaavio* (collaboration diagram)
   - kuvaa olioiden välistä vuorovaikutussuhdetta keskittyen tilaan, kun taas rinnakkaiskaavio sekvenssikaavio keskittyy kuvaamaan aikaa.
7. *Toimintokaavio* (activity diagram)
   - on tilakaavion erikoistapaus, joka kuvaa tietyn tehtävän sisäisen logiikan ja kuhunkin tilaan liittyvät mahdolliset operaatiot, sekä ehdolliset tilasiirtymät ja kertoo mitä pitää olla tehtynä ennen siirtymistä seuraavaan vaiheeseen.
8. *Komponenttikaavio* (component diagram)
   - kuvaa komponentit ja niiden väliset suhteet, sekä esittää koodin rakenteen sisältäen mahdollisesti komponenttien lisäksi koostesuhteita ja rajapintoja.
9. *Käyttöönottokaavio* (deployment diagram)

- on järjestelmän rakenteen kuvaus, joka esittää suoritusaikaiset prosessointielementit, niiden yhteydet ja ohjelmistojen osien sijoittumisen niihin. Käyttöönottokaavio kuvaa sekä laitteistoyksiköiden rakenteen että joka yksikössä toimivan ohjelmiston.

Kaavio	Mitä kaavio kertoo?
Käyttötapaus	Prosessi jaetaan käyttäjien kannalta järkeviin palasiin.
Sekvenssi	Missä järjestyksessä tapahtumat esiintyvät objektien välillä?
Yhteistyö	Kuinka eri objektit ovat vuorovaikutuksessa keskenään?
Luokka	Mitkä ovat asioiden ts. luokkien väliset staattiset suhteet?
Tila	Kuinka yksittäinen objekti muuttuu eri vaiheiden myötä?
Toiminto	Kuinka yksi tai useampi objekti käyttäytyy tai kuinka jokin sovellus etenee? Myös rinnakkaisen toiminnan kuvaaminen on mahdollista.

Taulukko 1.1 Yleisimpien kaavioiden käyttötarkoituksia

Kuva 1.3 UML kaavioiden kuvausvoima.



Kuva 1.4 Esimerkki eri kaavioiden liittymisestä toisiinsa

Tässä oppaassa keskitytään UML-kaavioista vain käyttötapaus-, luokka- ja sekvenssikaavioihin, sekä luokkakaavioiden esimerkkitapausten esittämiseen koodimuodossa.

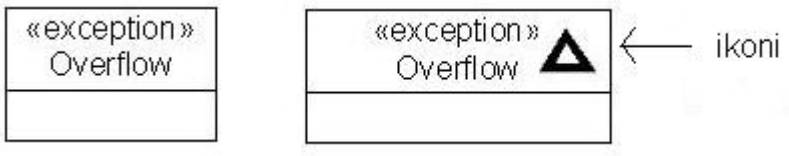
1.2 UML:n laajentaminen

UML:ssä on mahdollisuus laajennuksiin joitakin tiettyjä menetelmiä, organisaatioita tai käyttäjiä varten. Laajennuksilla lisätään olemassa oleviin UML-elementteihin lisämerkityksiä. Laajennusmekanismeista yleisimmät ovat nimetyt arvot, rajoitukset ja stereotyypit.

1.2.1 Stereotyypit

Stereotyypeillä voidaan elementteihin antaa lisämerkitystä. UML määrittelee yli 40 ns. standardistereotyyppejä (esim. actor, interface, parameter, use, jne.) mutta käyttäjä voi antaa myös omia stereotyyppejä. Stereotyyppi kirjoitetaan merkkijonona **kaksoiskulmasulkujen** (guillements) sisään elementin nimen viereen: <<stereotyypin nimi>>. Stereotyypille voidaan määritellä myös oma ikoni (Kuva 1.5).

Kuvassa 1.5 Overflow on luokka, jolla on stereotyyppi <<exception>>.

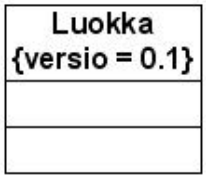


Kuva 1.5 Stereotyyppi

1.2.2 Nimetyt arvot

Nimetyt arvot (tagged values) ovat UML:n elementteihin kiinnitettyjä ominaisuuksia, joilla lisätään niihin tietoa. UML:ssä on valmiina joukko näitä ominaisuuksia, mutta käyttäjä voi myös itse määritellä omia ominaisuuksia. Kaavioissa ominaisuudet merkitään aaltosulkujen sisään lähelle ko. elementtiä.

Kuva 1.6 luokalla on ominaisuutena käyttäjän itse määrittelemä nimetty arvo.



Kuva 1.6 Esimerkki nimetystä arvosta.

1.2.3 Rajoitteet

Rajoitteet (constraint) rajoittavat joko elementin käyttöä tai sen merkitystä. UML:ssä on 18 standardoitua rajoitusta: *global, local, parameter, self, vote, broadcast,t complete, disjoint, incomplete, overlapping, implicit, or, ordered, new, destroyed* ja *transient* Rajoitteita voidaan määrätä myös itse. (Ks. 3.4.2.2 ja 3.4.4)

Kuvassa (Kuva 1.7) rajoitus rajaa, mitkä Henkilö-oliot voivat osallistua assosiaatioon.

Kuva 1.7 Esimerkki rajoituksen käytöstä.

1.3 Esimerkkijärjestelmä

Käsikirjan laajana esimerkkitapauksena on ”Työnjohtojärjestelmä”. Kyseessä on LVI-asennuksia tekevä yritys, jonka palveluksessa ovat pääasiassa konttorissa työskentelevä työnjohto, sekä kentällä toimivat kymmenet asentajat. Toimeksiantojen mukaan työnjohto pilkkoo tehtävät ja jakaa ne asentajille. Tätä työtä varten tarvitaan tietokonepohjainen työnjohtojärjestelmä.

Järjestelmään kuuluvat yrityksen tiloissa oleva palvelin ja työntekijöillä olevat omat päätteet (esim. PDA -laitteet). Palvelimella on puhelinverkkoyhteys kentän koneisiin. Työnjohto saa puhelimitse asennustyöpyynnön, kirjaa sen määrämuotoisena järjestelmään ja jakaa toimeksiannon suppeammiksi tehtäviksi. Työnjohdon tulee tietää jo sovitut ja kiinnitetyt toimeksiannot aikataulutietoineen, sekä kunkin työntekijän aikataulut ja asiantuntemus kiinnittääkseen suoritusajankohdat. Uusista työtehtävistä tulee palvelimen kautta tieto työntekijälle, kun hän ottaa yhteyttä palvelimeen.

Kun työtehtävä on tehty, työntekijä päivittää aikataulunsa ja lähettää tehtäväkohtaisesti eriteltyinä laskutustiedot (työtunnit, tarvikkeet) kannettavansa kautta palvelimelle. Hän päivittää mahdolliset viivästykset samalla aikatauluunsa. Työnjohto viimeistelee laskut laskutustietojen perusteella.

Järjestelmän avulla voidaan selvittää kunkin työntekijän senhetkinen työtehtävä ja kohdepaikka, jotta kiireellisen hälytyksen tehtävä voidaan osoittaa lähimmälle vapaalle työntekijälle. Kuukausittainen seurantaraporttien laatiminen erilaisten ryhmittelyjen ja rajausten mukaisesti voidaan myös toteuttaa järjestelmällä.

2. KÄYTTÖTAPAUSMALLINNUS

Käyttötapausmallinnus (use case modeling**)** on *Ivar Jacobsonin* kehittämä tekniikka, jolla kuvataan, *mitä* järjestelmän *tulisi tehdä* tai *mitä* se *tekee*. Järjestelmä on ”**musta laatikko**” (black box), joka tarjoaa käyttötapausten esittämät toiminnot ulkoisille toimijoille. Se ei kerro toimintatapaa eli *miten* järjestelmä *tekee*. Kyseessä on iteratiivinen työtapa, johon kuuluu neuvotteluja asiakkaan ja toimijoita vastaavien henkilöiden kanssa.

Käyttötapausmallin tärkeimmät osat ovat **käyttötapaukset** (use cases), **toimijat** (actor) ja **mallinnettava järjestelmä** (system). Järjestelmän rajat määritellään sen **toimintojen** (action) mukaan. Toiminnot kuvataan käyttötapauksilla, joista jokainen kuvaa yhden täydellisen toiminnon.

Käyttötapausmallin tekemiseen kuuluu järjestelmän määrittely, toimijoiden ja käyttötapausten löytäminen, käyttötapausten kuvailu, käyttötapausten välisten suhteiden määrittely ja lopulta mallin hyväksyntä.

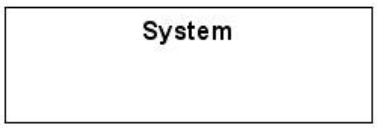
Käyttötapausmalli tarjoaa järjestelmän **käyttötapausnäkymän** (use case view), joka vaikuttaa kaikkiin muihin järjestelmän näkymiin. Sekä looginen että fyysinen arkkitehtuuri toteuttavat nimenomaan käyttötapausmallin määrittämät toiminnot. Käyttötapausmalli toimii ohjelmiston kehityksen perustana.

2.1 Käyttötapauskaavio

Käyttötapausmalli kuvataan UML:ssä **käyttötapauskaaviona** (use case diagram), joka näyttää järjestelmän toimijat, käyttötapaukset ja niiden väliset suhteet. Se kuvaa siis toimijoiden ja järjestelmän välistä vuorovaikutusta. Seuraavissa kappaleissa kuvataan käyttötapauskaavion mallinnuselementtejä, sekä niiden välisiä yhteyksiä ja riippuvuuksia.

2.2 Järjestelmä

Käyttötapausmallinnuksen aikana **järjestelmän** (system) rajat kehitetään ja määritellään. Olennaista on kerätä luettelo keskeisimmistä asioista eli **entiteeteistä** (entities), sekä niille sopivista termeistä ja määritelmistä. Nämä termit kuvailevat käyttötapaukset. Järjestelmä kuvataan käyttötapauskaaviossa laatikkona, jonka yläpuolella tai sisällä on järjestelmän nimi (Kuva 2.1). Laatikossa ovat myös järjestelmän käyttötapausten symbolit. (Ks. kuva 2.8)



Kuva 2.1 Järjestelmä.

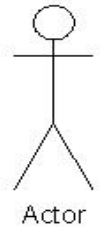
2.3 Toimijat

Toimija (actor) kommunikoi järjestelmän kanssa, ts. on järjestelmän käyttäjä. Toimija on käynnistäjä, joka voi olla ihminen tai toinen järjestelmä, johon järjestelmästä on yhteys tai jokin järjestelmän kanssa kommunikoiva laite.

Toimija on **tyyppi** eli luokka. Se vastaa jotakin roolia, ei yksittäistä järjestelmän käyttäjää. Toimijalla on **nimi**, jonka tulisi vastata toimijan roolia.

Toimijoita voivat olla **päätoimija** (primary actor) sekä **toissijainen toimija** (secondary actor). Käyttötapauksen voi käynnistää *aktiivinen toimija* ja osallistujana on *passiivinen toimija*.

Käyttötapauskaavioissa käytetään toimijoista yleensä standardoitua tikku-ukon kuvaketta, jonka alapuolella on toimijan nimi (Kuva 2.2).



Kuva 2.2 Toimija

2.3.1 Toimijoiden löytäminen

Toimijat löydetään miettimällä vastauksia seuraaviin kysymyksiin:

- Kuka käyttää järjestelmän päätoimintoja (päätoimija)?
- Kuka tarvitsee järjestelmältä apua päivittäisten tehtäviensä suorittamiseen?
- Kuka tarvitaan ylläpitämään, hallinnoimaan ja huoltamaan järjestelmää (toissijaiset toimijat)?
- Mitä laitteita järjestelmän on käsiteltävä?
- Minkä muiden järjestelmien kanssa järjestelmän on toimittava?
- Ketä tai mitä kiinnostaa järjestelmän tuottamat tulokset?

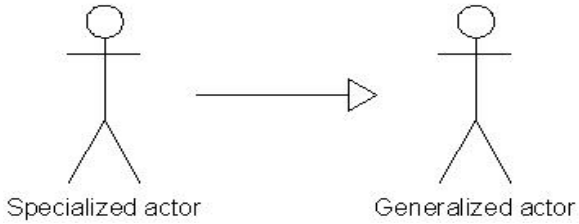
Käyttäjä voi siis olla kuka tai mikä tahansa, joka suoraan tai epäsuorasti on vuorovaikutuksessa järjestelmän kanssa ja käyttää järjestelmän palveluja jonkin asian suorittamiseksi. Samalla käyttäjällä voi olla myös eri rooleja eri aikoina, riippuen järjestelmässä kulloinkin käytössä olevista toiminnoista.

2.3.2 Toimijoiden väliset suhteet

Toimijoiden välisiä yhteisiä toimintoja kuvataan käyttötapauskaavioissa vain yleistyssuhteilla.

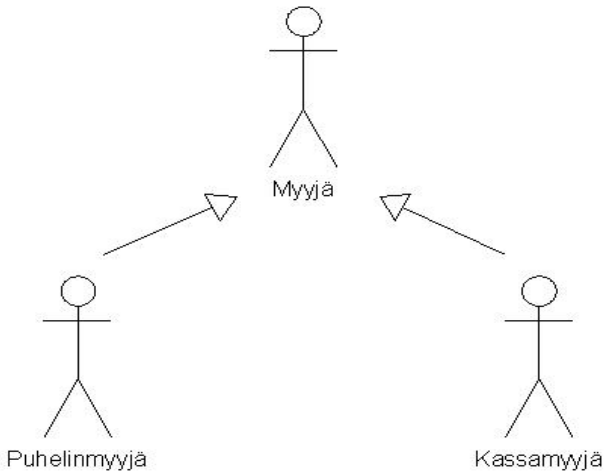
Yleistys (generalization): Kun useat toimijat oman roolinsa osana toimivat yleisemmässä roolissa, kuvataan se yleistyksellä. Tällöin yleisen roolin toiminnot voidaan kuvata toimijayliluokassa, jolloin erikoistuneet toimijat perivät yliluokan toiminnallisuuden ja laajentavat sitä jotenkin.

Toimijoiden välinen yleistys piirretään viivana, jonka yliluokan puoleisessa päässä on ontto kolmio (Kuva 2.3).



Kuva 2.3 Yleistys

Kuvassa (Kuva 2.4) toimijakantaluokka Myyjä kuvailee yleisen roolin, johon kuuluvat Puhelinmyyjä ja Kassamyyjä. Käyttötapaukset, jotka eivät välitä myyjän laadusta, voivat käyttää yleistä toimijaa, kun taas myyntitapaa korostavat käyttötapaukset toimivat sopivan toimija-aliluokan kanssa.



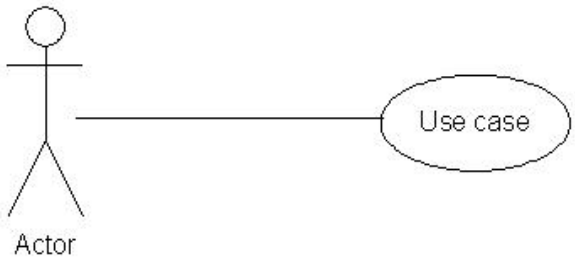
Kuva 2.4 Esimerkki yleistyksestä

2.4 Käyttötapaukset

Käyttötapaus kuvaa yhtä loogista käyttäjän toimintatilannetta järjestelmässä. UML:ssä se määritellään sarjana järjestelmän suorittamia toimintoja, joiden tuloksena saadaan tietty toimijan havaitsema tulos.

Käyttötapauksia voidaan käyttää hyväksi asiakasvaatimusten konkretisoinnissa ja kommunikoinnissa asiakkaan kanssa, järjestelmän toiminnallisen määrittelyn työstämisessä, käyttöohjeiden laatimisessa, järjestelmätestauksessa jne.

Käyttötapaukset kuvataan UML:ssä ellipsinä, jonka sisällä tai alapuolella on käyttötapauksen nimi. Käyttötapaus sijoitetaan yleensä järjestelmän rajojen sisäpuolelle ja se on yhteydessä toimijaan joko yhteydellä tai viestiyhteydellä, joka näyttää käyttötapauksen ja toimijan välisen kommunikointitavan.



Kuva 2.5 Assosiaatio toimijan ja käyttötapauksen välillä

2.4.1 Käyttötapausten piirteet

Käyttötapauksen käynnistää aina toimija: Käyttötapaus suoritetaan aina jollekin toimijalle ja toimijan on suoraan tai epäsuorasti käskettävä järjestelmää käynnistämään käyttötapaus.

Käyttötapaus tarjoaa toimijalle jotain: Käyttötapauksen on toimitettava käyttäjälle jokin selvä hyöty tai etu.

Käyttötapaus on täydellinen: Käyttötapauksen on oltava täydellinen kuvaus. Yleisin virhe on jakaa käyttötapaus moneksi pieneksi käyttötapaukseksi. Käyttötapaus ei ole täydellinen ennen kuin lopputulos on tuotettu.

Käyttötapaukset kytketään toimijoihinsa *yhteyksillä*, joita kutsutaan **viestiassosiaatioiksi** (communication associations). Assosiaatio on yleensä **yksi yhteen – suhde** (one-to-one) ilman suuntaa, jolloin yksi toimijailmentymä kommunikoi yhden käyttötapausilmentymän kanssa molemminsuuntaisesti.

Käyttötapaus on luokka, ei ilmentymä. Se kuvaa toimintoa kokonaisuutena, mukaan lukien mahdolliset vaihtoehdot, virheet ja poikkeukset, joita voi tapahtua käyttötapauksen suorituksen aikana. Käyttötapauksen ilmentymä vastaa yhtä järjestelmän oikeaa käyttöä ja sitä kutsutaan **tilanteeksi** (scenario).

Esimerkkeinä tilausten käsittelyn eri skenaarioista voisivat olla normaali tilaus, luottorajan ylitys ja tuotteen loppuminen.

2.4.2 Käyttötapausten löytäminen

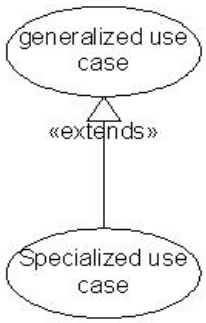
Käyttötapausten etsiminen aloitetaan aikaisemmin löydetyistä toimijoista kysymällä niiltä seuraavat kysymykset:

- Mitä järjestelmän toimintoja toimija tarvitsee?
- Mitkä ovat toimijan tehtävät järjestelmän käytössä? Täytyykö toimijan lukea, luoda, tuhota, muokata tai tallentaa jotain tietoa järjestelmässä?
- Täytyykö toimijan tietää joistakin järjestelmän tapahtumista vai täytyykö sen ilmoittaa niistä järjestelmälle?
- Voisiko toimijan päivittäistä työtä helpottaa tai tehostaa uusilla toiminnoilla?

2.4.3 Käyttötapausten väliset suhteet

Käyttötapausten välillä on kolmenlaisia suhteita: *laajennus-, käyttö- ja ryhmittelysuhteet*.

- **Laajennussuhde** (extends association) on yleistyssuhde, jossa yksi käyttötapaus laajentaa yleistä käyttötapausta lisäämällä siihen toimintoja. Laajentavassa käyttötapauksessa voi olla laajennettavan käyttötapauksen toimintoja, riippuen laajennuksen ehdoista.
  Laajennussuhde kuvataan käyttötapausten välillä yleistyksenä, eli viivana, jonka laajennettavan käyttötapauksen päässä on ontto kolmio ja jolla on stereotyyppi <<extends>> (Kuva 2.6).

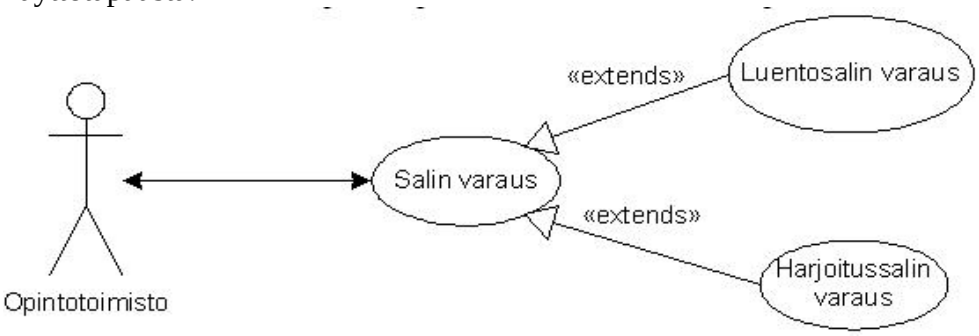


Kuva 2.6 Laajennussuhde.

- **Käyttösuhde** (uses association) on yleistyssuhde, jossa yksi käyttötapaus käyttää toista siten, että erikoistuneeseen käyttötapaukseen liittyy osana yleisen käyttötapauksen toiminnot.
  Käyttösuhde kuvataan yleistyssuhteella eli viivalla, jonka käytettävän käyttötapauksen päässä on ontto kolmio ja jolla on stereotyyppi <<uses>> (vrt. Kuva 2.6)

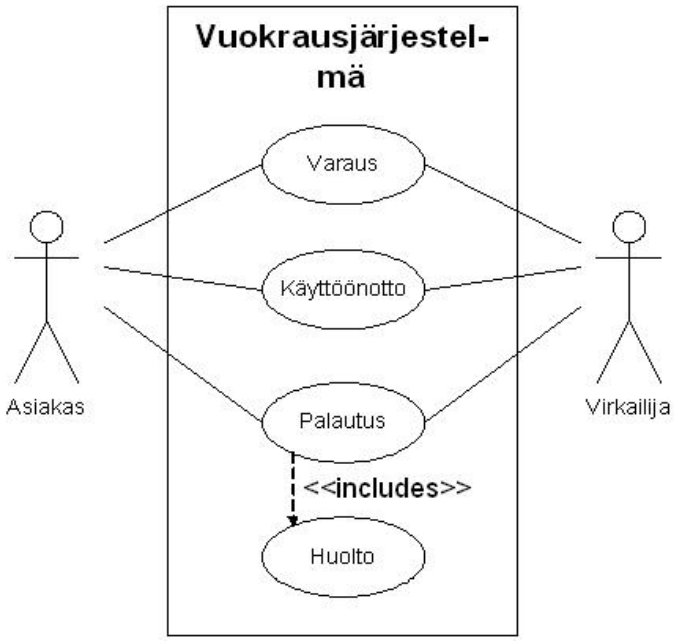
- **Ryhmittelysuhteessa** joukko käyttötapauksia käsittelee samantapaista toiminnallisuutta, tai liittyy toisiinsa jollakin tavalla. Tällöin ne voidaan ryhmitellä UML – pakettiin (ks. 3.6).

Kuvassa (Kuva 2.7) luento- ja harjoitussalin varaus laajentavat salin varaus –käyttötapausta.



Kuva 2.7 Esimerkki laajennussuhteesta

Kuvassa (Kuva 2.8) stereotyyppi <<includes>> tarkoittaa, että käyttötapauksen suorittaminen edellyttää myös toisen käyttötapauksen suorittamista. (Vastaa << uses >> -suhdetta.)



Kuva 2.8 Esimerkki käyttötapauskaavioista

2.4.4 Käyttötapausten kuvailu

Käyttötapausten kuvaus, joka on yksinkertainen määritelmä toimijoiden ja käyttötapausten vuorovaikutuksesta, tehdään usein sanallisesti. Siinä käytetään sitä kieltä ja termistöä, jota järjestelmän asiakas tai käyttäjä käyttää.

Käyttötapausten kuvaus kertoo ulkoisen toimijan näkökulmasta mitä tehdään, ei järjestelmän sisäistä toimintaa.

Sanalliseen kuvaukseen kuuluu:

- Käyttötapauksen tarkoitus, eli lopulliset tavoitteet
- Tieto käyttötapauksen käynnistyksestä, eli käyttötapauksen suorittamisen aloitus ja tilanne.
- Viestin kulku toimintojen ja käyttötapauksen välillä.
- Käyttötapauksen vaihtoehtoiset suoritukset, riippuen ehdoista tai poikkeustilanteista, sekä erityisten virheenkäsittelyjen kuvaukset tilanteina.
- Tieto käyttötapauksen valmistumisesta ja tuloksen palautus toimijalle.

Sanallisessa kuvauksessa kerrotaan siis kuka tekee, mitä tekee, miten tekee, millä tekee, koska tekee ja kuinka usein tekee jne.

Käyttötapausten kuvaamisten jälkeen voidaan selvittää, onko aikaisemmin kuvatut suhteet tunnistettu.

2.4.5 Käyttötapausten testaaminen ja esittäminen

Myös testausvaiheessa hyödynnetään käyttötapauksia kahdella erityyppisellä testillä:

- **Tarkistuksessa** (verification) varmistetaan, että järjestelmä on tehty oikein ja määritysten mukaan. Sitä ei siis voida tehdä ennen kuin järjestelmässä on toimivia osia.

- **Hyväksynnässä** (validation) varmistetaan, että kehitettävä järjestelmä on se, jota asiakas todella haluaa. Hyväksyntä tehdään heti kehittelyvaiheen alussa ja esitetään asiakkaille, jopa käyttötapausmallin kehityksen aikana.

Käyttötapausten tarkistamisessa voidaan käyttää myös **käyttötapausten esittämistä** (walking the use case), jolloin mallinnusryhmän eri henkilöt roolipelaavat toimijoita ja järjestelmää tietyssä käyttötapauksessa. Roolipelin ulkopuolella olevat suunnittelijat tekevät muistiinpanoja ja yrittävät löytää puutteita esitetyistä käyttötapauksista.

2.4.6 Käyttötapausten toteuttaminen

Käyttötapaukset ovat toteutuksesta riippumattomia kuvauksia järjestelmän toiminnoista, jotka **toteutetaan** (realize) järjestelmään. Vastuu toimintojen suorituksesta jaetaan yhteistyössä toimiville olioille, jotka yhdessä toteuttavat toiminnon.

UML:n periaatteet käyttötapausten toteutukselle ovat:

- Käyttötapaus toteutetaan **yhteistyönä** (collaboration), joka näyttää käyttötapauksen sisäisen, toteutusriippuvaisen ratkaisun luokkina tai olioina ja niiden välisinä suhteina. Yhteistyön symboli on ellipsi, jonka sisällä on yhteistyön nimi.
- UML:ssä *yhteistyö kuvataan useina kaavioina*, jotka näyttävät yhteistyön ympäristön ja vuorovaikutuksen. Yhteistyöhön osallistuu joukko luokkia ja

yhteistyön ilmentymässä joukko olioita. Mahdolliset kaaviot ovat yhteistyö-, viestiyhteys- ja toimintokaaviot käyttötapauksesta riippuen.

- *Tilanne* on käyttötapauksen tai yhteistyön ilmentymä, joka vastaa käyttötapauksen yhtä käyttöä. Käyttötapauksen kohdalla vain toimijoiden näkemä ulkoinen toiminnallisuus kuvaillaan. Yhteistyön ilmentymässä kuvataan yhteistyöhön liittyvät luokat, niiden operaatiot ja viestinnät.

Käyttötapauksen toteutuksessa muutetaan sanallisen tai toimintokaaviona kuvatun käyttötapauskuvauksen eri askel ja toiminnot luokiksi, luokkien operaatioiksi ja luokkien välisiksi suhteiksi. Tätä kutsutaan eri toimintojen vastuun jakamiseksi käyttötapauksen toteuttavaan yhteistyöhön osallistuville luokille.

Käyttötapauksen askel (toiminto) muutetaan yleensä useaksi operaatioksi. Luokka voi osallistua useaan käyttötapaukseen ja luokan kokonaisvastuu onkin sen kaikkien käyttötapausroolien yhdistelmä.

Käyttötapauksen ja sen yhteistyötoteutuksen välinen suhde näytetään joko **toteutussuhteella** (realization relationship), joka kuvataan katkoviivana, jonka toisessa päässä on nuoli ja jolla on stereotyyppi <<realizes>>, tai esim. hyperlinkillä käyttötapauskaaviosta sen toteuttamaan yhteistyökaavioon.

Onnistunut vastuunjako luokkien välillä vaatii kokemusta ja iteratiivisuutta. Jacobson (Booch, Jacobson, Rumbaugh. 1999) käyttää työmenetelmää, joka määrittää *kolme stereotyyppioliota:*

- **Rajaoliot** (boundary objects): Järjestelmän sisällä oleva oliotyyppi, joka on lähellä järjestelmän rajaa. Se välittää viestejä ulkoisten toimijoiden ja järjestelmän muiden olioiden välillä.

- **Hallintaoliot** (control objects): Oliotyyppi, joka hallitsee olioryhmän sisäisiä vuorovaikutuksia.

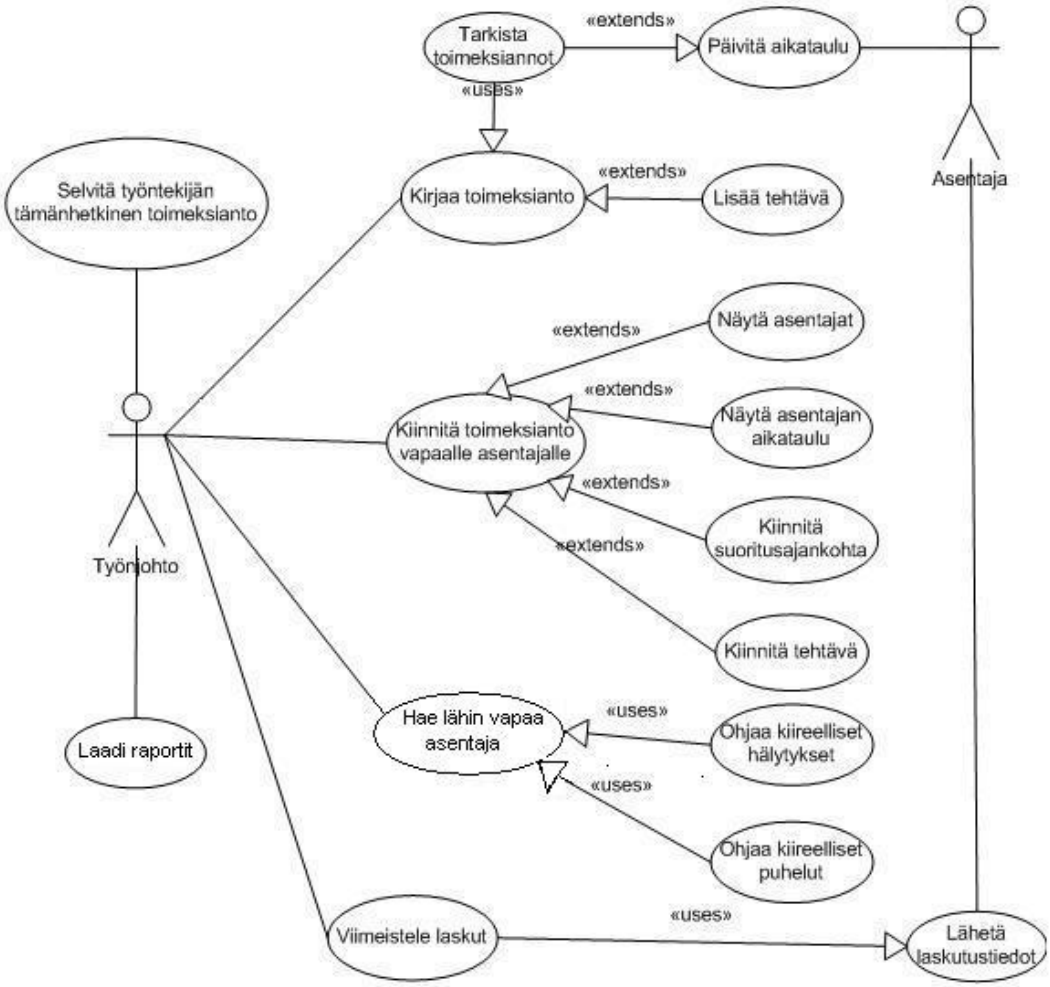
- **Tieto-oliot** (entity objects): Oliotyyppi, joka vastaa jotakin asiaa järjestelmän käsittelemällä alueella. Se on yleensä passiivinen.

Kun erityyppiset oliot on määritelty ja yhteistyöt muodostettu, voidaan etsiä niiden välisiä yhtäläisyyksiä, jotta osaa luokista voidaan hyödyntää useissa käyttötapauksissa.

2.5 Esimerkkijärjestelmä

Työnjohtojärjestelmän käyttötapauskuvauksessa toimeksianto on jaettu yhdeksään käyttötapaukseen: kirjaa toimeksianto, lisää tehtävä, kiinnitä toimeksianto vapaalle asentajalle, viimeistele laskut, laadi raportit, selvitä työntekijän tämänhetkinen toimeksianto, hae lähin vapaa asentaja, päivitä aikataulu sekä lähetä laskutustiedot.

Kuvassa (Kuva 2.9) on esitelty työnjohtojärjestelmän käyttötapauskuvaus. Käsikirjan liitteenä (Liite 1) on sanallinen kuvaus jokaisesta em. yhdeksästä käyttötapauksesta.



Kuva 2.9 Työnjohtojärjestelmä; Use Case Diagram

3. LUOKAT, OLIOT JA SUHTEET

Oliosuuntautuneen mallinnuksen tärkeimmät mallinnuselementit ovat luokat, oliot ja niiden väliset suhteet.

3.1 Luokat ja oliot

Olio (object) on ympäristöstään erottuva kokonaisuus. Sillä on oma identiteetti, sisäinen rakenne, sekä suhteet tiettyyn ympäristöön. Se on asia, josta voidaan keskustella ja jota voidaan käsitellä. Se ei ole pelkästään toiminnallinen tai tietoa säilyttävä, vaan se sisältää molemmat aspektit.

Luokka (class) on kuvaus olion tyypistä. Se kuvaa yhden tyypin olioiden ominaisuudet ja käyttäytymisen. Olioita luodaan luokista, joten oliot ovat jonkin luokan **ilmentymiä** (instance). Olio on yksittäinen luokan edustaja. Luokka voi olla kuvaus oliosta missä tahansa järjestelmässä.

Esimerkkejä luokista voivat olla esim. Asiakas, Sopimus, Lasku, Moottori, Painonappi, Tiedosto, Laite, Kuvake jne.

Em. Asiakas-luokan olioita voisivat olla esimerkiksi Kalle Koehenkilö, Erkkilä Väinö, Saaranen Jutta, Palomaa Juuso jne.

3.2 Luokkakaavio

Luokkakaavio (class diagram) on staattinen mallinnustyyppi. Se kuvaa järjestelmän pysyvät rakenteet luokkien ja niiden välisten suhteiden avulla. Luokkakaavio määrittää perustan muille kaavioille. Luokkakaavion luokka voidaan toteuttaa suoraan oliosuuntautuneella ohjelmointikielellä, joka tukee suoraan luokkarakennetta.

Luokka piirretään UML – kielessä suorakaiteena, joka on jaettu kolmeen lohkoon: **nimi-** (name), **attribuutti-** (attribute), ja **operaatio**-osastoon (operation). Luokka voidaan piirtää myös suorakaiteena, jossa on pelkkä nimiosasto (Kuva 3.1). Kuvassa (Kuva 3.2) on esimerkkejä luokasta.

Kuva 3.1 Luokan esittämistavat

3.2.1 Nimiosasto

Luokan nimi kirjoitetaan ylimpään laatikkoon *keskitettynä* ja *vahvennettuna*. Nimenä käytetään yleensä substantiivia ja sen tulisi olla sekä kuvaava että mahdollisimman yksikäsitteinen (esim. lasku tai velka). Nimessä ei pitäisi käyttää etu- tai takaliitteitä. Nimiosastoon voidaan kirjoittaa myös luokkaa koskevia rajoitteita ja lisämääreitä.

3.2.2 Attribuutit

Luokkien **attribuutit** (attributes) ovat tietoja, jotka kuvaavat olioiden ominaisuuksia. Ne sisältävät tiedot, jotka kuvaavat yksittäistä luokan ilmentymää ja erottavat sen muista saman luokan ilmentymistä.

Attribuutilla on *tietotyyppi*, joka kertoo millainen attribuutti on kyseessä. Yleisiä attribuutin tyyppejä ovat **kokonaisluku** (integer), **totuusarvo** (boolean), **desimaaliluku** (real), **piste** (point), **alue** (area) ja **numerointi** (enumeration), joita kutsutaan **alkeistietotyypeiksi** (primitive types). UML:ssä ei ole valmiiksi määriteltyjä alkeistietotyyppejä, vaan UML-kaavioiden piirtämiseen käytetty työkalu voidaan konfiguroida tiettyä ohjelmointikieltä varten, jolloin kyseisen kielen alkeistietotyypit tulevat käyttöön.

Attribuuteilla voi olla erilaisia **näkyvyyksiä** (visibility):

- **Julkisessa** (public) näkyvyydessä attribuutti on käytettävissä luokan ulkopuolella.
- **Yksityisessä** (private) näkyvyydessä attribuuttiin ei päästä käsiksi muista luokista.
- **Suojattua** (protected) näkyvyyttä käytetään yleistyksen ja erikoistumisen kanssa. Siinä attribuuttien käsittely on rajoitettu luokan ja sen aliluokkien sisään.

Julkinen näkyvyys ilmaistaan plusmerkillä (+), yksityinen miinusmerkillä (-) ja suojattu risuaidalla (#). Määrittelemättömissä tapauksissa merkkejä ei käytetä.

Luokkatason attribuutti eli **luokkamuuttuja** (class variable) jaetaan kaikkien luokan olioiden kesken, sen arvo on kaikilla olioilla sama. UML-kaavioissa luokkamuuttujat alleviivataan.

Attribuuteille voidaan määrätä myös *arvoalue*, joka merkitään aaltosulkujen sisään vaihtoehdot pilkulla erottaen.

Attribuutin formaali syntaksi:

näkyvyys nimi: tietotyyppi = alkuarvo {ominaisuus}

Vain nimi ja tyyppi ovat pakollisia. Attribuuteille voidaan antaa myös oletusarvoja (alkuarvoja). Arvoalueen lisäksi ominaisuudella voidaan määrittää myös muita attribuutin tietoja, kuten **säilyvyyttä** (persistence).

Esimerkkejä attribuuttien muodoista:

+ rekisterinumero: String
– palkka: Real
+ tilanne: Tilanne = maksamatta {maksamatta, maksettu}
<u>– ostojenMäärä: Integer</u> (Huom. kyseessä on luokkamuuttuja)

Jos osa luokan attribuuteista muodostaa kiinteän ryhmän, voidaan näistä muodostaa oma luokkansa. Esimerkiksi, jos henkilöllä on useita osoitetietoja (koti-, opiskelu-, työosoitteet) kannattaa muodostaa luokat **Henkilö** ja **Osoite**.

3.2.3 Operaatiot

Operaatioilla (operation) käsitellään attribuutteja tai tehdään muita toimenpiteitä. Ne ovat luokan sisäisiä ja niitä voidaan kutsua vain luokan olioille. **Operaation muodon** (operation signature) muodostavat operaation palautusarvo, nimi ja nolla tai useampi parametri.

Luokan operaatiot kuvaavat luokan toiminnallisuutta. Mitä palveluja luokka tarjoaa. Operaatiota voidaan siis ajatella luokan rajapintana. Myös operaatioilla voi olla näkyvyys (ks. 3.2.2) ja **taso** (scope).

Luokalla voi olla myös *luokkatason operaatioita*, joita voidaan kutsua ilman luokan olioita. Luokkatason operaatiot voivat käsitellä kuitenkin vain luokkatason muuttujia.

Operaation muodollinen syntaksi:

näkyvyys nimi (parametrilista): palautusarvotyyppi {ominaisuus}

Parametrilista on pilkulla erotettu lista muodollisia parametreja, jotka määritellään syntaksilla:

nimi: tietotyyppi = oletusarvo

Parametreilla voi olla oletusarvoja, joita käytetään, jos operaatiota kutsuttaessa kyseinen parametri jätetään antamatta.

Esimerkkejä operaatioiden muodoista:

+ piirrä()
+ aja(nopeus: Integer = 80)
<u>Laskuri():Integer</u>

Operaation toteutus on nimeltään **metodi** (method) ja se määritellään **muodon** (signature) avulla, tai **esiehtojen** (precondition), **jälkiehtojen** (postcondition), algoritmin ja vaikutuksen avulla. Esiehto on ehto, jonka on oltava tosi ennen operaation toteuttamista ja jälkiehdon on oltava tosi operaation suorittamisen jälkeen.

Luokka on *säilyvä*, jos sen oliot ovat olemassa sen jälkeenkin, kun ne luonut ohjelma on päättynyt. Luokka voidaan määritellä säilyväksi lisäämällä ominaisuus ”persistent” sen nimiosastoon. Luokkakaaviossa nimen jälkeen lukee tällöin {persistent}.

Olioita luotaessa niiden tulisi yleensä alustaa muuttujansa ja **yhteytensä** (links) muihin olioihin. Tämä voidaan toteuttaa esim. luokkatason operaatiolla nimeltä luo, jota käytetään luomiseen ja alustamiseen. Luokassa voi olla myös operaatio, jolla on sama nimi kuin luokalla ja jota kutsutaan luokan olion alustamiseksi. (Vrt. javan ja c++:n **muodostinmetodi** (constructor)).

Kuvassa (Kuva 3.2) on luokka, jolla on sekä julkisia että yksityisiä attribuutteja ja julkisia operaatioita. Ikä-attribuutti on alustettu nollaksi.

Kuva 3.2 Esimerkki luokasta.

Henkilö-luokka koodina:

```
class Henkilö {
// Attributes:
  public:
  // public data members:
    String nimi;  //julkinen attribuutti nimi
  private:
  // private data members:
    Integer ikä;  //yksityinen attribuutti ikä on alustettu nollaksi
// Operations:
  public:
  // public member functions:
    vanhene();  //julkinen operaatio vanhene()
    tervehdi();  //julkinen operaatio tervehdi)
   Henkilö() {
         ikä = 0;
         }
   ~Henkilö();{
         }

};
```

3.2.4 Luokkien löytäminen

Luokkia etsittäessä voidaan miettiä seuraaviin kysymyksiin vastauksia:

- Onko olemassa tietoa, joka pitäisi tallentaa tai analysoida?
- Onko olemassa ulkoisia järjestelmiä?
- Onko olemassa kaavoja, luokkakirjastoja, komponentteja tai muita vastaavia rakennelmia?
- Täytyykö järjestelmän käsitellä laitteita?

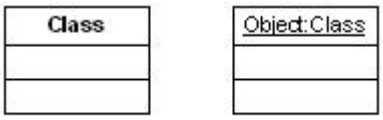
- Onko olemassa organisaation osia? (etenkin liiketoimintamalleissa)
- Mitä rooleja liiketoiminnan toimijoilla on? (käyttäjä, ylläpitäjä, asiakas)

3.3 Oliokaavio

Luokkakaavio sisältää vain luokkia, mutta sen muunnos, **oliokaavio** (object diagram), kuvaa luokkarakennetta ilmentymätasolla. Oliokaavio näyttää yksittäisten luokkien ilmentymien ja niiden välisten yhteyksien tilanteen jollakin hetkellä. Se on otos kuvattavan järjestelmän yhdestä mahdollisesta tilasta.

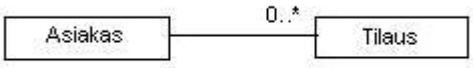
UML:n oliokaavioissa käytetään samaa merkintätapaa ja suhteita kuin luokkakaavioissa. Olio näytetään luokkana nimi alleviivattuna. Olion nimi voidaan näyttää ennen luokan nimeä: olionNimi:luokanNimi. Jos oliolla ei ole nimeä, näytetään vain kaksoispistealkuinen luokan nimi alleviivattuna.

Tekstin vahvennus ja alleviivaus kertovat, esittääkö symboli luokkaa vai oliota (Kuva 3.3).



Kuva 3.3 Luokka ja olio

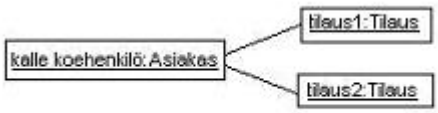
Kuvassa (Kuva 3.4) on luokkakaavio ja sitä vastaava oliokaavio (Kuva 3.5), jossa kalle koehenkilö–oliolla on kaksi tilausta: tilaus1 ja tilaus2 (olioita).



Kuva 3.4 Esimerkki luokkakaaviosta

Esimerkin luokat koodeina:

```
class Asiakas {
// Attributes:
  private:
    Tilaus* theTilauss;
};

class Tilaus {
};
```



Kuva 3.5 Esimerkki oliokaaviosta

3.4 Suhteet

Luokkakaaviot koostuvat luokista ja niiden välisistä suhteista, joita ovat: *assosiaatio, yleistykset, riippuvuudet* ja *tarkennukset*.

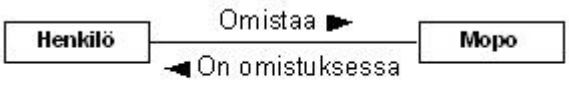
3.4.1 Assosiaatiot

Assosiaatio on yleensä kaksisuuntainen yhteys luokkien välillä.

1. **Tavallinen assosiaatio:**

Yleisin assosiaatio on pelkkä luokkien välinen yhteys, joka piirretään yhtenäisenä viivana kahden luokan väliin. Assosiaation nimi (yleensä verbi) kirjoitetaan viivan läheisyyteen. Nimiä voi olla kaksi. Tällöin nimen suunta esitetään pienellä täytetyllä kolmiolla ennen nimeä tai sen jälkeen, riippuen yhteyden suunnasta.

Kuvassa (Kuva 3.5) Henkilö-luokalla on assosiaatio Mopo-luokkaan. Henkilö omistaa mopon ja mopo on omistuksessa henkilöllä.



Kuva 3.6 Esimerkki assosiaatiosta

Esimerkki koodina:

```
class Henkilö {
// Attributes:
  private:
    Mopo* theMopo;
};

class Mopo {
// Attributes:
  private:
    Henkilö* theHenkilö;
};
```

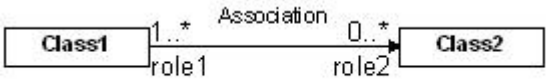
Jos assosiaatio on yksisuuntainen, lisätään sen toiseen päähän nuoli, jolloin assosiaatiota voidaan käyttää vain nuolen suuntaan.

Assosiaation **kerrannaisuus** (multiplicity) ilmaistaan lukuvälillä, joka kertoo, kuinka monta oliota yhteydellä voi olla. Lukuväli voi olla:

- Nollasta yhteen (0..1)
- Nollasta moneen (0..* tai pelkkä *)
- Yhdestä moneen (1..*)
- Kaksi (2)

- Neljästä seitsemään (4..7)
- Lukusarja (1, 5, 7, 9…12)
- Jne.

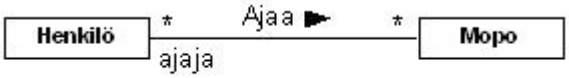
Kerrannaisuus (kirjallisuudessa käytetään myös termiä *kardinaalisuus*) on oletusarvoisesti yksi (1) ja se näytetään assosiaation päässä sen luokan lähellä, johon se pätee. (Kuva 3.7)



Kuva 3.7 Assosiaatio, kerrannaisuus ja roolit.

Assosiaatiolla voi olla **rooli** (role) liitettynä siihen kuuluviin luokkiin. Roolinimi on merkkijono ja se kertoo, missä ominaisuudessa oliot toimivat. Roolinimet kuuluvat assosiaatioon, eivät luokkiin, ja niiden käyttö on valinnaista.

Kuvassa (Kuva 3.7) henkilö toimii ajajan roolissa Mopon ja Henkilön välissä ajaa-assosiaatiossa.

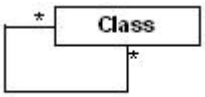


Kuva 3.8 Esimerkki rooleista assosiaatiossa.

Mallinnettaessa monimutkaisia järjestelmiä tulosten eli mallin selittäminen on tärkeää.

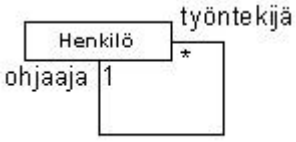
2. Rekursiivinen assosiaatio:

Luokka voidaan kytkeä itseensä assosiaatiolla. Luokan assosiaatio itseensä on **rekursiivinen assosiaatio** (recursive association).



Kuva 3.9 Rekursiivinen assosiaatio.

Kuvassa (Kuva 3.10) ohjaajan ja työntekijän välillä on assosiaatio. Sekä ohjaaja että työntekijä ovat henkilöitä.



Kuva 3.10 Esimerkki rekursiivisesta assosiaatiosta.

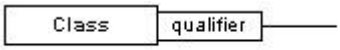
Esimerkki koodina:

```
class Henkilö {
// Attributes:
  private:
    Henkilö* ohjaaja;
};
```

3. Valitsinassosiaatio:

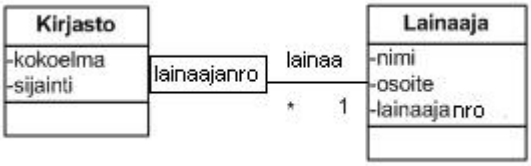
Jos assosiaatioon liittyy tieto, joka määrää assosiaation ilmentymien toisessa päässä olevien olioiden joukon, voidaan tämä tieto esittää nk. **yksilöintinä** (qualification). **Valitsinassosiaatiota** (qualifier association) käytetään siis moninkertaisissa assosiaatioissa. Tällöin assosiaatiossa oleva **valitsin** (qualifier) erottelee moni-päässä olevat oliot toisistaan yksilöiden ne. Valitsin määrittää, miten yksittäinen olio assosiaation moni-päässä valitaan.

Valitsin piirretään luokkasymboliin liittyvänä pienenä laatikkona assosiaation siihen päähän, josta navigoinnin tulee tapahtua. Laatikon sisään kirjoitetaan linkin yksilöivä tieto. (Kuva 3.11)



Kuva 3.11 Valitsinassosiaatio.

Kuvassa (Kuva 3.12) lainaajanumero (lainaajanro) identifioi yksiselitteisesti lainaajan tietyn kirjaston kannalta.



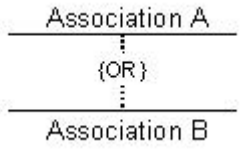
Kuva 3.12 Esimerkki valitsinassosiaatiosta.

```
class Kirjasto {
// Attributes:
  private:
    kokoelma;
    sijainti;
    Lainaaja* theLainaaja;
};

 class Lainaaja {
 //Attributes:
    private:
      nimi;
      osoite;
      lainaajanro;
     };
```

Kun lainaaja lisätään, tarkistetaan attribuutilla lainaajanro. Näin varmistetaan, että vastaavia numeroita ei ole olemassa.

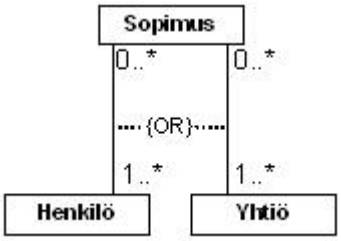
4. **Tai-assosiaatio:**

Tai-assosiaatio on kahden tai useamman assosiaation **rajoitin** (constraint), joka määrittelee, että luokan oliot voivat osallistua korkeintaan yhteen assosiaatioon kerrallaan. Tai-assosiaatio piirretään katkoviivana siihen liittyvien assosiaatioiden välille ja siihen liitetään määritelmä {or} (Kuva 3.13).



Kuva 3.13 Tai-rajoitus assosiaatioiden välillä

Kuvassa (Kuva 3.14) on esimerkki tai-assosiaatiosta, joka näyttää, että vain toinen assosiaatio on käytössä samanaikaisesti. Sopimuksen voi tehdä vain joko henkilö tai yhtiö.



Kuva 3.14 Esimerkki tai-assosiaatiosta.

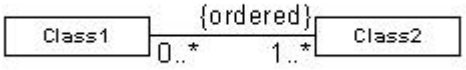
Esimerkki koodina:

```
class Sopimus {
// Attributes:
  private:
    Henkilö* theHenkilös;
    Yhtio* theYhtiös;
};
```

Ajettaessa tarkistetaan, että **vain** toinen tekee sopimuksen.

5. **Järjestetty assosiaatio** (ordered association):

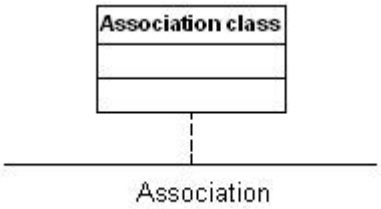
Yleensä kertautuvan suhteen "moninkertaisella" puolella on vain järjestämätön joukko olioita. Jos ne kuitenkin ovat järjestyksessä, merkitään assosiaation kertautuvaan päähän {ordered} (Kuva 3.15). Oletuksena on, että assosiaatio on järjestämätön.



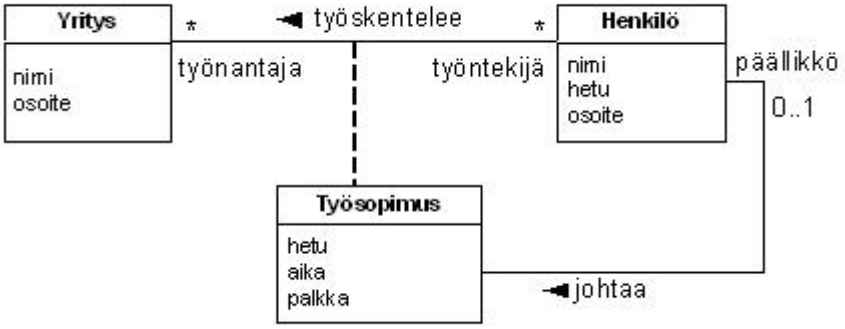
Kuva 3.15 Järjestetty assosiaatio.

6. Assosiatiivinen luokka

Assosiatiivinen luokka (association class) on assosiaatioon kytketty luokka. Se on kuten mikä tahansa luokka attribuutteineen, operaatioineen ja muine assosiaatioineen. Assosiatiivinen luokka antaa lisätietoa/toiminnallisuutta yhteydelle ja jokainen assosiaation yhteys on yhteydessä myös assosiatiivisen luokan olioon. Assosiatiivinen luokka liitetään itse assosiaatioon katkoviivalla (Kuvat 3.16 ja 3.17).



Kuva 3.16 Assosiatiivinen luokka.



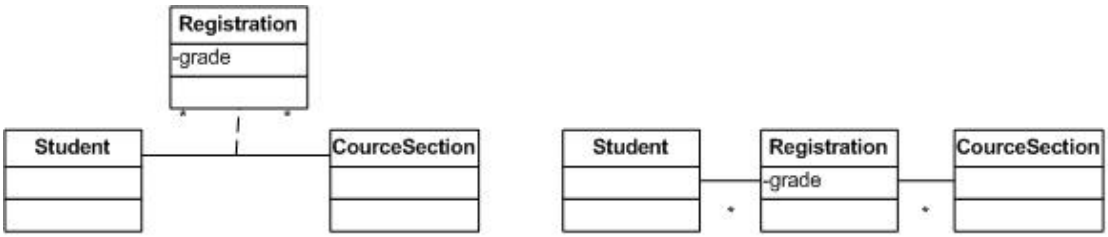
Kuva 3.17 Esimerkki assosiaatioluokasta.

Assosiaatioluokka koodissa:

```
class Työsopimus {
// Attributes:
  private:
    aika;
    palkka;
    hetu;
};

class Henkilö {
// Attributes:
  private:
    nimi;
    osoite;
    hetu;
    Yritys* työnantaja;
    Työsopimus* theTyösopimus;
    Työsopimus* johtaa;
};

class Yritys {
// Attributes:
  private:
```

```
   nimi;
   osoite;
};
```

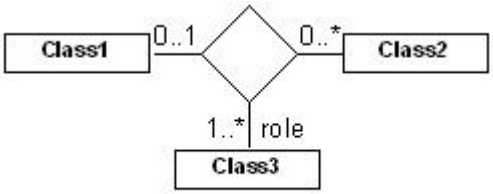
Joskus kahteen luokkaan liittyvä attribuutti ei sijoitu hyvin kumpaankaan luokista. Kumpikin alla olevista malleista kuvaa samaa asiaa.



Kuva 3.17b Assosiaatioluokan kuvausvaihtoehtoja

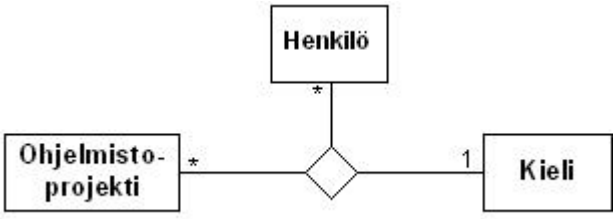
7. Kolmioassosiaatio

Joskus harvoin on tarpeellista muodostaa kolmen luokan välinen assosiaatio. **Kolmioassosiaatio** (ternary association) liittää kolme luokkaa keskenään. Se piirretään suurena vinoneliönä (Kuva 3.18). Roolit ja kerrannaisuudet voidaan näyttää, mutta seuraavassa käsiteltäviä koostumussuhteita ja valitsimia ei hyväksytä. Assosiaatioluokka voidaan liittää kolmioassosiaatioon piirtämällä katkoviiva johonkin vinoneliön nurkista.



Kuva 3.18 Kolmioassosiaatio.

Kuvan (Kuva 3.19) mukaan, jos on annettu projekti ja kieli, voivat nolla tai useampi henkilöä käyttää tätä kieltä. Annetussa projektissa annettu henkilö käyttää yhtä kieltä. Henkilö voi käyttää annettua kieltä yhdessä tai useammassa projektissa. Kolmioassosiaatio ei tee muutoksia koodiin.

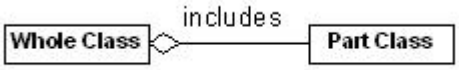


Kuva 3.19 Esimerkki kolmioassosiaatiosta.

8. Koostumussuhde

Koostumussuhde (aggregation) on assosiaation erityinen muoto. Kooste esittää suhteen ”on osa” tai ”kuuluu” luokkien välillä.

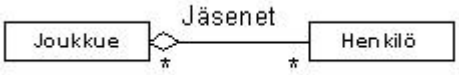
Koostumussuhde piirretään viivana, jonka sisältävässä päässä on pieni ontto vinoneliö (Kuva 3.20). Siihen voidaan soveltaa normaaliin tapaan kertautumista, rooleja ja valitsimia. Koostumuksella voi olla myös suunta ja nimi.



Kuva 3.20 Koostumussuhde.

Jaetussa koostumuksessa (shared aggregation) osat voivat olla useamman kokonaisen osia ja koostumuksen jako näkyy sen kerrannaisuudessa. Jaetussa koostumuksessa kokonaispuolen kerrannaisuus on jotain muuta kuin yksi (1).

Kuvassa (Kuva 3.21) joukkue koostuu joukkueen jäsenistä. Yksi henkilö voi olla useamman joukkueen jäsen.



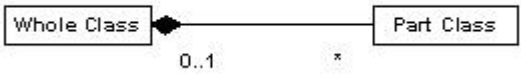
Kuva 3.21 Esimerkki jaetusta koostumuksesta.

Jaettu koostumus koodina:

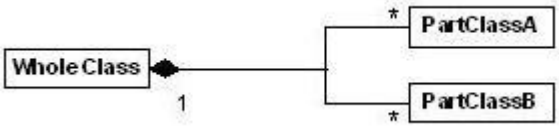
```
class Joukkue {
// Attributes:
  private:
    Henkilö* theHenkilös;
};

class Henkilö {
};
```

Vahva koostumus (composition aggregation) omistaa osansa ja siinä on hyvin voimakas omistussuhde. Osat kuuluvat kokonaisuuteen ja ne tuhotaan kokonaisuuden kanssa. Kokonaispuolen kerrannaisuuden on oltava nolla tai yksi, mutta osapuolelle kerrannaisuus voi olla mikä tahansa väli.

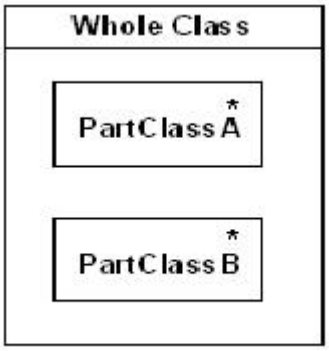
Vahva koostumus voidaan piirtää kolmella eri tavalla: Se voidaan esittää viivana, jonka kokonaispuolella on täytetty vinoneliö (Kuva 3.22). Toiseksi, jos samaan kokonaisuuteen kuuluu useita osia, ne voidaan piirtää puuna, joka yhdistyy kokonaispuolella yhdeksi pääksi (Kuva 3.23). Kolmas tapa on sijoittaa osaluokat kokonaisluokan sisään (Kuva 3.24).



Kuva 3.22 Vahva koostumus 1

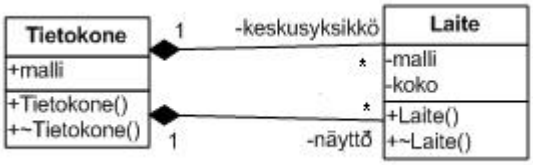


Kuva 3.23 Vahva koostumus 2



Kuva 3.24 Vahva koostumus 3

Kuvassa (Kuva 3.25) Tietokone omistaa Keskusyksikön ja Näytön.



Kuva 3.25 Esimerkki vahvasta koostumuksesta

Vahva koostumus koodina:

```
class Tietokone {
// Attributes:
  private:
    malli:
    Laite keskusyksikkö;
    Laite näyttö;
// Operations:
  public:
    Tietokone(){
       * CPU = new Laite;
       * Monitor = new Laite;
       }
    ~Tietokone() {
       delete CPU;
       delete Monitor;
```

```
      }
};

class Laite {
// Attributes:
  private:
    malli;
    koko;
// Operations:
  public:
    Laite();
    ~Laite();
};
```

3.4.2 Yleistys

Yleistyksessä erikoisempi elementti on täysin yhteensopiva yleisen elementin kanssa ja sisältää lisätietoa. Erikoiselementin ilmentymää voidaan käyttää kaikkialla, missä yleisen elementin ilmentymä kelpaa. Yleistäminen tai **periminen** (inheritance) mahdollistaa elementtien erikoistamisen uusiksi elementeiksi.

1. **Tavallinen yleistys**

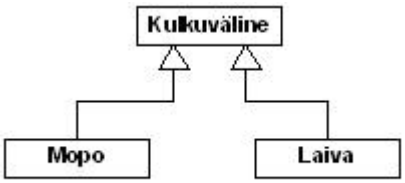
Yleistys (generalization) on erikoisen ja yleisen luokan välillä oleva suhde, missä erikoisempi luokka eli **aliluokka** (subclass) perii yleiseltä luokalta eli **yliluokalta** (subclass) kaikki sen ominaisuudet (attribuutit, operaatiot ja assosiaatiot). Luokka voi olla sekä yli- että aliluokka.

Julkiseksi määritellyt yliluokan **jäsenet** (members), attribuutit ja operaatiot, ovat julkisia myös aliluokassa. Yksityiset jäsenet peritään myös, mutta niitä ei voi käsitellä aliluokassa. Jäsenten käsittely voidaan rajoittaa luokan ja sen aliluokkien sisään asettamalla niille **suojattu** (protected) **näkyvyys**, jota merkitään risuaidalla (#).

Yleistys kuvataan yhtenäisellä viivalla luokkien välillä, jonka yliluokan puoleisessa päässä on ontto kolmio (Kuva 3.26).

Kuva 3.26 Yleistys.

Kuvassa (Kuva 3.27) kulkuväline on yleinen luokka eli yliluokka, josta on johdettu aliluokkia (mopo ja laiva) perimisellä, jota kutsutaan myös yleistykseksi tai erikoistumiseksi.



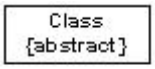
Kuva 3.27 Esimerkki yleistyksestä.

Koodi:

class Kulkuväline{};

class Mopo : Kulkuväline{}; ***class Laiva : Kulkuväline{};***

Abstrakti luokka (abstact class) on luokka, jolla ei voi olla yhtään oliota. Se kuvailee siitä perittävien luokkien yhteisiä attribuutteja ja toimintoja. Abstrakti luokka on ”puolivalmis” luokka, joka antaa vapaat kädet puuttuvien metodien toteutukselle. Luokka voidaan määritellä abstraktiksi lisäämällä sen nimiosastoon nimen alle ominaisuus {abstract} (Kuva 3.28).



Kuva 3.28 Abstrakti luokka

Abstraktilla luokalla on yleensä myös **abstrakteja operaatioita** (abstract operation), joista tunnetaan vain sen muoto, ei toteuttavaa metodia. Vain abstraktilla luokalla voi olla abstrakteja operaatioita. Jos luokka perii abstraktin luokan, on sen toteutettava kaikki abstraktit operaatiot, tai oltava itse abstrakti luokka. Abstraktit operaatiot määritellään muodon perässä olevalla ominaisuudelle {abstract} tai kirjoittamalla operaation muoto kursiivilla.

Aliluokat voivat määritellä operaatiota uudelleen. Uudelleenmääritellyn operaation muodon eli palautusarvon, nimen ja parametrien, on oltava sama kuin yliluokan operaation muoto. Aliluokan uudelleenmääriteltyä metodia käytetään luokan kaikissa ilmentymissä. Aliluokkiin voidaan lisätä myös uusia operaatioita, attribuutteja ja assosiaatioita.

Polymorfismissa (polymorphism) aliluokan olio toimii yliluokan oliona ja yksi tai useampi yliluokan operaatioista on uudelleenmääritelty. Todellinen käytetty operaation toteutus eli metodi riippuu sen olion tyypistä, jonka operaatioita kutsutaan.

Yleistys-erikoistumis -suhteita voidaan tarkentaa **erottimilla** (discriminator), jotka kertovat, millä perusteella yleistys ja erikoistuminen tapahtuvat.

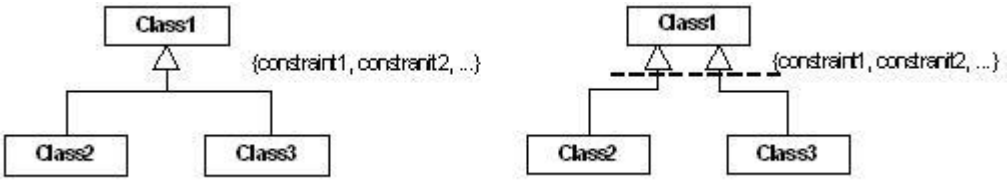
2. Yleistysten rajoitukset

Yleistykselle voidaan määritellä **rajoituksia** (constraint), jotka antavat lisätietoa yleistyksen jatkokäytöstä. Seuraavanlaisia yleistyksiä, joihin liittyy useampi kuin yksi aliluokka, on määritelty:

- **Päällekkäisessä** (overlapping) yleistyksessä kaikki seuraavat aliluokat, jotka perivät tämän perimisen aliluokkia, voivat periä useamman kuin yhden aliluokista (yhteisen kantaluokan moniperintä).
- **Erillinen** (disjoint) yleistys on päällekkäisen yleistyksen vastakohta. Aliluokkia ei saa periä yhteiseen aliluokkaan. Erillinen perintä on oletusarvoinen.

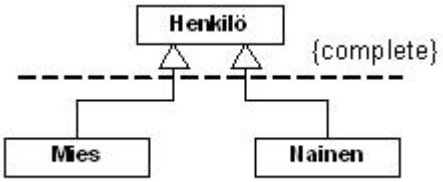
- **Täydellisessä** (complete) yleistyksessä kaikki aliluokat on määritelty, eikä uusia aliluokkia saa tehdä.
- **Epätäydellinen** (incomplete) yleistys on oletusarvoinen ja edellisen vastakohta. Aliluokkia saa lisätä myöhemmin.

Rajoitteet kirjoitetaan aaltosulkujen {} sisään lähellä yleistyssuhteen onttoa kolmiota, jos kolmio on jaettu aliluokkien kesken. Tapauksessa, jossa jokaisella aliluokalla on oma kolmio, piirretään katkoviiva kaikkien yleistyssuhdeviivojen yli ja rajoite merkitään katkoviivan lähelle (Kuva 3.29).



Kuva 3.29 Eri tapoja näyttää perimisen rajoitukset

Kuvassa (Kuva 3.30) yleistys on rajoitettu täydelliseksi, joten uusia aliluokkia ei voida lisätä.



Kuva 3.30 Esimerkki yleistyksen rajoituksesta.

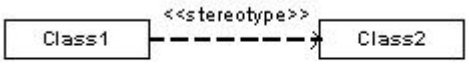
Perimisen rajoitukset eivät näy luokkien koodeissa.

3.4.3 Riippuvuus- ja tarkennussuhteet

Riippuvuussuhde (dependency) on riippumattoman ja riippuvaisen mallinnuselementin (luokka, paketti, käyttötapaus tai jokin muu elementti) välinen yhteys. Muutos riippumattomassa itsenäisessä elementissä vaikuttaa riippuvaiseen elementtiin.

Riippuvuussuhde muodostuu esim. toisen luokan ottaessa toisen luokan olion parametrinaan tai kun luokka käsittelee toisen luokan globaalia oliota tai kutsuu toisen luokan luokkatason operaatiota.

Riippuvuussuhde kuvataan kahden elementin välisellä katkoviivalla, jossa on nuoli toisessa päässä (Kuva 3.31). Suhteeseen voidaan liittää riippuvuuden lajin ilmoittama stereotyyppi.



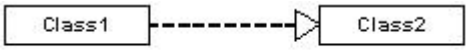
Kuva 3.31 Riippuvuussuhde.

Riippuvuus ilmenee esimerkiksi, jos toinen luokka saa toisen funktion parametriksi. Koodissa se näkyisi näin:

```
class Class1 {
//Operations:
  public:
    toimi(Class2);
};

class Class2 {
};
```

Tarkennussuhde (refinement) on abstraktiotasolla toimivien kahden saman asian kuvausten välinen suhde. Se voi olla esim. tyypin ja sen toteuttavan luokan välillä, jolloin sitä kutsutaan **toteutukseksi** (realization). Tarkennussuhde voi olla myös suhde analyysiluokan ja samaa asiaa mallintavan suunnitteluluokan välillä tai yleisen ja tarkan kuvauksen välillä.

Tarkennussuhde kuvataan kahden elementin välisenä katkoviivana, jossa on ontto kolmio toisessa päässä (Kuva 3.32).



Kuva 3.32 Tarkennussuhde.

Tarkennussuhde Java-koodina:

```
public class Class_1 implements Class_2{
}
```

3.4.4 Säännöt eli rajoitukset ja johteet

UML:ssä voidaan ilmaista **sääntöjä**, jotka ovat joko mallia rajoittavia **rajoituksia** (constraint) tai **johteita** (derivations). Johteet ovat sääntöjä asioiden johtamiselle. (Esim. tuotteen tuotto on sen myyntihinta miinus kustannukset.) Kaikki säännöt näytetään aaltosulkujen sisällä elementin lähellä tai huomautuselementissä, joka on yhdistetty mallinnuselementtiin.

Sääntöjä voidaan käyttää mihin tahansa mallinnuselementteihin (esim. attribuutit, assosiaatiot, periminen, roolit ja myöhemmin käsiteltävien dynaamisten mallien aikarajoituksille). Yleistyksellä on vain rajoituksia, ei johteita (ks. 3.4.2). Rooleilla voi olla rajoituksia, jotka rajoittavat yhden olion esittämien roolien yhdistelmiä

Kun sääntöjä ilmaistaan rajoituksina ja johteina, ne viittaavat mallissa olevaan mallinnuselementtiin. Tämä tehdään **navigointilausekkeella** (navigation expression),

johon kuuluu peruskieli sääntöjen määrittämiseen. Joukko tarkoittaa kaikissa tapauksissa joko oliota tai oliojoukon lauseketta.

joukko '.' attribuutti:
Lausekkeen arvo on assosiaation kerrannaisuudesta riippuen attribuutin arvo tai arvojoukko. **Esim:** *Lasku.maara > 100 €;*

joukko '.' rooli:
Rooli tarkoittaa assosiaation kohdepuolta. Lausekkeen tulos on olio tai oliojoukko assosiaatiosta riippuen. **Esim:** *Auto.Ajaja.ajokortti = true;*
joukko '.' '~' rooli:
Tässä rooli tarkoittaa assosiaation lähdepuolta. Lausekkeen tulos kuten edellä.
Esim: *Henkilö.~Asiakas.ostos < 100 €;*

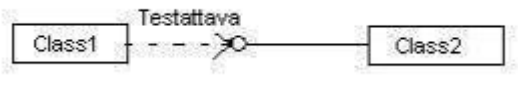
joukko '[' totuusarvolauseke ']':
Totuusarvolauseke kirjoitetaan joukon kannalta. Lausekkeen tulos on oliojoukon alijoukko, eli totuusarvolausekkeen toteuttavat oliot.
Esim: *Henkilo [Toimittaja.Mahdollisuus] < Henkilo[Toimittaja.Riski];*

joukko '.' '[' valitsinarvo ']':
Valitsinarvo määrittää valitsinassosiaation joukolle. Lausekkeen tulos on valitsinassosiaation takana oleva olio, joka sopii valitsimen ehtoon.
Esim: *Pankki.Asiakas.[tilinumero];*

3.5 Rajapinnat

Rajapinnat (interfaces) ovat tärkeitä hyvin järjestellyn järjestelmän rakentamisessa ja ne toimivat sopimuksina yhteistyössä toimivien mallinnuselementtijoukkojen välillä. Ne kuvaavat olioiden ulospäin näkyvän toiminnallisuuden.

UML:ssä rajapinta näytetään pienenä ympyränä, jolla on nimi ja se kytketään mallinnuselementtiinsä viivalla. Luokka, joka käyttää rajapintaa, jonka jokin luokka on toteuttanut, yhdistetään riippuvuussuhteella (katkoviivalla) rajapintaympyrään. (Kuva 3.33a) Riippuvainen luokka on riippuvainen ainoastaan rajapinnan operaatioista ja voi käyttää niitä. Mikäli luokka on riippuvainen muista luokan ominaisuuksista, pitää riippuvuusnuolen kulkea luokkaan asti, ei vain rajapintaan. Yksi luokka voi toteuttaa useita rajapintoja.



Kuva 3.33a Rajapinta

Rajapinta määritellään luokaksi, jolla on stereotyyppi <<interface>>, joten sen operaatiot voidaan näyttää tavallisella luokkasuorakaiteella. Rajapinta voi erikoistua kuten luokkakin. Perimissuhde rajapintojen välillä näytetään luokkakaaviossa, jossa rajapinnat näytetään luokkalaatikkoina stereotyyppinä <<interface>>.

Kuvassa (Kuva 3.33b) luokka Class2 toteuttaa rajapinnan Testattava ja luokka Class1 käyttää rajapintaa Testattava luokassa Class2. Testattava määritellään luokkana, jolla on stereotyyppi <<interface>>.

Kuva 3.33b Rajapinta

Rajapinta Java-koodina:

```
public interface Testattava {
// Attributes:
   Class_2 theClass_2;
// Operations:
   public testaa();
};
```

3.6 Paketit

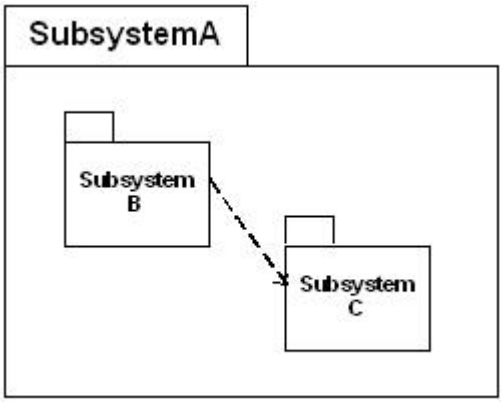
Paketti (package) on UML:n kaikissa kaaviotyypeissä käytettävä ryhmittely- ja kokoamisväline. Pakkausta käytetään kokoamaan yhteen toisiinsa liittyviä mallien osia, jotka tyypillisesti muuttuvat samanaikaisesti. UML-standardin mukaan paketteihin voi ryhmitellä mitä tahansa UML -mallinnuselementtejä, mutta tyypillisesti paketteihin ryhmitellään luokkia ja olioita. Paketti omistaa mallinnuselementtinsä, joten elementti ei voi kuulua kuin yhteen pakettiin. Paketteja kuvataan usein **alijärjestelminä** (subsystem).

Paketit voivat **tuoda** (import) elementtejä muista paketeista ja ne kuvataan riippuvuutena, jonka stereotyyppi on <<imports>>. Elementin tuonnissa viitataan vain pakettiin, joka omistaa elementin. Myös paketteja voidaan yhdistää suhteilla (riippuvuus, tarkennus ja yleistys).

Paketti piirretään suurena suorakaiteena, jonka vasempaan ylänurkkaan on kiinnitetty pienempi laatikko, ns. välilehti (Kuva 3.34). Mikäli paketti on suljettu, sen sisältöä ei näytetä. Kirjoitetaan vain paketin nimi laatikon sisään; muussa tapauksessa nimi kirjoitetaan välilehteen.

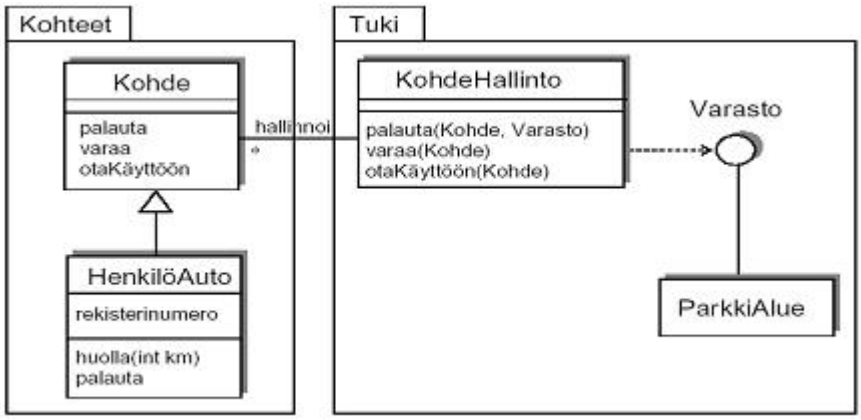
Kuva 3.34 Paketti.

Kuvassa (Kuva 3.35) **alijärjestelmä** (subsystem) B on riippuvainen alijärjestelmästä C. Alijärjestelmät B ja C ovat alijärjestelmän A sisällä. Kaikki alijärjestelmät esitetään paketteina.



Kuva 3.35 Esimerkki paketeista.

Paketti toimii myös suojausmekanismina, sillä se voi määritellä sisältämiensä elementtien *näkyvyyden* (ks. 3.2.2). Mahdollisia näkyvyyksiä on neljä: *yksityinen*, *julkinen*, *suojattu* ja *toteutus* (implementation). *Toteutusnäkyvyys* on samankaltainen kuin yksityinen näkyvyys, mutta paketista riippuvat mallinnuselementit eivät voi käyttää toteutuspaketin sisältöä. Jos paketti on toteutuspaketti, yksikään muu paketti ei voi tuoda siitä elementtejä.



Kuva 3.36 Esimerkki paketista.

Pakettiesimerkin luokat koodeina:

```
class Henkilöauto : Kohde {
//Attributes:
  private:
    rekisterinumero;
//Operations:
  public:
    huolla(int km);
    palauta()
};
```

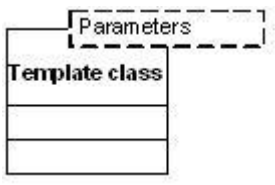
```
class Kohde {
//Attributes:
  private:
  KohdeHallinto* theKohdeHallinto;
//Operations:
 public:
    palauta();
    varaa();
    otaKäyttöön();
};

class KohdeHallinto {
//Operations:
 public:
   palauta(Kohde, Varasto);
  varaa(Kohde);
  otaKäyttöön(Kohde);
};

class Varasto {
//Attributes:
  private:
    ParkkiAlue*  theParkkiAlue;
};

class ParkkiAlue {
};
```

Paketilla voi olla myös rajapinta, joka julkaisee sen toiminnallisuuden ja se näytetään kuten luokkien yhteydessä pienenä ympyränä (ks. 3.5). Yleensä yksi tai useampi luokka paketin sisällä toteuttaa rajapinnan.

3.7 Mallinteet

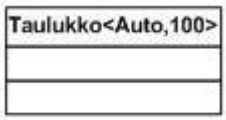
Mallinne (template) on luokka, jota ei ole täysin määritelty ja viimeinen määritelmä tehdään mallinteen parametrin (esim. luokka tai alkeistyyppi) avulla. Mallinne on ohje kääntäjälle luoda geneerisestä tyyppiriippumattomasta ohjelmakoodista tyyppiin sidottua ohjelmakoodia. Sen avulla voidaan samasta lähdekoodista luoda uusia instansseja eri tyyppien käsittelyä varten kirjoittamatta lähdekoodia uudelleen.

Parametroitua luokkaa käytetään määrittämään ryhmä luokkia. Parametroitu luokka voi olla esimerkiksi taulukko, jolloin mallinteen ilmentymät ovat luokkia, jotka voivat toimia taulukoina autoille, väreille ja niin edespäin. Sillä on parametrilista, jossa on pilkuilla eroteltuna jokaisen parametrin tyyppi ja nimi. Parametrilista näytetään katkoviivalaatikkona parametriluokan laatikon oikeassa ylänurkassa.



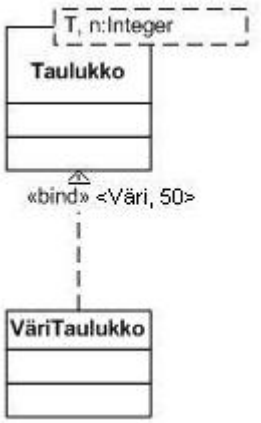
Kuva 3.37 Mallinne

Mallinteen ilmentymä eli luokkamäärittely näytetään tavallisena luokkalaatikkona, mutta luokan nimessä näkyy, mistä mallinteesta se on luotu ja millä parametreilla kuten esimerkiksi Taulukko<Auto,100> (Kuva 3.38a).



Kuva 3.38a Luokkalaatikko, jossa mallinne ja parametri

Kuvassa (Kuva 3.38b) on luokka Taulukko, jolla on parametrit T (luokka, jonka tyypin näyttäminen on vapaaehtoista) ja n (kokonaisluku). Kuvan kahdesta ilmentymästä toisessa on annettu mallin nimi ja parametri luokkasymbolin sisällä ja toisessa on määritelty sidontastereotyyppinen tarkennussuhde <<bind>> ilmentymäluokasta malliin. Käytetyt parametrit merkitään tyypin ”bind” jälkeen.

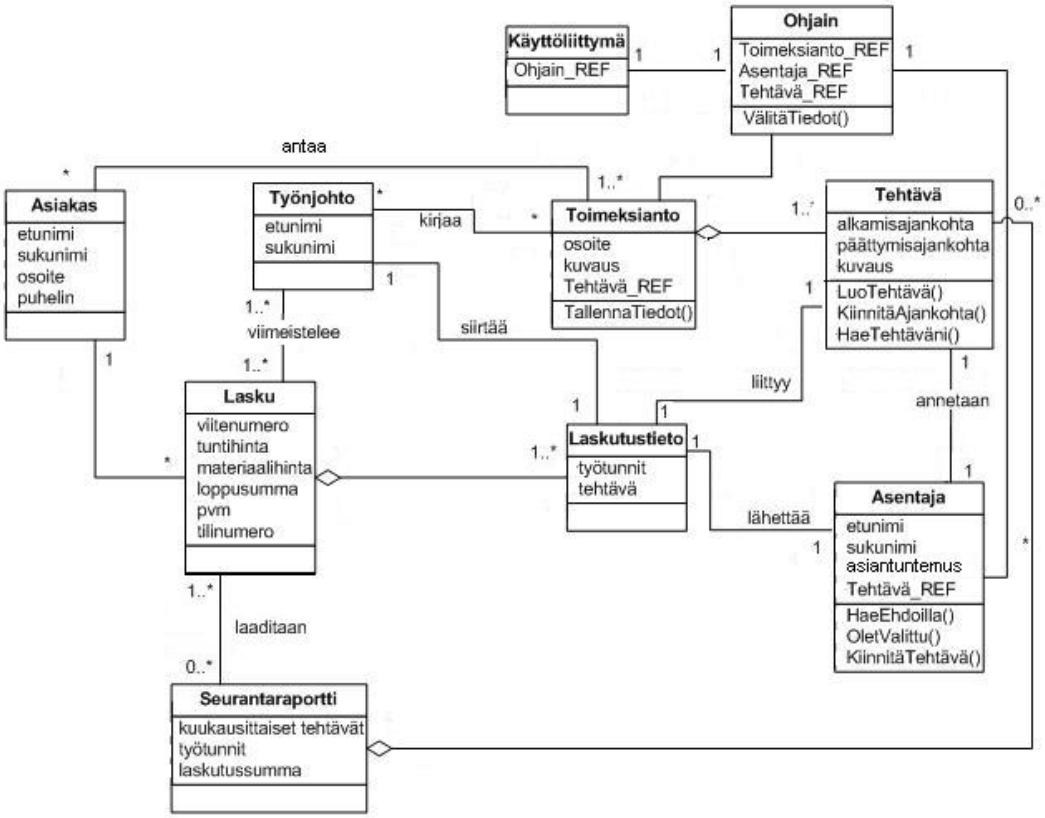


Kuva 3.38b Mallinteet parametroituna luokkana.

Kaikki ohjelmointikielet eivät tue mallinteita, esimerkiksi Javassa ei ole vastinetta mallinteille mutta C++ tukee niitä.

3.8 Esimerkkijärjestelmä

Luokkakaavion esitys on piirroksena (Kuva 3.39). Sanallinen selvitys luokista ja assosiaatioista esitetään liitteissä. Tietohakemisto on Liite 2:ssa ja luokkamääritykset ovat Liite 3:ssa.



Kuva 3.39 Suunnitteluvaiheen luokkakaavio

Suunnitteluvaiheen luokat koodeina:

```
class Käyttöliittymä {
// Attributes:
  private:
    Ohjain_REF;
    Ohjain* theOhjain;
};

class Ohjain {
// Attributes:
  private:
    Toimeksianto_REF;
    Asentaja_REF;
    Tehtävä_REF;
    Toimeksianto* theToimeksianto;
    Asentaja* theAsentajas;
    Tehtävä* theTehtävä;
// Operations:
  public:
    VälitäTiedot();
};

class Asiakas {
// Attributes:
  private:
    etunimi;
    sukunimi;
    osoite;
```

```
    puhelin;
    Toimeksianto* theToimeksiantos;
};

class Työnjohto {
// Attributes:
  private:
    etunimi;
    sukunimi;
    Lasku* theLaskus;
    Toimeksianto* theToimeksianto;
    Laskutustieto* theLaskutustieto;
};

class Toimeksianto {
// Attributes:
  private:
    osoite;
    kuvaus;
    Tehtävä_REF;
    Tehtävä* theTehtäväs;
// Operations:
  public:
    TallennaTiedot();
};

class Tehtävä {
// Attributes:
  private:
    alkamisajankohta;
    päättymisajankohta;
    kuvaus;
    Asentaja* theAsentaja;
// Operations:
  public:
    LuoTehtävä();
    KiinnitäAjankohta();
    HaeTehtäväni();
};

class Asentaja {
// Attributes:
  private:
    etunimi;
    sukunimi;
    asiantuntemus;
    Tehtävä_REF;
    Laskutustieto* theLaskutustieto;
// Operations:
  public:
    HaeEhdoilla();
    OletValittu();
    KiinnitäTehtävä();
};

class Laskutustieto {
// Attributes:
  private:
    työtunnit;
    tehtävä;
```

```
   Tehtävä* theTehtävä;
};

class Lasku {
// Attributes:
  private:
    viitenumero;
    tuntihinta;
    materiaalihinta;
    loppusumma;
    pvm;
    tilinumero;
    Laskutustieto* theLaskutustietos;
    Asiakas* theAsiakas;
    Seurantaraportti* theSeurantaraporttis;
};

class Seurantaraportti {
// Attributes:
  private:
    kuukausittaisettehtävät;
    työtunnit;
    laskutussumma;
    Tehtävä* theTehtäväs;
};
```

4. DYNAAMINEN MALLINNUS

Järjestelmän käyttäytyminen eli sen dynaamisuus näyttää, kuinka oliot ovat vuorovaikutuksessa järjestelmän suorituksen aikana. *Järjestelmän dynamiikkaan* kuuluu olioiden välinen viestintä (viestintäyhteistyö) ja sen seuraukset sekä olioiden tilojen muutokset järjestelmän ollessa käynnissä.

Järjestelmän dynaamisen käyttäytymisen mallintamisen kannalta keskeiset UML:n kaaviotyypit ovat tila-, viestiyhteys-, yhteistyö- ja toimintakaaviot. Tässä oppaassa käymme edellisistä läpi vain viestiyhteys- eli sekvenssikaavioita.

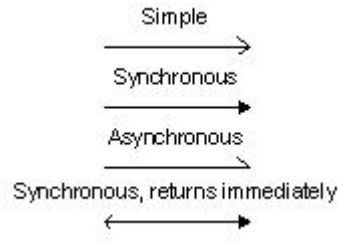
4.1 Olioiden välinen vuorovaikutus

Kahden olion välinen vuorovaikutus on olioiden välistää viestintää. Viesti on useimmiten operaatiokutsu, jossa olio kutsuu toisen olion operaatiota. Tosiaikaisissa järjestelmissä viesti voi olla oikea viesti, joka lähetetään jotakin viestintämekanismia käyttäen tietoverkon yli tai koneen sisällä. Viestejä käytetään kaikissa dynaamisissa kaavioissa olioiden välisenä viestintätapana ja ne piirretään viestin lähettäjän ja vastaanottajan välisenä viivana, jonka toisessa päässä on nuolenkärki.

UML:ssä käytetyt viestityypit ovat (Kuva 4.1):

- **Yksinkertainen** (simple) viesti, joka vastaa yksinkertaista **suorituksen etenemistä** (flow of control). Viestinnän yksityiskohtia ei kuvata. Tätä tyyppiä käytetään myös synkronisen viestin paluun osoittamiseen, mahdollisen palautusarvon kanssa.
- **Synkroninen** (synchronous) viesti on sisäkkäinen suorituksen siirto, joka yleisimmin toteutetaan operaatiokutsuna. Viestin käsittelevä operaatio suoritetaan loppuun ennen kuin kutsuja saa jatkaa. Palautus voidaan kuvata yksinkertaisena viestinä tai jättää merkitsemättä.
- **Asynkroninen** (asynchronous) viesti on epäsynkroninen suorituksen siirto, jossa ei ole erillistä palautusta ja jossa kutsuja jatkaa suoritustaan viestin lähettämisen jälkeen odottamatta sen käsittelyä.

Yksinkertainen ja synkroninen viesti voidaan yhdistää yhdeksi viivaksi, jolloin palautus tapahtuu lähes välittömästi operaatiokutsun jälkeen (Kuva 4.1, alimmainen).



Kuva 4.1 Viestityypit.

Laajennuksia UML:n viestityyppeihin ovat **peruttava viesti** (balking) sekä **luovutusviesti** (time-out). Peruttava viesti lähtee vain, jos vastaanottaja hyväksyy sen ja luovutusviesti perutaan, jos sitä ei käsitellä tietyn ajan kuluessa. Nämä, kuten muutkin vastaavat muunnokset, kuvataan *viestien stereotyyppeinä*.

4.2 Sekvenssikaavio

Sekvenssikaaviot (sequence diagrams) ts. *viestiyhteyskaaviot* kuvaavat olioita ja niiden välisiä suhteita (oliojoukon sisäinen viestintä) tietyssä tilanteessa. Sekvenssikaavio kertoo, mitä avustavia olioita tilanteen suorittamiseen tarvitaan sekä mitä olioiden palveluita ja missä järjestyksessä käytetään.

4.2.1 Sekvenssikaavion piirtäminen

Sekvenssikaaviossa on kaksi akselia, joista *aika* kulkee ylhäältä alas. *Vuorovaikutustapahtumat* kuvataan tässä aikaskaalassa vaakatasossa olevina nuolina sanoman lähettäjältä sen vastaanottajalle. Nuoli kertoo viestin laadun (synkroninen, asynkroninen tai yksinkertainen, ks. 4.1).

Sekvenssikaavion osallistujat kuvataan pystyviivoina, joiden yläpäässä on osallistujan yksilöivä symboli. Esimerkiksi oliot kuvataan tavanomaisella oliosymbolilla (suorakaide, jossa tunnisteteksti alleviivattuna) ja alijärjestelmät pakkaussymbolilla. Järjestelmän ulkoinen **toimija** (actor) voi olla myös osallistujana sekvenssikaaviossa. Käyttäjällä on oma kuvakkeensa ns. tikku-ukko.

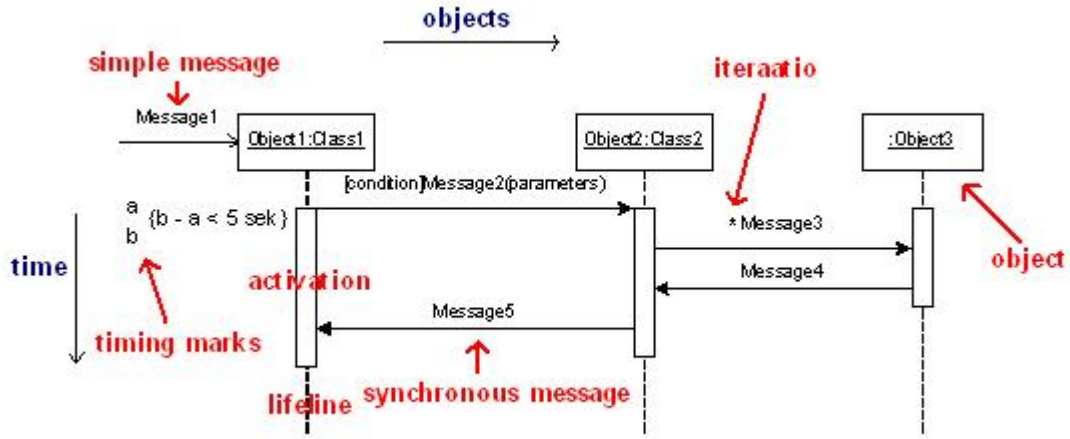
Pystysuora katkoviiva eli **olion elämänviiva** (object lifeline), kuvaa olion luomista ja sen viestintää sarjan aikana.

Viestit (yleensä metodikutsuja tai signaaleja) ovat olioiden välistä kommunikointia. Kun viesti vastaanotetaan, vastaanottavassa oliossa tapahtuu **aktivoituminen** (activation). *Aktivaatio* näyttää suorituksen kulun, tietyllä hetkellä toiminnassa olevat oliot. Aktivaatio piirretään ohuena suorakaiteena olion elämänviivan päällä aktivaatiohetkestä aina siihen hetkeen asti, jolloin olio ei enää ole aktiivinen.

Jokaisella viestillä voi olla *muoto*, jossa on nimi ja parametrit, esim. piirrä(k: kuvio). Viesteillä voi olla myös **ehtoja** (conditions), joiden on oltava tosia, jotta viesti lähetetään ja vastaanotetaan.

Haarautumisessa voidaan piirtää useita viestejä, joiden ehdoista riippuu mikä/mitkä viesteistä lähetetään.

Olio voi lähettää viestin itselleenkin, jolloin viestinuoli piirretään oliosta itseensä.



Kuva 4.2 Sekvenssikaavion notaatio.

Sekvenssikaaviota luetaan ylhäältä alaspäin, vasemmalta oikealle, jolloin nähdään viestien välitys ajan kuluessa.

4.2.2 Yleinen muoto ja ilmentymämuoto

Sekvenssikaavion ilmentymämuoto kertoo yhden tietyn tilanteen yksityiskohtaisesti, joten siinä ei ole ehtoja, haarautumia eikä toistoja. Yleinen muoto sitä vastoin kuvaa kaikki mahdolliset vaihtoehdot tilanteesta, joita on yleensä varsin monta.

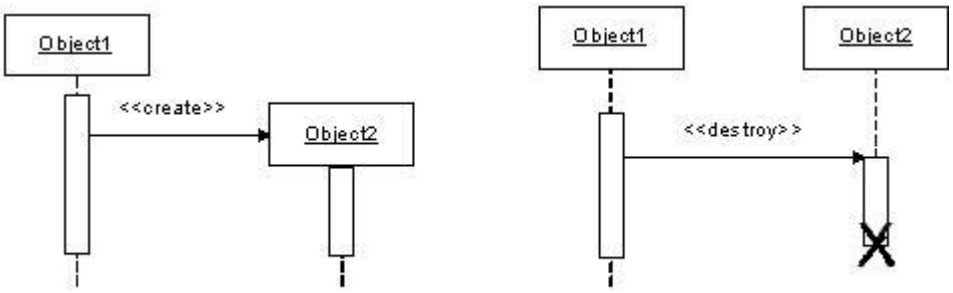
4.2.3 Toistot ja ehdot

Sekvenssikaavioiden marginaaleissa voidaan käyttää merkintöjä ja kommentteja. Ne voivat olla esimerkiksi kuvauksia toiminnoista, ajatusmerkkejä tai erilaisia ehtoja.

Sekvenssikaavion marginaaliin (tiettyyn kohtaan kuviteltua pystyssä olevaa aikajanaa) voidaan asettaa myös **aikamerkkejä** (timing mark), jotka edustavat kyseistä ajanhetkeä. Näitä aikamerkkejä voidaan käyttää ehdoissa kuvaamaan tiettyjä reaaliaikavaatimuksia, kuten viestien välillä kuluvan ajan tai viestin saapumiseen kuluvan ajan rajoittamista. *Kuvassa 4.2* on merkitty aikamerkit a ja b marginaaliin. Viestien message2 ja message3 lähetyksen välinen aika saa olla korkeintaan 5 sekuntia.

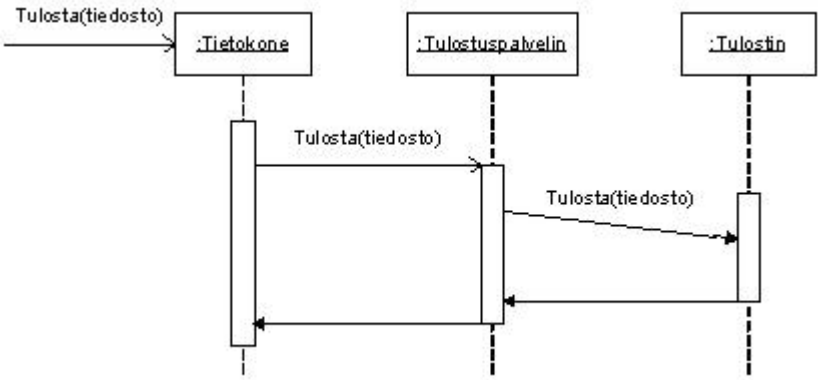
4.2.4 Olioiden luominen ja tuhoaminen sekä rekursio

Olioiden luominen ja tuhoaminen voidaan näyttää sekvenssikaaviossa. Jos olio *luo* toisen olion viestin (yleensä synkroninen) avulla, luodun olion oliolaatikko piirretään pystysuunnassa luomishetken kohdalle, viestinuolen päähän. Olion *hävittäminen* merkitään suurella X-merkillä ja lopettamalla olion elämänviiva samalla hetkellä.



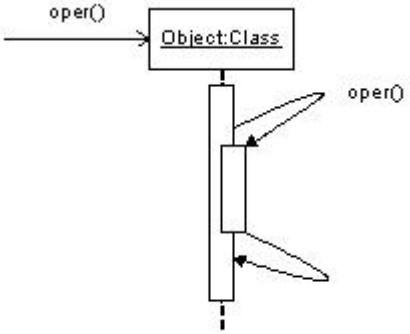
Kuva 4.3 Olioiden luominen ja tuhoaminen.

Kuvassa (Kuva 4.4) vinonuoli kertoo, että viestin lähetyksen ja vastaanoton välillä kuluu huomattavasti aikaa.



Kuva 4.4 Esimerkki sekvenssikaaviosta.

Rekursiossa (recursion) operaatio kutsuu itseään. Se kuvataan ylimääräisenä aktivaationa olion päällä (Kuva 4.5). Rekursiivinen viesti on synkroninen ja palautus näytetään yksinkertaisena viestinä. Operaatiossa on oltava rekursion lopettava ehto.

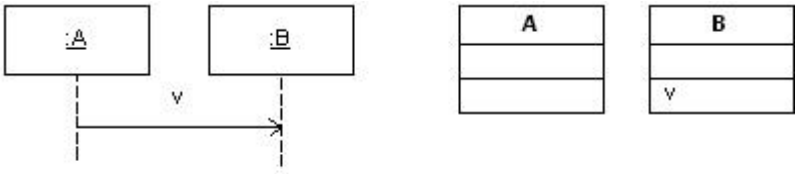


Kuva 4.5 Rekursio.

4.2.5 Sekvenssikaavion ja luokkakaavion yhteys

Vaikka sekvenssikaavio ja luokkakaavio kuvaavat ohjelmiston varsin erilaisista näkökulmista, on niillä myös yhtymäkohtia. Esimerkiksi jos sekvenssikaaviossa luokan A ilmentymä lähettää viestin v luokan B ilmentymälle, on v:n oltava B luokan operaatio. B:n ollessa aktiivinen luokka v on B:n tunnistama signaali. Molemmissa tapauksissa **v:n tulee esiintyä B:n yhteydessä luokkakaaviossa** ja mahdollisten **todellisten parametrien tulee täsmätä operaation kutsumuodon kanssa**. Luokkien A ja B välillä on myös todennäköisesti oltava assosiaatio, koska sanoman lähettäminen viittaa suhteeseen näiden luokkien välillä. Jos A-olio kommunikoi usean eri B-olion kanssa sekvenssikaavioissa, tulee assosiaation olla moninkertainen.

Kuvassa (Kuva 4.6) luokkien A ja B välillä oleva kommunikaatio (v) tulkitaan palvelupyynnöksi, joka muuttaa B:n tilaa. Tämä palvelupyyntö toteutetaan luokassa B metodina v.

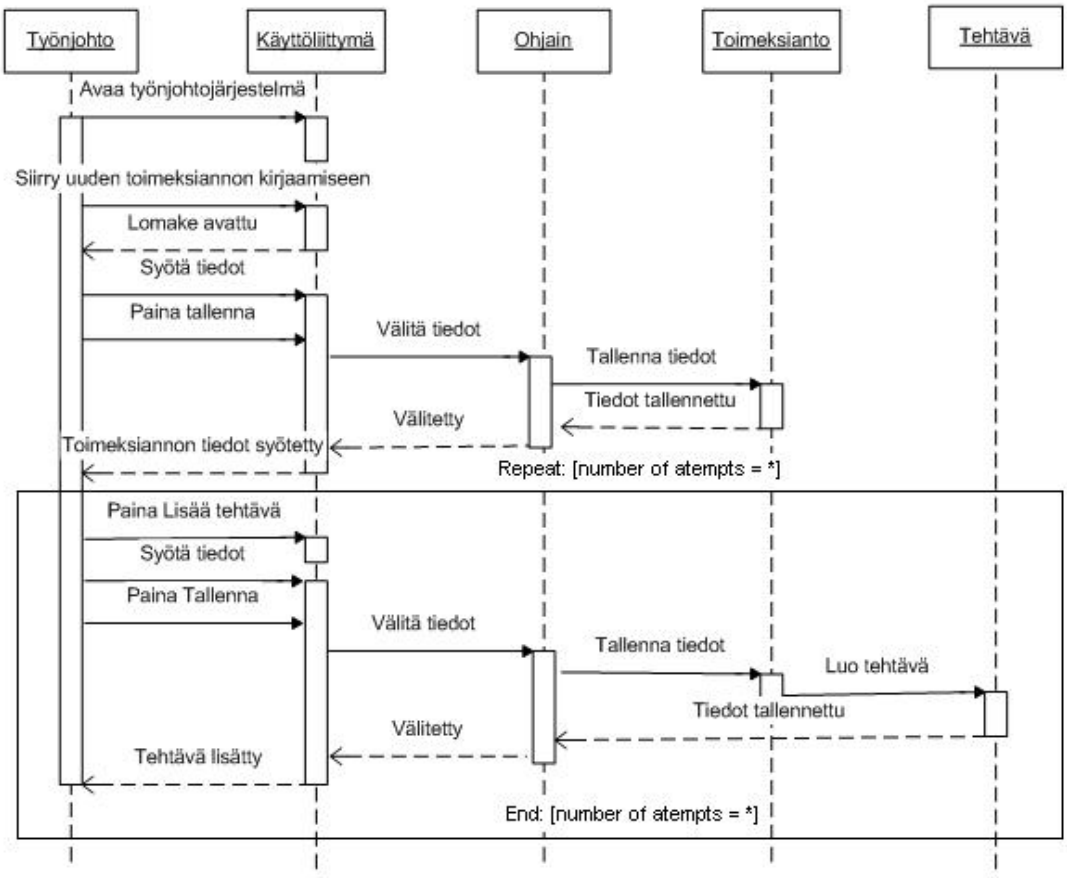


Kuva 4.6 Sekvenssi- ja luokkakaavion välinen yhteys

4.3 Esimerkkijärjestelmä

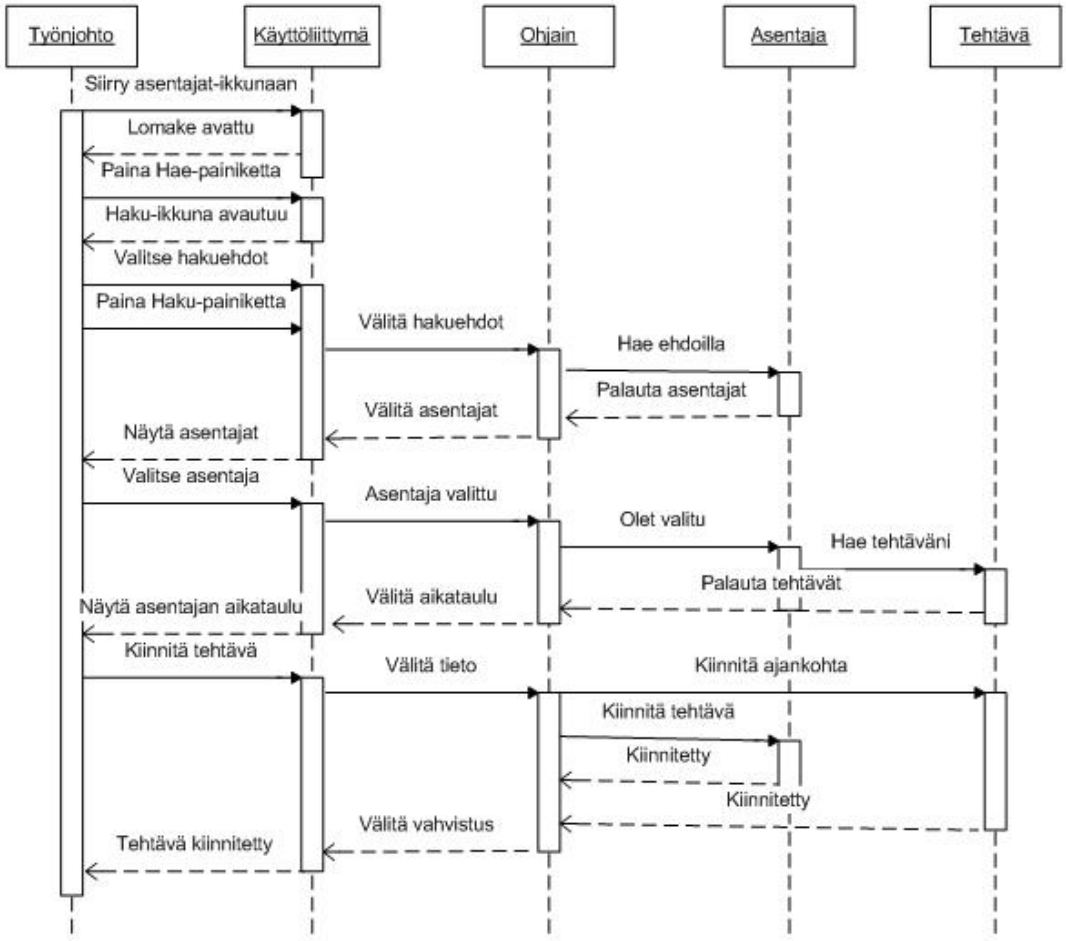
Sekvenssikaavioista on esitetty kolme käyttöliittymäkuvauksen tapahtumaa, jotka ovat Kirjaa toimeksianto, Kiinnitä toimeksianto vapaalle suunnittelijalle ja Hae lähin vapaa työntekijä. Tapahtumat näkyvät kuvissa (Kuvat 4.7, 4.8 ja 4.9).

Käyttöliittymäkuvauksen tapahtuma: Kirjaa toimeksianto



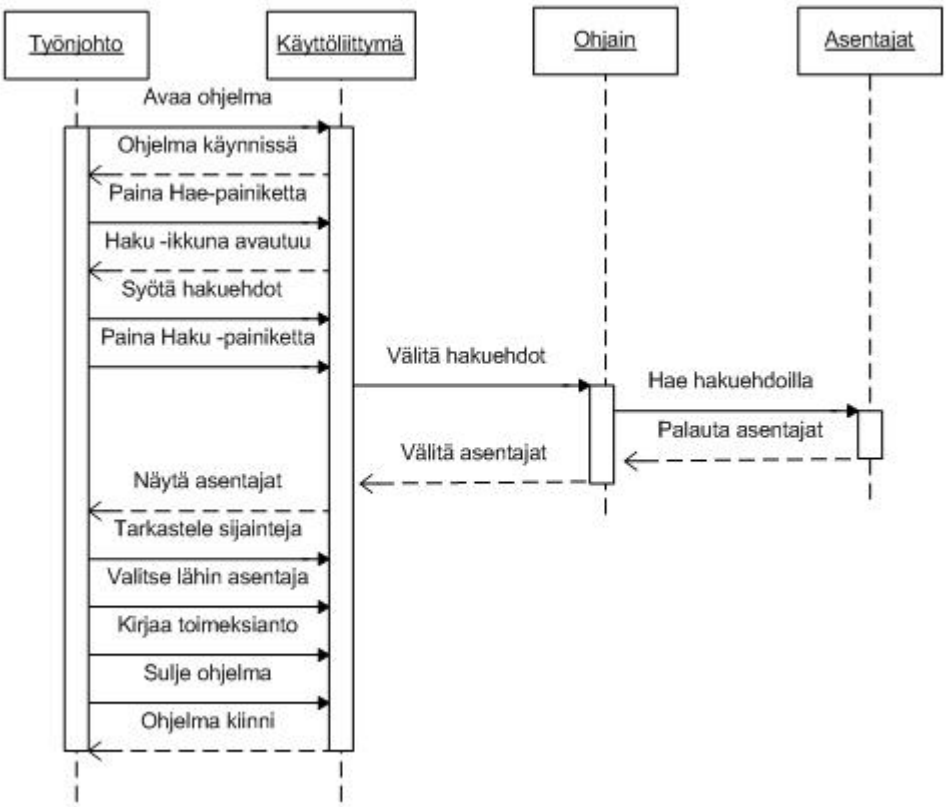
Kuva 4.7 Kirjaa toimeksianto; Sekvenssikaavio

Käyttöliittymäkuvauksen tapahtuma: Kirjaa toimeksianto vapaalle suunnittelijalle



Kuva 4.8 Kiinnitä toimeksianto vapaalle asentajalle; Sekvenssikaavio

Käyttötapausliittymän kuvaus: Hae lähin vapaa työntekijä.

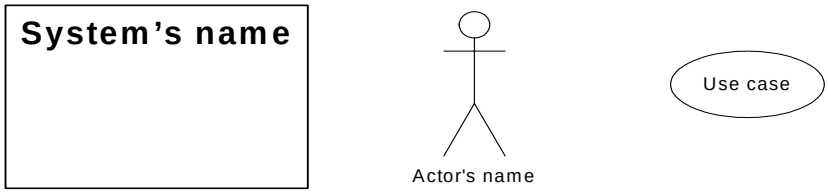


Kuva 4.9 Hae lähin vapaa työntekijä; Sekvenssikaavio

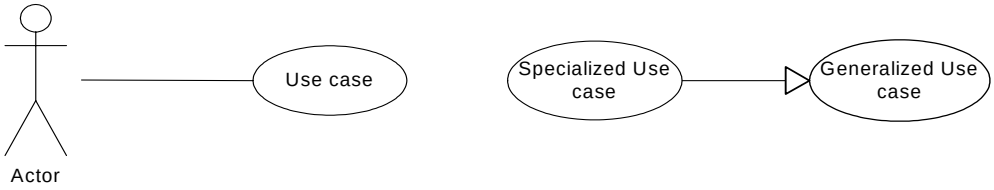
5. MALLINNUSELEMENTTEJÄ

Käyttötapauskaavio

Järjestelmä, toimija ja käyttötapaus:



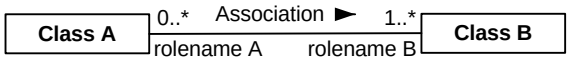
Assosiaatio ja yleistys:



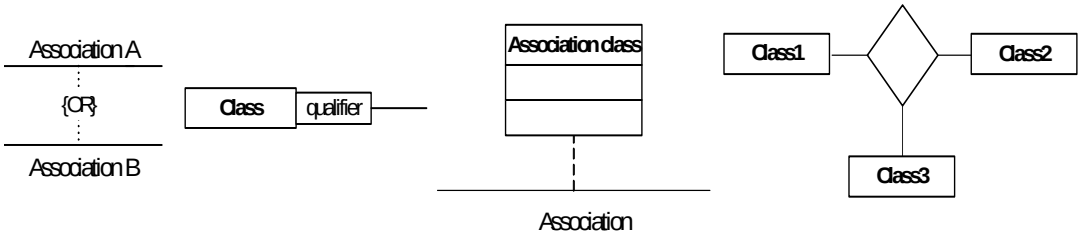
Luokkakaavio

Luokka ja assosiaatio:

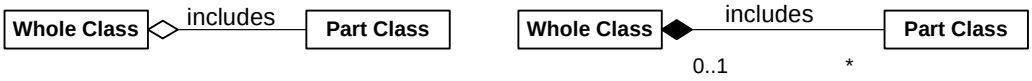
Kerrannaisuus ja roolinimet assosiaatiossa:



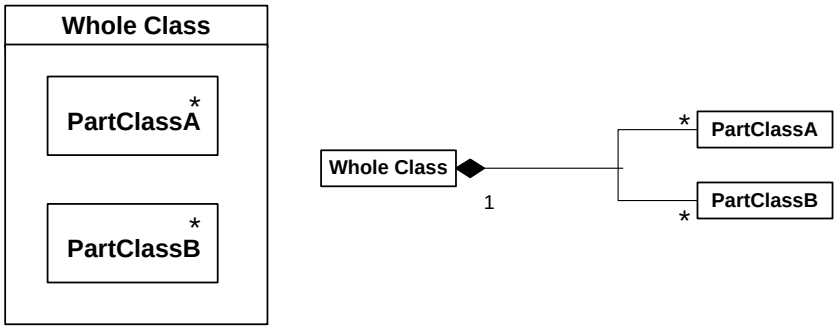
Tai-assosiaatio, valitsinassosiaatio, assosiatiivinen luokka ja kolmioassosiaatio:



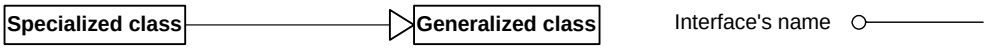
Kooste ja vahva kooste:



Vaihtoehtoja vahvalle koosteelle:

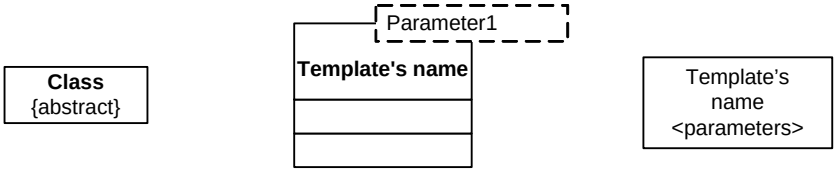


Yleistys ja rajapinta:

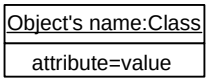


Riippuvuus ja tarkennus:

Abstrakti luokka, parametroitu luokka eli mallinne ja mallinteen ilmentymä:

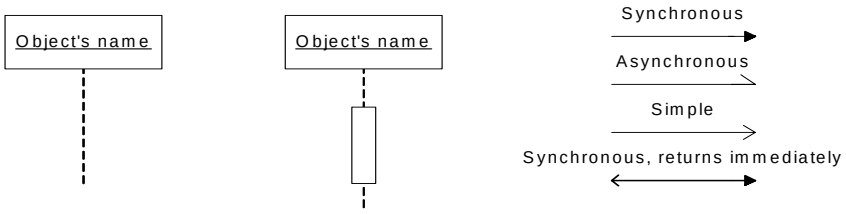


Olio:

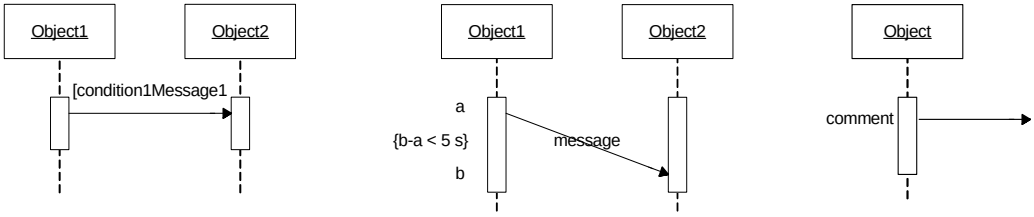


Sekvenssikaavio

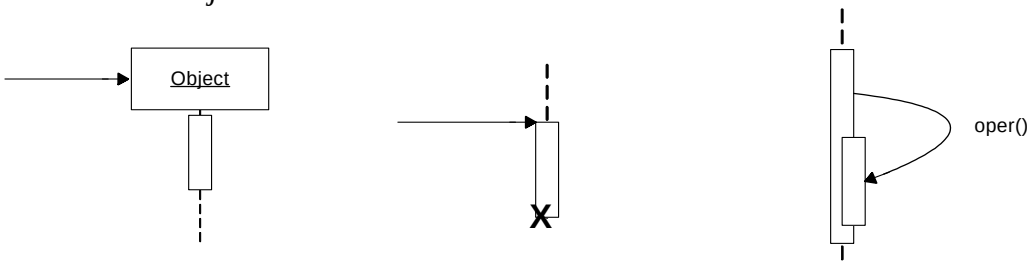
Olioilmentymä, olion aktivointi ja viestit:



Ehto, viestin lähetysaika ja skriptikommentti:

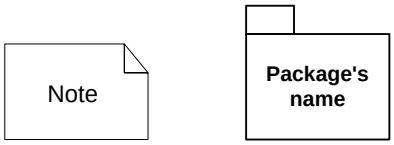


Olion luominen ja tuhoaminen sekä rekursio:



Yleisiä elementtejä

Huomautus ja paketti:



Stereotyyppi, rajoitus ja ominaisuus:

<<Stereotype's name>> {constraint} {name = value}

6. HAKEMISTO

7. SANASTO (suomi-englanti)

abstrakti luokka	abstract class
aikamerkki	timing mark
ajonaikainen komponentti	run-time component
alijärjestelmä	subsystem
aliluokka	subclass
alitila	substate
alkeistietotyyppi	primitive type
alkutila	initial state
alue	area
arvoaluemääritys	property string
asiaankuuluva	relevance
assosiaatiivinen luokka	association class
asynkroninen	asynchronous
attribuutti	attribute
desimaaliluku	real
dokumentointiominaisuus	documentation property
dynaaminen kaavio	behaviour diagram
dynaaminen malli	behaviour model
elämänviiva	object lifeline
entiteetti	entity
epätäydellinen	incomplete
erillinen	disjoint
erotin	discriminator
esiehto	precondition
etäoperaatiokutsu	Remote Procedure Call (RPC)
funktio	function
haarautua	branch
haarautuma	fork
hallintaolio	control object
historiamerkki	history indicator
huomautus	note
hyväksyntä	validation
ilmentymä	instance
jaettu koostumus	shared aggregation
johde	derivation
julkinen	public
julkisivu	facade
jälkiehto	postcondition
järjestelmä	system

jäsen……..	member
kaavio……	diagram
kaksoiskulmasulku……	guillement
kerrannaisuus……	multiplicity
kielioppisääntö……..	syntactic
kokonaisluku…….	integer
kolmioassosiaatio……..	ternary association
kommunikaatio……..	communication
komponentti……..	component
komponenttikaavio……	component diagram
komponenttinäkymä…….	component view
koostumussuhde……	aggregation
koriste……	adornment
kriittinen osa…….	critical region
käyttöliittymä……	user interface
käyttösuhde……	uses association
käyttösääntö……...	pragmatic
käyttötapaus……..	use case
käyttötapauskaavio……	use case diagram
käyttötapausmalli……..	use case model
käyttötapausnäkymä…….	use case view
käyttöönottokaavio……...	deployment diagram
käyttöönottonäkymä…….	deployment view
laajennussuhde…….	extends association
laite……...	device
looginen näkymä……..	logical view
lopputila……	final state
luokka……...	class
luokkakaavio…….	class diagram
luokkamuuttuja…….	class variable
lähetyslause……...	send-clause
malli……..	model
mallinne……	template
mallinnuselementti……...	model element
mallinnuskieli……...	modeling language
merkityssääntö……..	semantic
metodi, menetelmä……...	method
monikertainen siirtymä…….	complex transition
muodostinmetodi……..	constructor
muoto……	signature
navigointilauseke…….	navigation expression
nimetty ominaisuus……..	tagged value

notaatio	notation
numerointi	enumeration
näkymä	view
näkyvyys	visibility
ohjaaja	semaphore
ohjausluokka	semaphore class
ohjelmointi	programming
olio	object
oliokaavio	object diagram
OMG	Object Management Group
operaatio	operation
operaation muoto	operation signature
osasto	compartment
paketti	package
periminen	inheritance
peräkkäinen	sequential
piste	point
polymorfismi	polymorphism
prosessi	process
prosessori	processor
päällekkäinen	overlapping
päätoimija	primary actor
rajaolio	boundary object
rajapinta	interface
rajoitin	constraint
rekursiivinen assosiaatio	recursive association
riippuvuussuhde	dependency
samanaikainen alitila	concurrent substate
samanaikaisnäkymä	concurrency view
samanaikaisuus	concurrency
sekventiaalinen	sequential
solmu	node
suoritusjärjestys	scheduling
signaali	signal
suhde	relation
suojattu	protected
suoritussäie	execution thread
synkroninen	synchronous
säie	thread
säilyvyys	persistence
sääntö	rule
tapahtumantunniste	event-signature

tarkennussuhde……	refinement
tarkistus……	verification
taso……...	scope, layer
tietokanta……..	database
tieto-olio……...	entity object
tilakaavio……..	state diagram
tilanne……..	scenario
toimenpidetila……..	swimline
toimija……..	actor
toimintaohje…….	process
toiminto……	action
toimintokaavio…….	activity diagram
toimintolauseke……	action expression
toissijainen toimija……...	secondary actor
toteuttaa……	implement
toteutus…….	realization
toteutuskaavio……..	realization diagram
toteutusolio……..	implementation object
toteutussuhde……...	realization relationship
totuusarvo……	boolean
tukea…….	support
tuoda……	import
täydellinen……	complete
uimarata……...	swimlane
UML……	Unified Modelling Language
vahva koostumus…….	composition aggregation
valitsinassosiaatio……	qualified association
valvonta……	monitor
valvottu……	quarded
varmuusehto…….	guard condition
viestiyhteyskaavio……	sequence diagram
viestiassosiaatio……...	communication association
yhdistyminen……	join
yhteistyö……...	collaboration
yhteistyökaavio……	collaboration diagram
yleiset merkinnät…….	general mechanism
yhteys……...	connection, link
yksinkertainen……..	simple
yksityinen…….	private
yleistys ……	generalization
yliluokka……..	superclass

8. SANASTO (englanti-suomi)

abstract class……..…….... abstrakti luokka
action…… toiminto
action expression…….. toimintolauseke
activity diagram……... toimintokaavio
actor……. toimija
adornment…… koriste
aggregation……... koostumussuhde
area……... alue
association class……... assosiaatiivinen luokka
asynchronous…… asynkroninen
attribute…… attribuutti
behaviour diagram……... dynaaminen kaavio
behaviour model……... dynaaminen malli
boolean……. totuusarvo
boundary object…… rajaolio
branch……... haarautua
class…….. luokka
class diagram…… luokkakaavio
class variable…… luokkamuuttuja
collaboration……. yhteistyö
collaboration diagram…….. yhteistyökaavio
communication…… kommunikaatio
communication association…….. viestiassosiaatio
compartment…… osasto
complete……... täydellinen
complex transition…… monikertainen siirtymä
component…… komponentti
component diagram…….. komponenttikaavio
component view……... komponenttinäkymä
composition aggregation…….. vahva koostumus
concurrency view……. samanaikaisnäkymä
concurrent…… samanaikainen
concurrent substate…….. samanaikainen alitila
connection…… yhteys
constraint……. rajoitin
constructor…… muodostinmetodi
control objects……. hallintaoliot
critical region……... kriittinen osa
database…… tietokanta
dependency…….. riippuvuussuhde

deployment diagram……. käyttöönottokaavio
deployment view…….. käyttöönottonäkymä
derivation……. johde
device……laite
diagram……kaavio
discriminator…….erotin
disjoint…….. erillinen
documentation property……dokumentointiominaisuus
entity……entiteetti
entity object…….. tieto-olio
enumeration…….. numerointi
event-signature……. tapahtumantunniste
execution thread……... suoritussäie
extends association……... laajennussuhde
facade……julkisivu
final state…….. lopputila
fork……... haarautuma
function……funktio
general mechanism……... yleiset merkinnät
generalization……... yleistys
guard condition…….varmuusehto
guarded……. valvottu
guillement…….kaksoiskulmasulku
history indicator……historiamerkki
implement……toteuttaa
implementation object…….. toteutusolio
import……... tuoda
incomplete……. epätäydellinen
inheritance……. periminen
initial state……. alkutila
instance…….ilmentymä
integer……... kokonaisluku
interface……. rajapinta
join……yhdistyminen
layer…….. taso, kerros
link……yhteys
logical view…….. looginen näkymä
member…….jäsen
method…….. metodi, menetelmä
model……. malli
model element…….. mallinnuselementti
modelling language…….. mallinnuskieli

monitor……. valvoja
multiplicity……... kerrannaisuus
navigation expression……... navigointilauseke
node…….. solmu
notation……. notaatio
note……... huomautus
object…… olio
object diagram…….. oliokaavio
object lifeline…… elämänviiva
Object Management Group…….. OMG
operation……... operaatio
operation signature……... operaation muoto
overlapping……... päällekkäinen
package……. paketti
persistence…… säilyvyys
point……..piste
polymorphism…… polymorfismi
postcondition…… jälkiehto
pragmatic…….. käyttösääntö
precondition…….. esiehto
primary actor…… päätoimija
primitive type……... alkeistietotyyppi
private……... yksityinen
process…….. prosessi, toimintaohje
processor……...prosessori
programming…… ohjelmointi
property string…….. arvoaluemääritys
protected……... suojattu
public…… julkinen
qualified association……. valitsinassosiaatio
real…… desimaaliluku
realization……. toteutus
realization diagram……... toteutuskaavio
realization relationship……. toteutussuhde
recursive association…….rekursiivinen assosiaatio
refinement…….tarkennussuhde
relation……..suhde
relevance…….. asiaankuuluva
Remote Procedure Call (RPC)……. etäoperaatiokutsu
rule……... sääntö
run-time component……. ajonaikainen komponentti
scheduling…… suoritusjärjestys

scenario…… tilanne
scope…… taso
secondary actor…… toissijainen toimija
semantic……... merkityssääntö
semaphore…… ohjaaja
semaphore class…… ohjausluokka
send-clause……... lähetyslause
sequence diagram……. viestiyhteyskaavio
sequential……. sekventiaalinen, peräkkäinen
shared aggregation……... jaettu koostumus
signal…… signaali
signature……... muoto
simple……... yksinkertainen
state diagram…… tilakaavio
subclass…… aliluokka
superclass……. yliluokka
substate……. alitila
subsystem……. alijärjestelmä
support……. tukea
swimlane…….. uimarata, toimenpidetila
synchronous……. synkroninen
syntactic……... kielioppisääntö
system……... järjestelmä
tagged value……. nimetty ominaisuus
template…… mallinne
ternary association……... kolmioassosiaatio
thread…… säie
timing mark…….. aikamerkki
Unified Modelling Language……... UML
use case…… käyttötapaus
use case diagram…….. käyttötapauskaavio
use case model……. käyttötapausmalli
use case view…… käyttötapausnäkymä
user interface…… käyttöliittymä
uses association…… käyttösuhde
validation…….. hyväksyntä
verification……... tarkistus
view…….. näkymä
visibility……... näkyvyys

9. OLIOSANASTO (suomi)

Abstrakti luokka, ***abstract class:*** Luokka, jolla ei voi olla välittömiä ilmentymiä.
Abstrakti operaatio, ***abstract operation:*** Operaatio, jolle on määritelty liittymä, mutta ei toteutusta.
Abstrakti tietotyyppi, ***abstract data type:*** Operaatioidensa kautta määritelty tietotyyppi.
Aliluokka, ***subclass:*** Luokka, joka periytymishierarkiassa on jonkun muun luokan alapuolella.
Alustaja, ***constructor:*** Operaatio, jonka avulla alustetaan luokan ilmentymät.
Alustus, ***initialization:*** Olion luonnin yhteydessä suoritettavat alkutoimet kuten alkuarvojen asettaminen.
Alustusoperaatio, ***constructor:*** Operaatio, jonka avulla alustetaan luokan ilmentymät.
Arkkitehtuuri, ***architecture:*** Järjestelmän komponenteista ja niiden välisistä yhteyksistä muodostuva rakennelma.
Attribuutti, ***attribute:*** Luokkakohtaisesti esitelty luokan ilmentymiin liittyvän tiedon määrittely. Synonyymi: *ominaisuus.*
Asiainkulku, ***scenario:*** Tietyn asia käsittelytavan kuvaus.
Delegointi, ***delegation:*** Olio välittää palvelupyynnön edelleen jonkin toisen olion suoritettavaksi.
Destruktori, ***destructor:*** Olion palvelu, joka vapauttaa olion varaamat resurssit. Synonyymi: lopetusoperaatio.
Dynaaminen oliomalli, ***object behavior model:*** Olioiden käyttäytymisen kuvaus.
Elinkaari, ***life-cycle:*** Erilaisista tiloista muodostuva polku olion synnystä sen poistumiseen.
Elinkaarimalli, ***life-cycle model:*** Luokkakohtainen, luokan olioiden elinkaaren kuvaus.
Erikoistaa, erikoistaminen, erikoistus, ***specialize, specialization:*** Tapa määritellä luokka erikoistapauksena toisesta luokasta. Määriteltävä luokasta tulee tällöin tämän toisen luokan aliluokka. Määriteltävään luokkaan liitetään usein uusia piirteitä periytyvien piirteiden lisäksi tai syrjäytetään periytyviä piirteitä.
Esittely, ***declaration:*** Kuvaus, joka määrittelee luokan, attribuutin tai palvelun hahmon.
Geneerinen luokka, ***generic class, parametrized class:*** Parametroitu luokan määritys, jonka avulla voidaan muodostaa useita samanrakenteisia luokkia parametrit kiinnittämällä..
Hahmo, ***signature:*** Operaation kutsurakenteen määrittelevä parametrilista. Synonyymi: *signatuuri, kutsumuoto.*
Hierarkkinen periytyminen, ***single inheritance:*** Luokan oliot voivat periä piirteitä vain yhdeltä välittömältä yliluokalta. Vastakohta: *moniperiytyminen.*
Identiteetti, ***identity:*** Erityispiirre, jonka perusteella olio kyetään erottamaan toisesta oliosta, vaikka oliot olisivat ominaisuuksiltaan samanlaiset. Tätä piirrettä käytetään olioiden tunnistukseen.
Ilmentymä, ***instance:*** Olio tarkasteltuna luokkansa jäsenenä.
Ilmentymäkaavio, ***instance diagram:*** Kaavio, joka esittää olioita ja niiden välisiä yhteyksiä. Kaaviota käytetään yleensä esimerkkinä havainnollistamaan luokkakaavion mukaista rakennetta.
Iteraattori, ***iterator:*** Ohjelman rakenne, jota käytetään kokoelmien jäsenten läpikäynnissä.
Johdettu attribuutti, ***derived attribute:*** Ominaisuus, joka arvo voidaan päätellä olion muiden ominaisuuksien tai olion yhteyksien perusteella.
Johdettu yhteys, ***derived association:*** Yhteys, jonka olemassaolo voidaan päätellä muiden yhteyksien ja niiden osapuolten ominaisuuksien avulla.
Julkinen, ***public:*** Näkyvyysmäärittely, jonka mukaan piirre näkyy kaikille. Ks. *näkyvyys.*
Jälkeläisluokka, ***descendant class:*** Ks. *aliluokka*
Kapselointi, ***encapsulation:*** Olioiden ulkoisen liittymän ja sisäisen rakenteen erottaminen toisistaan.
Konstruktori, ***constructor:*** Ohjelmointikielen yhteydessä käytetty termi operaatiosta, jonka avulla alustetaan luokan ilmentymät. Synonyymi: *alustaja.*
Kehittämismalli, ***methodology:*** Kokoelma järjestelmän kehittämistyön tekemiseen tarkoitettuja menetelmiä.
Kokoelma, ***collection, container:*** Luokka, jonka ilmentymät ovat oliokokoelmia.
Kokoonpano, ***aggregation, composition:*** Yhteys kokonaisuuden ja sen osien välillä.
Konkreettinen luokka, ***concrete class:*** Luokka, jolla voi olla välittömiä ilmentymiä.

Kooste, koottu olio, *aggregate, composite object:* Useammasta oliosta muodostuva olio. Synonyymi: *rakenteinen olio.*
Kuormittaminen, kuormitus, *overloading:* Tilanne, jossa samalla nimellä viitataan useisiin eri palveluihin. Oikean palvelun valinta riippuu pyynnön yhteydessä annettujen parametrien tai operaattoreiden tyypeistä ja saattaa olla ratkaistavissa vasta suoritusaikaisesti.
Kutsumuoto, kutsurajapinta, *signature:* Operaation kutsurakenteen määrittelevä parametrilista.
Käyttäjä, käyttäjärooli, *actor:* Järjestelmän käyttäjä tai käyttäjän rooli.
Käyttäytyminen, *behaviour:* Luokan olioiden toimintatapa.
Käyttäytymismalli, *behavioral model:* Erilaisista tiloista muodostuva polku olion synnystä sen poistumiseen. Synonyymi: *elinkaarimalli.*
Käyttötapaus, *use case:* Järjestelmän käyttöön liittyvä työ- tai tehtäväkokonaisuus.
Käyttötapausmalli, *use case model:* Järjestelmän kaikkien käyttötapausten kuvaus.
Liiketoimintaluokka, *business class:* Liiketoimintaolioiden luokka.
Liiketoimintaolio, *business object:* Olio, joka kuvaa liiketoiminnan kohdetta tai osapuolta.
Liiketoimintasääntö, *business rule:* Liiketoimintaolioiden käyttäytymiseen tai liiketoimintatapahtumien käsittelyyn liittyvä sääntö.
Liittymä, *interface:* Mikä tahansa liittymä ohjelmien välillä, ohjelmien ja laitteiden tai ohjelmien ja käyttäjien välillä.
Liittymäolio, *interface object:* Ohjelman ulkoisen liittymän, esim. käyttöliittymän, toteutukseen käytettävä olio.
Lopetusoperaatio, *destructor:* Olion palvelu, joka vapauttaa olion varaamat resurssit. Synonyymi: *destruktori.*
Luokan ilmentymä, *class member:* Luokkaan kuuluva olio. Termiä käytetään haluttaessa korostaa sitä, että olio on luokan jäsen.
Luokka, *class:* Samankaltaisten olioiden malli.
Luokka-aihio, *generic class, parametrized class:* Parametroitu luokan määritys, jonka avulla voidaan muodostaa useita samanrakenteisia luokkia parametrit kiinnittämällä. Synonyymi: *geneerinen luokka, parametroitu luokka.*
Luokka-attribuutti, *class attribute:* Attribuutti, jonka arvo on luokka- eikä oliokohtainen, esimerkiksi luokan olioiden lukumäärä.
Luokkahierarkia, *class hierarchy:* Luokkien väliset periytymisyhteydet kuvaava rakenne.
Luokkakaavio, *class diagram:* Luokat ja niiden väliset yhteydet kuvaava kaavio.
Luokkakirjasto, *class library:* Luokkamäärityksiä sisältävä kirjasto.
Luokkaoperaatio, luokkapalvelu, *class operation:* Olioluokkaan liittyvä operaatio, jonka suorittajana ei ole mikään luokkaan kuuluva olio. Yleensä tällainen operaatio kohdistuu luokan ilmentymäkokoelmaan, esim. ilmentymän luonti.
Menetelmä, *method:* Määrämuotoinen tapa järjestelmän kehittämistyön tekemiseen.
Metaluokka, *metaclass:* Luokka, jonka ilmentymät ovat luokkia.
Metodi, *method:* Palvelun toteutus.
Moduuli, *module:* 1) Olioista, luokista ja palvelukuvauksista muodostuva kokonaisuus. 2) Käännösyksikkö.
Moniluokitus, *multiple classification:* Tilanne, jossa olio voi kuulua useaan luokkaan eikä kuuluminen perustu periytymiseen.
Monimuotoisuus, *polymorfism:* Monimerkityksellisyys. Nimen merkitys vaihtelee. **Moniperiytyminen, *multiple inheritance:*** Luokka perii piirteitä usealta välittömältä yliluokalta.
Näkyvyys, *accessibility, visibility:* Luokan tietojen ja palveluiden käytettävyys. Ohjelmointikieleen liittyvä käsite. Esimerkiksi C++-kielessä näkyvyys voi olla: julkinen = kaikkien käytettävissä, suojattu = perillisten käytettävissä, yksityinen = käytettävissä vain luokan välittömissä ilmentymissä.
Olio, *object:* Ympäristöstä erottuva kokonaisuus. Tietokokonaisuus, joka tarjoaa myös palveluja sisältämiensä tietojen käsittelyyn. Luokan ilmentymä.
Olioarkkitehtuuri, *object architecture:* Yrityksen tärkeimmät (liiketoiminta)luokat ja niiden väliset riippuvuudet kuvaava rakenne.
Oliokaavio, *object diagram:* Staattisen oliomallin kuvaesitys.

Oliokannanhallintajärjestelmä, *object database management system, ODBMS:* Oliokannan hallintaohjelmisto.
Oliokanta, *object database, ODB:* Oliorakenteinen tietokanta.
Oliokategoria, *(object) gathegory:* Ryhmä olioluokkien jaottelussa. Oliokategorioita voisivat olla esimerkisi liittymäolot, tieto-oliot ja ohjausoliot.
Olioluokka, *(object) class:* Samankaltaisten olioiden malli.
Oliomalli, *object model:* Olioiden ja niiden välisten kytkentöjen kuvaus.
Olion palvelu, olio-operaatio, *instance function:* Olion suorittama toiminta. Termiä käytetään haluttaessa korostaa sitä, että kyseessä on luokan ilmentymän suorittama toiminta.
Oliorakennemalli, *object structure model:* Olioiden ja niiden välisten yhteyksien rakenteellinen kuvaus, johon voi sisältyä sekä luokkatason kuvaus että ilmentymätason kuvauksia. Synonyymi: *staattinen oliomalli.*
Oliotunniste, *object identifier:* Olion identiteetin toteuttava mekanismi. Tapa yksilöidä olio.
Olioyhteistyökaavio, *object interaction diagram:* Jonkin tehtävän suorittamiseksi tarvittavaa olioiden välistä yhteistyötä kuvaava kaavio. Etuliite 'olio' voidaan usein jättää pois.
Operaatio, *operation:* Yhtenä kokonaisuutena pyydettävissä oleva palvelu.
Operaatiomonimuotoisuus, *operation polymorfism:* Operaatiolla on periytymishierarkian eri tasoilla sama nimi, mutta operaation toteutustapa voi olla erilainen. Käytettävä toteutustapa valitaan suorittavan olion luokan perusteella.
Osallistumisrajoite, *participation restriction, cardinality constraint:* Yhteystyyppiin ja sen osapuolten välisen kytkennän luonnehdinta, joka ilmaisee, miten monessa kyseistä tyyppiä olevassa yhteydessä kukin osapuoliluokan ilmentymä voi olla mukana.
Palvelu, *operation:* Yhtenä kokonaisuutena pyydettävissä oleva tietojenkäsittely-toimenpide. Synonyymi: *palvelu.*
Palveluliittymä, *interface:* Luokan tarjoamien palvelujen hahmojen määrittelyistä muodostuva kokonaisuus, joka esittelee palvelut ja niiden parametrit.
Palvelun toteutus, *implementation:* Tapa, jolla palvelu on toteutettu.
Parametroitu luokka, *generic class, parametrized class:* Parametroitu luokan määritys, jonka avulla voidaan muodostaa useita samanrakenteisia luokkia parametrit kiinnittämällä. Synonyymi: *luokka-aihio, geneerinen luokka.*
Perittävä luokka, *ancestor class:* Luokka, jolta tarkasteltava luokka perii jonkin ominaisuuden tai palvelun.
Periytyminen, *inheritance:* Luokka saa luokan yliluokkiin liitetyt piirteet ilman erillistä määrittelyä.
Piirre, characteristic, *feature:* Yhteisnimitys attribuutille ja palvelulle.
Pysyvyys, *persistency:* Olioiden tilan säilyminen ohjelman suorituskertojen välillä.
Pysyvä olio, *persistent object:* Olio, jonka tila säilyy ohjelman suorituskertojen välillä.
Rakenteinen olio, *aggregate, composite object:* Useammasta oliosta muodostuva olio. Synonyymi: *kooste, koottu olio.*
Ratkaisumalli, *pattern:* Tapa ratkaista jokin tehtävä, esimerkiksi oliosuunnittelun ratkaisumalli (design pattern).
Rooli, *role:* Nimi, joka kuvaa yhteyden osapuolen asemaa yhteydessä, esimerkiksi avioliitto-yhteydessä roolit olisivat 'aviomies' ja 'vaimo'.
Selain, *browser:* Ohjelmistotyökalu, jonka avulla ohjelmoija voi tutkia luokkien ominaisuuksien ja palvelujen määrittelyjä.
Signatuuri, *signature:* Operaation kutsurakenteen määrittelevä parametrilista. **Siirtymä, *state transition:*** Eteneminen vaiheesta toiseen olion elinkaaressa. Siirtymän aiheuttaa yleensä jokin tapahtuma.
Sitominen, *binding:* Nimen ja sisällön (esim. luokan tai operaation) yhdistäminen. Sitominen voi olla käännösaikaista (early binding, static binding) tai suoritusaikaista (late binding, dynamic binding).
Skenaario, *scenario:* Tietyn asian käsittelytavan kuvaus.
Sovelluskehys, *framework:* Tiettyä käyttötarkoitusta varten muodostettu tiivisti yhteenkytkettyjen luokkien kokoelma.

Staattinen oliomalli, *object structure model:* Olioiden ja niiden välisten yhteyksien rakenteellinen kuvaus. Kuvaukseen voi sisältyä sekä luokkatason kuvaus että ilmentymätason kuvauksia. Synonyymi: *olirakennemalli*.
Suojattu, *protected:* Näkyvyysmääritys, jonka mukaan periytynyt piirre näkyy luokan jälkeläisille. Termi on käytössä esim. C++ -kielessä.
Suoritusaikainen, *run time:* Ohjelman suoritusaikana tehtävä.
Suoritusaikainen sidonta, *dynamic binding:* Nimen kytkeminen sisältöön suoritusaikaisesti. Esimerkiksi palvelun suorittavan metodin valinta sen perusteella, minkä olioluokan välitön ilmentymä palvelua suorittava olio on.
Suunnittelun ratkaisumalli, *design pattern:* Suunnitteluvaiheessa sovellettava ratkaisumalli.
Syrjäyttää, *override:* Uuden toteutustavan (metodin) määrittely yliluokalta periytyvälle palvelulle.
Tapahtuma, *event:* Jokin ärsyke, johon järjestelmän pitäisi reagoida.
Tehtävä, *activity:* Joukosta toimenpiteitä muodostuva kokonaisuus, jolle voidaan määritellä kesto.
Tiedon kätkeminen, *information hiding:* Olioiden ulkoisen liittymän ja sisäisen tietorakenteen erottaminen toisistaan siten, että sisäinen tietorakenne ei ole näkyvissä olion palveluiden käyttäjille.
Tila, *state:* Vaihe olion elinkaaressa.
Toimenpide, *action:* Hetkellinen teko tai toimi.
Toteutus, *implementation:* 1) Järjestelmän kehittämisessä vaihe, jossa tuotetaan ratkaisuun tarvittavat ohjelmat. 2) Tapa, jolla palvelu on toteutettu. Synonyymi: *palvelun toteutus, metodi*.
Toteutuserä, *increment:* Yhdellä kertaa toteutettava järjestelmän osa.
Toteutuva luokka, *concrete class:* Luokka, jolla voi olla välittömiä ilmentymiä. **Tyyppi, type**: Olion tai tiedon rakenteen määrittelevä malli.
Uudelleenkäyttö, uuskäyttö, *re-use:* Kuvauksen käyttö muussa kuin sen alkuperäisessä käyttöyhteydessä.
Vastuu, *responsibility:* Kokoelma palveluita, jotka yhdessä kattavat jonkin asian hoitamiseen tarvittavat toimet.
Vastuupohjainen suunnittelu, *responsibility driven design:* Olioiden määrittely niille asetetun toiminnallisen vastuun perusteella.
Viesti, *message:* Oliolle lähetettävä palvelupyyntö tai sanoma, johon olio reagoi käynnistämällä halutun palvelun.
Viestien kulku, *event trace:* Kuvaus viestien kulusta olioiden välillä.
Väistyvä, *virtual:* Piirre, joka voidaan syrjäyttää.
Välillinen ilmentymä, *indirect instance:* Ilmentymä on luokan kannalta välillinen, jos se on luokan jonkin jälkeläisluokan välitön ilmentymä.
Välitön ilmentymä, *direct instance:* Ohjelmassa olio esitellään luontinsa yhteydessä jonkin luokan ilmentymäksi. Tällöin olio on kyseisen luokan välitön ilmentymä. Ilmentymien välillisyys ja välittömyys ovat ohjelmointitason käsitteitä.
Yhteistyö, *collaboration:* Palvelujen hyväksikäyttö jonkin tehtävän suorittamiseksi.
Yhteistyökaavio, *object interaction diagram:* Jonkin tehtävän suorittamiseksi tarvittavaa olioiden välistä yhteistyötä kuvaava kaavio.
Yhteydenpitomalli, *communication model:* Liiketoiminnan osapuolten välistä yhteydenpitoa kuvaava kaavio.
Yhteys, yhteystyyppi, *association (type), relationship (type):* Kahden tai useamman olion välinen kytkentä, joka määritellään tyyppitasolla yhteystyyppinä kytkennän osapuolten luokkien välille. Yhteys on yhteystyypin ilmentymä.
Yksityinen, *private:* Näkyvyysmääritys, jonka mukaan piirre näkyy vain luokan välittömille ilmentymille.
Yleistää, *generalize:* Tapa määritellä luokka siten, että luokkaan kootaan yhteiset piirteet joukosta luokkia, joista tehdään määriteltävän luokan aliluokkia.
Yleistys, yleistyssuhde, *generalization:* 1) Luokan ja sen aliluokkien välinen suhde. 2) Yleistämällä aikaansaatu luokka.
Yliluokka, *superclass:* Luokka, joka on perintähierarkiassa toisen luokan yläpuolella.

10. OLIOSANASTO (englanti)

Abstract class, *abstrakti luokka:* A class that cannot be directly instantiated. Contrast: *concrete class.*
Abstraction, *abstraktio:* The essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer.
Action, *toimenpide:* The specification of an executable statement that forms an abstraction of a computational procedure. An action typically results in a change in the state of the system, and can be realized by sending a message to an object or modifying a link or a value of an attribute.
Action sequence, *toimenpiteiden järjestys:* An expression that resolves to a sequence of actions.
Action state, *toimenpiteen tila:* A state that represents the execution of an atomic action, typically the invocation of an operation.
Activation, *toimenpiteen suoritus:* The execution of an action.
Active class, *aktiivinen luokka:* A class whose instances are active objects. See: *active object.*
Active object, *aktiivinen olio:* An object that owns a thread and can initiate control activity. An instance of active class. See: *active class, thread.*
Activity graph, *toimenpidekaavio:* A special case of a state machine that is used to model processes involving one or more classifiers. Contrast: *statechart diagram.*
Actor [class], *käyttäjä, toimija:* A coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each use case with which it communicates.
Actual parameter, *todellinen parametri:* Synonym: *argument.*
Aggregate [class], *kooste, koottu olio:* A class that represents the "whole" in an aggregation (whole-part) relationship. See: *aggregation.*
Aggregation, *kokoonpano:* A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. See: *composition.*
Analysis, *analyysi:* The part of the software development process whose primary purpose is to formulate a model of the problem domain. Analysis focuses what to do, design focuses on how to do it. Contrast: *design*.
Analysis time, *analyysivaihe:* Refers to something that occurs during an analysis phase of the software development process. See: *design time, modelling time.*
Architecture, *arkkitehtuuri:* The organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.
Argument, *todellinen parametri:* A binding for a parameter that resolves to a run-time instance. Synonym: *actual parameter*. Contrast: *parameter*.
Artefact, *tuotos:* A piece of information that is used or produced by a software development process. An artifact can be a model, a description, or software. Synonym: *product*.
Association, *(asia)yhteys:* The semantic relationship between two or more classifiers that specifies connections among their instances.
Association class, *yhteysluokka, linkkiluokka*: A model element that has both association and class properties. An association class can be seen as an association that also has class properties, or as a class that also has association properties.
Association end, *yhteyden loppukohta*: The endpoint of an association, which connects the association to a classifier.
Asynchronous action, *tahdistamaton toimenpide:* A request where the sending object does not pause to wait for results. Contrast: *synchronous action*.
Attribute, *attribuutti, ominaisuus*: A feature within a classifier that describes a range of values that instances of the classifier may hold.
Behaviour, *käyttäytyminen:* The observable effects of an operation or event, including its results.
Behavioural feature, *käyttäytymistapa:* A dynamic feature of a model element, such as an operation or method.
Behavioral model aspect, *käyttäytymisen mallintaminen:* A model aspect that emphasizes the behaviour of the instances in a system, including their methods, collaborations, and state histories.
Binary association, *kahdenvälinen yhteys*: An association between two classes. A special case of an n-ary association.

Binding, *sitominen (käännös- tai suoritusaikainen):* The creation of a model element from a template by supplying arguments for the parameters of the template.
Boolean, *boolen totuusarvo:* An enumeration whose values are true and false.
Boolean expression, *boolen lauseke:* An expression that evaluates to a boolean value.
Cardinality *lukumäärä, osallistumisrajoite:* The number of elements in a set. Contrast: *multiplicity.*
Child, *lapsi(luokka):* In a generalization relationship, the specialization of another element, the parent. See: *subclass, subtype*. Contrast: *parent.*
Call, *kutsu* : An action state that invokes an operation on a classifier.
Class, (olio)*luokka:* A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See: *interface*.
Classifier, *lajitin:* A mechanism that describes behavioural and structural features. Classifiers include interfaces, classes, datatypes, and components.
Class diagram, *luokkakaavio:* A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.
Client, *asiakas* : A classifier that requests a service from another classifier.
Collaboration, *yhteistyö:* The specification of how an operation or classifier, such as a use case, is realized by a set of classifiers and associations playing specific roles used in a specific way. The collaboration defines an interaction. See: *interaction*.
Collaboration diagram, *yhteistyökaavio:* A diagram that shows interactions organized around the structure of a model, using either classifiers and associations or instances and links. Unlike a sequence diagram, a collaboration diagram shows the relationships among the instances. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. See: *sequence diagram*.
Comment, *kommentti:* An annotation attached to an element or a collection of elements. A note has no semantics. Contrast: *constraint.*
Communication association, *kommunikointiyhteys:* In a deployment diagram an association between nodes that implies a communication. See: *deployment diagram.*
Compile time, *käännösaikainen:* Refers to something that occurs during the compilation of a software module. See: *modelling time, run time*.
Component, *komponentti:* A physical, replaceable part of a system that packages implementation and conforms to and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files.
Component, diagram *komponenttikaavio:* A diagram that shows the organizations and dependencies among components.
Composite [class], *kooste[luokka]*: A class that is related to one or more classes by a composition relationship. See: *composition*.
Composite aggregation, *kooste, kokoonpano:* Synonym: *composition.*
Composite state, *tilan kooste*: A state that consists of either concurrent (orthogonal) substates or sequential (disjoint) substates. See: *substate.*
Composite substate, *koosteen alitila:* A substate that can be held simultaneously with other substates contained in the same composite state. Synonym: *region*. See: *composite substate.*
Composition, *vahva kooste:* A form of aggregation association with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it (i.e., they share lifetimes). Such parts can also be explicitly removed before the death of the composite. Composition may be recursive. Synonym: *composite aggregation.*
Concrete class, *konkreettinen (toteutuva) luokka, jolla voi olla ilmentyminä olioita*: A class that can be directly instantiated. Contrast: *abstract class.*
Concurrency, *rinnakkaisuus (limittämällä tai säikeistämällä):* The occurrence of two or more activities during the same time interval. Concurrency can be achieved by interleaving or simultaneously executing two or more threads. See: *thread.*
Concurrent substate, *rinnakkainen tila:* A substate that can be held simultaneously with other substates contained in the same composite state. See: *composite state.* Contrast: *disjoint substate.*

Constraint, *rajoite*: A semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML. See: *tagged value, stereotype.*

Container, *kokoelma:* 1. An instance that exists to contain other instances, and that provides operations to access or iterate over its contents. (for example, arrays, lists, sets). 2. A component that exists to contain other components.

Containment hierarchy, *sisältöhierarkia:* A namespace hierarchy consisting of model elements, and the containment relationships that exist between them. A containment hierarchy forms an acyclic graph.

Context, *viitekehys, asiayhteys, konteksti:* A view of a set of related modelling elements for a particular purpose, such as specifying an operation.

Datatype, *tietotyyppi:* A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations.

Defining model, *määrittelymalli:* The model on which a repository is based. Any number of repositories can have the same defining model.

Delegation, *delegointi:* The ability of an object to issue a message to another object in response to a message. Delegation can be used as an alternative to inheritance. Contrast: *inheritance*.

Dependency, *riippuvuus:* A relationship between two modelling elements, in which a change to one modelling element (the independent element) will affect the other modelling element (the dependent element).

Deployment diagram, *toteutuskaavio, komponenttikaavio:* A diagram that shows the configuration of run-time processing nodes and the components, processes, and objects that live on them. Components represent run-time manifestations of code units. See: *component diagrams*.

Derived element, *johdettu alkio:* A model element that can be computed from another element, but that is shown for clarity or that is included for design purposes even though it adds no semantic information.

Design, *suunnittelu:* The part of the software development process whose primary purpose is to decide how the system will be implemented. During design strategic and tactical decisions are made to meet the required functional and quality requirements of a system.

Design time, *suunnitteluvaihe:* Refers to something that occurs during a design phase of the software development process. See: *modelling time*. Contrast: *analysis time*.

Development process, *kehitysprosessi:* A set of partially ordered steps performed for a given purpose during software development, such as constructing models or implementing models.

Diagram, *kaavio:* A graphical presentation of a collection of model elements, most often rendered as a connected graph of arcs (relationships) and vertices (other model elements). UML supports the following diagrams: class diagram, object diagram, use case diagram, sequence diagram, collaboration diagram, state diagram, activity diagram, component diagram, and deployment diagram.

Disjoint substate, *erillinen alitila:* A substate that cannot be held simultaneously with other substates contained in the same composite state. See: composite state. Contrast: *concurrent substate*.

Distribution unit, *hajautusyksikkö:* A set of objects or components that are allocated to a process or a processor as a group. A distribution unit can be represented by a run-time composite or an aggregate.

Domain, *kohdealue:* An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area.

Dynamic classification, *dynaaminen luokittelu:* A semantic variation of generalization in which an object may change type or role. Contrast: *static classification*.

Element, *alkio:* An atomic constituent of a model.

Entry action, *aloitustoimenpide (tilamallissa):* An action executed upon entering a state in a state machine regardless of the transition taken to reach that state.

Enumeration, *luetteleminen:* A list of named values used as the range of a particular attribute type. For example, RGBColor = {red, green, blue}. Boolean is a predefined enumeration with values from the set {false, true}.

Event, *tapahtuma:* The specification of a significant occurrence that has a location in time and space. In the context of state diagrams, an event is an occurrence that can trigger a transition.

Exit action, *lopetustoimenpide (tilamallissa):* An action executed upon exiting a state in a state machine regardless of the transition taken to exit that state.

Export, *tehdä näkyväksi:* In the context of packages, to make an element visible outside its enclosing namespace. See: *visibility.* Contrast: *import*.

Expression, ***lauseke, ilmaisu:*** A string that evaluates to a value of a particular type. For example, the expression "(7 + 5 * 3)" evaluates to a value of type number.

Extend, ***laajentaa*****:** A relationship from an extension use case to a base use case, specifying how the behaviour defined for the extension use case can be inserted into the behaviour defined for the base use case.

Feature, ***piirre:*** A property, like operation or attribute, which is encapsulated within a classifier, such as an interface, a class, or a data type.

Final state, ***lopputila (tilamallissa):*** A special kind of state signifying that the enclosing composite state or the entire state machine is completed.

Fire, ***laukaista (tilasiirtymä):*** To execute a state transition. See: *transition*.

Focus of control, ***kontrollipiste:*** A symbol on a sequence diagram that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

Formal parameter, (*muodollinen) parametri:* Synonym: *parameter*.

Framework, ***(sovellus)kehys*****:** A micro-architecture that provides an extensible template for applications within a specific domain.

Generalizable element, ***yleistettävä alkio:*** A model element that may participate in a generalization relationship. See: *generalization*.

Generalization, ***yleistys(suhde):*** A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. See: *inheritance.*

Guard condition, ***rajoite-ehto:*** A condition that must be satisfied in order to enable an associated transition to fire.

Implementation, ***(palvelun) toteutus:*** A definition of how something is constructed or computed. For example, a class is an implementation of a type; a method is an implementation of an operation.

implementation inheritance, ***rakenteen (attribuuttien ja operaatioiden) sekä koodin perintä:*** The inheritance of the implementation of a more specific element. Includes inheritance of the interface. Contrast: *interface inheritance.*

Import, ***riippuvuus, näkyvyys (paketeilla):*** In the context of packages, a dependency that shows the packages whose classes may be referenced within a given package (including packages recursively embedded within it). Contrast: *export.*

Include, ***sisältyvä käyttötapaus:*** A relationship from a base use case to an inclusion use case, specifying how the behaviour defined for the inclusion use case can be inserted into the behaviour defined for the base use case.

Inheritance, ***periytyminen:*** The mechanism by which more specific elements incorporate structure and behaviour of more general elements related by behaviour. See: *generalization.*

Instance, ***ilmentymä:*** An entity to which a set of operations can be applied and which has a state that stores the effects of the operations. See: *object.*

Interaction, ***vuorovaikutus:*** A specification of how stimuli are sent between instances to perform a specific task. The interaction is defined in the context of a collaboration. See *collaboration.*

Interaction diagram, ***vuorovaikutuskaavio:*** A generic term that applies to several types of diagrams that emphasize object interactions. These include: collaboration diagrams, sequence diagrams, and activity diagrams.

Interface, ***liittymä:*** A named set of operations that characterize the behaviour of an element.

Interface inheritance, ***liittymän periytyminen:*** The inheritance of the interface of a more specific element. Does not include inheritance of the implementation. Contrast: *implementation inheritance.*

Internal transition, ***tilasiirtymä samaan tilaan:*** A transition signifying a response to an event without changing the state of an object.

Layer, ***kerros:*** The organization of classifiers or packages at the same level of abstraction. A layer represents a horizontal slice through an architecture, whereas a partition represents a vertical slice. Contrast: *partition*.

Link, ***yhteys, linkki, yhteystyypin ilmentymä:*** A semantic connection among a tuple of objects. An instance of an association. See: *association.*

Link end, ***yhteyden pää:*** An instance of an association end. See: *association end*.

Message, *viesti, palvelupyyntö:* A specification of the conveyance of information from one instance to another, with the expectation that activity will ensue. A message may specify the raising of a signal or the call of an operation.

Metaclass, *metaluokka, jonka ilmentymät ovat luokkia:* A class whose instances are classes. Metaclasses are typically used to construct metamodels.

Meta-metamodel, *meta-metamalli:* A model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.

Metamodel, *metamalli:* A model that defines the language for expressing a model.

Metaobject, *metaolio:* A generic term for all metaentities in a metamodelling language. For example, metatypes, metaclasses, metaattributes, and metaassociations.

Method, *metodi, menetelmä:* The implementation of an operation. It specifies the algorithm or procedure associated with an operation.

Model, *malli:* A semantically closed abstraction of a subject system. See: *system*.
Model aspect, *mallin näkökulma:* A dimension of modelling that emphasizes particular qualities of the metamodel. For example, the structural model aspect emphasizes the structural qualities of the metamodel.

Model elaboration, *mallin installointi:* The process of generating a repository type from a published model. Includes the generation of interfaces and implementations which allows repositories to be instantiated and populated based on, and in compliance with, the model elaborated.

Model element, *mallin element:* An element that is an abstraction drawn from the system being modelled. Contrast: *view element*.

Modeling time, *suunnitteluaikainen:* Refers to something that occurs during a modelling phase of the software development process. It includes analysis time and design time. Usage note: When discussing object systems, it is often important to distinguish between modelling-time and run-time concerns. See: *analysis time, design time*. Contrast: *run time*.

Module, *mooduuli (käännösaikainen, binäärikoodinen, suoritettava):* A software unit of storage and manipulation. Modules include source code modules, binary code modules, and executable code modules. See: *component*.

Multiple classification, *moniluokitus:* A semantic variation of generalization in which an object may belong directly to more than one class. See: *dynamic classification*.

Multiple inheritance, *moniperiytyminen, moniperintä:* A semantic variation of generalization in which a type may have more than one supertype. Contrast: *single inheritance*.

Multiplicity, *monimuotoisuus:* A specification of the range of allowable cardinalities that a set may assume. Multiplicity specifications may be given for roles within associations, parts within composites, repetitions, and other purposes. Essentially a multiplicity is a subset of the non-negative integers. Contrast: *cardinality*.

Multi-valued, *moniarvoinen*: A model element with multiplicity defined whose Multiplicity Type: upper attribute is set to a number greater than one. The term multi-valued does not pertain to the number of values held by an attribute, parameter, etc. at any point in time. Contrast: *single-valued*.

N-ary association, *n-yhteys, yhteydessä on mukana vähintään 3 luokkaa:* An association among three or more classes. Each instance of the association is an n-tuple of values from the respective classes. Contrast: *binary association*.

Name, *nimi:* A string used to identify a model element.

Namespace, *nimiavaruus:* A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning. See: *name*.

Node, *solmu, suoritusaikainen koneresurssi:* A node is classifier that represents a run-time computational resource, which generally has at least a memory and often processing capability. Run-time objects and components may reside on nodes.

Object, *olio, luokan ilmentymä*: An entity with a well-defined boundary and identity that encapsulates state and behaviour. State is represented by attributes and relationships, behaviour is represented by operations, methods, and state machines. An object is an instance of a class. See: *class, instance*.

Object diagram, *luokkakaavio:* A diagram that encompasses objects and their relationships at a point in time. It may be considered a special case of a class diagram or a collaboration diagram. See: *class diagram, collaboration diagram.*

Object flow state, *olion tila toimintakaaviossa:* A state in an activity graph that represents the passing of an object from the output of actions in one state to the input of actions in another state.

Object lifeline, *olion elinkaari (sekvenssikaaviossa):* A line in a sequence diagram that represents the existence of an object over a period of time. See: *sequence diagram*.
Operation, *palvelu:* A service that can be requested from an object to effect behaviour. An operation has a signature, which may restrict the actual parameters that are possible.
Package, *paketti:* A general-purpose mechanism for organizing elements into groups. Packages may be nested within other packages.
Parameter, *parametri (palvelussa, viestissä tai tapahtumassa):* The specification of a variable that can be changed, passed, or returned. A parameter may include a name, type, and direction. Parameters are used for operations, messages, and events. Synonyms: *formal parameter*. Contrast: *argument*.
Parameterized element, *parametroitu alkio:* The descriptor for a class with one or more unbound parameters. Synonym: *template*.
Parent, *vanhempi, isäluokka perintäyhteydessä:* In a generalization relationship, the generalization of another element, the child. See: *subclass*, *subtype*. Contrast: *child*.
Participates, *osapuoli:* The connection of a model element to a relationship or to a reified relationship. For example, a class participates in an association, an actor participates in a use case.
Partition, *ositus, osio:* 1. activity graphs: A portion of an activity graphs that organizes theresponsibilities for actions. See: *swimlane*. 2. architecture: A subset of classifiers or packages at the same level of abstraction. A partition represents a vertical slice through an architecture, whereas a layer represents a horizontal slice. Contrast: *layer*.
Persistent object, *pysyvä olio*: An object that exists after the process or thread that created it has ceased to exist.
Postcondition, *jälkiehto:* A constraint that must be true at the completion of an operation.
Precondition, *esiehto:* A constraint that must be true when an operation is invoked.
Primitive type, *alkeistyyppi:* A pre-defined basic datatype without any substructure, such as an integer or a string.
Process, *prosessi:* 1. A heavyweight unit of concurrency and execution in an operating system. Contrast: *thread*, which includes heavyweight and lightweight processes. If necessary, an implementation distinction can be made using stereotypes. 2. A software development process—the steps and guidelines by which to develop a system. 3. To execute an algorithm or otherwise handle something dynamically.
Projection, *projektio*: A mapping from a set to a subset of it.
Property, *ominaisuus:* A named value denoting a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others may be user defined. See: *tagged value*.
Pseudo-state, *näennäistila:* A vertex in a state machine that has the form of a state, but doesn't behave as a state. Pseudo-states include initial and history vertices.
Published model, *julkaistu (valmis) malli:* A model, which has been frozen, and becomes available for instantiating repositories and for the support in defining other models. A frozen model's model elements cannot be changed.
Qualifier, *tarkennus:* An association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.
Receive [a message], *(sanoman) vastaanotto:* The handling of a stimulus passed from a sender instance. See: *sender, receiver*.
Receiver [object], *vastaanottaja, toteuttaja, tuottaja:* The object handling a stimulus passed from a sender object. Contrast: *sender*.
Reception, *vastaanotto:* A declaration that a classifier is prepared to react to the receipt of a signal.
Reference, *viittaus*: 1. A denotation of a model element. 2. A named slot within a classifier that facilitates navigation to other classifiers. Synonym: *pointer*.
Refinement, *tarkennus:* A relationship that represents a fuller specification of something that has already been specified at a certain level of detail. For example, a design class is a refinement of an analysis class.
Relationship, *yhteys, yhteystyyppi:* A semantic connection among model elements. Examples of relationships include associations and generalizations.
Repository, *tietohakemisto:* A storage place for object models, interfaces, and implementations.
Requirement, *vaatimus:* A desired feature, property, or behaviour of a system.
Responsibility, *vastuu:* A contract or obligation of a classifier.
Reuse, *uudelleenkäyttö:* The use of a pre-existing artifact.

Role, *rooli:* The named specific behaviour of an entity participating in a particular context. A role may be static (e.g. an association end) or dynamic (e.g. a collaboration role).
Run time, *ajoaikainen:* The period of time during which a computer program executes. Contrast: *modelling time.*
Scenario, *skenaario, käsittelytavan kuvaus:* A specific sequence of actions that illustrates behaviours. A scenario may be used to illustrate an interaction or the execution of a use case instance. See: *interaction.*
Schema, *kaava, kaavio:* Analogous to a package which is a container of model elements. Contrast: *metamodel, package.*
Semantic variation point, *semanttinen tulkintapiste:* A point of variation in the semantics of a metamodel. It provides an intentional degree of freedom for the interpretation of the metamodel semantics.
Send [a message], *lähettää (viesti):* The passing of a stimulus from a sender instance to a receiver instance. See: *sender, receiver*.
Sender [object], *lähettäjä, asiaka:* The object passing a stimulus to a receiver object. Contrast: *receiver*.
Sequence diagram, *sekvenssikaavio:* A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in the interaction and the sequence of messages exchanged. Unlike a collaboration diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario). Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. See: *collaboration diagram*.
Signal, *viesti:* The specification of an asynchronous stimulus communicated between instances. Signals may have parameters.
Signature, *hahmo:* The name and parameters of a behavioural feature. A signature may include an optional returned parameter.
Single inheritance, *hierarkkinen periytyminen, yksiperintä:* A semantic variation of generalization in which a type may have only one supertype. Contrast: *multiple inheritance.*
Single valued, *yksiarvoinen:* A model element with multiplicity defined is single valued when its Multiplicity Type: upper attribute is set to one. The term single-valued does not pertain to the number of values held by an attribute, parameter, etc., at any point in time, since a single-valued attribute (for instance, with a multiplicity lower bound of zero) may have no value. Contrast: *multi-valued*.
Specification, *määritys:* A declarative description of what something is or does. Contrast: *implementation*.
State, *tila:* A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
Statechart diagram, *tilakaavio:* A diagram that shows a state machine. See: *state machine*.
State machine, *tila-automaatti:* A behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.
Static classification, *staattinen luokittelu:* A semantic variation of generalization in which an object may not change type or may not change role. Contrast: *dynamic classification.*
Stereotype, *stereotyyppi:* A new type of modelling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of three extensibility mechanisms in UML. See: *constraint, tagged value.*
Stimulus, *heräte:* The passing of information from one instance to another, such as raising a signal or invoking an operation. The receipt of a signal is normally considered an event. See: *message*.
String, *merkkijono:* A sequence of text characters. The details of string representation depend on implementation, and may include character sets that support international characters and graphics.
Structural feature, *rakenteellinen ominaisuus:* A static feature of a model element, such as an attribute.
Structural model, aspect *rakennenäkymä:* A model aspect that emphasizes the structure of the objects in a system, including their types, classes, relationships, attributes, and operations.
Subactivity state, *toimintakaavion alitila:* A state in an activity graph that represents the execution of a non-atomic sequence of steps that has some duration.

Subclass, *aliluokka*: In a generalization relationship, the specialization of another class; the superclass. See: *generalization*. Contrast: *superclass*.
Submachine state, *tila-automaatin tila:* A state in a state machine which is equivalent to a composite state but its contents is described by another state machine.
Substate, *alitila:* A state that is part of a composite state. See: *concurrent state, disjoint state*.
Subsystem, o*sasysteemi, alisysteemi* : A subsystem is a grouping of model elements, of which some constitute a specification of the behaviour offered by the other contained model elements. See *package, system*.
Subtype, *alityyppi:* In a generalization relationship, the specialization of another type; the supertype. See: *generalization*. Contrast: *supertype*.
Superclass, *yliluokka:* In a generalization relationship, the generalization of another class; the subclass. See: *generalization.* Contrast: *subclass*.
Supertype, *päätyyppi:* In a generalization relationship, the generalization of another type; the subtype. See: *generalization*. Contrast: *subtype*.
Supplier, *tuottaja:* A classifier that provides services that can be invoked by others. Contrast: *client*.
Swimlane, *uimarata:* A partition on a activity diagram for organizing the responsibilities for actions. Swimlanes typically correspond to organizational units in a business model. See: *partition*.
Synch state, *tahdistettu tila* : A vertex in a state machine used for synchronizing the concurrent regions of a state machine.
Synchronous action, *tahdistettu toiminta* : A request where the sending object pauses to wait for results. Contrast: *asynchronous action*.
System, *systeemi:* 1. A collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. Synonym: physical system. 2. A top-level subsystem.
Tagged value, *merkitty arvo, ominaisuuden nimi ja arvo:* The explicit definition of a property as a name-value pair. In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML; others may be user defined. Tagged values are one of three extensibility mechanisms in UML. See: *constraint, stereotype.*
Template, *kaavain, luotta:* Synonym: *parameterized element*.
Thread, *säie:* A single path of execution through a program, a dynamic model, or some other representation of control flow. Also a stereotype for the implementation of an active object as lightweight process. See *process*.
Time, *aika:* A value representing an absolute or relative moment in time.
Time event, *ajoitettu tapahtuma:* An event that denotes the time elapsed since the current state was entered. See: *event*.
Time expression, *aikailmaisu:* An expression that resolves to an absolute or relative value of time.
Timing mark, *aikaleima:* A denotation for the time at which an event or message occurs. Timing marks may be used in constraints.
Trace, *jäljitys:* A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.
Transient object, *väliaikainen olio:* An object that exists only during the execution of the process or thread that created it.
Transition, *(tila)siirtymä:* A relationship between two states indicating that an object in the first state will perform certain specified actions and enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to fire.
Type, *tyyppi, olion rakenteen määrittelemä malli:* A stereotype of class that is used to specify a domain of instances (objects) together with the operations applicable to the objects. A type may not contain any methods. See: *class, instance*. Contrast: *interface*.
Type expression, *tyyppi-lauseke:* An expression that evaluates to a reference to one or more types.
Uninterpreted, *tulkitsematon:* A placeholder for a type or types whose implementation is not specified by the UML. Every uninterpreted value has a corresponding string representation.
Usage, *käyttötarve*: A dependency in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation.
Use case [class], *käyttötapaus (luokka):* The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system. See: *use case instances.*

Use case diagram, *käyttötapauskaavio:* A diagram that shows the relationships among actors and use cases within a system.
Use case instance, *käyttötapauksen ilmentymä:* The performance of a sequence of actions being specified in a use case. An instance of a use case. See: *use case class*.
Use case model, *käyttötapausmalli:* A model that describes a system's functional requirements in terms of use cases.
Utility, *apuohjelma:* A stereotype that groups global variables and procedures in the form of a class declaration. The utility attributes and operations become global variables and global procedures, respectively. A utility is not a fundamental modelling construct, but a programming convenience.
Value, *arvo:* An element of a type domain.
Vertex, *tilasiirtymän alku- tai lopputila:* A source or a target for a transition in a state machine. A vertex can be either a state or a pseudo-state. See: *state, pseudo-state*.
View, *näkymä:* A projection of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective.
View element, *näkymän alkio:* A view element is a textual and/or graphical projection of a collection of model elements.
View projection, *näkymän projektio:* A projection of model elements onto view elements. A view projection provides a location and a style for each view element.
Visibility, *näkyvyys (julkinen, suojattu tai yksityinen tieto/palvelu):* An enumeration whose value (public, protected, or private) denotes how the model element to which it refers may be seen outside its enclosing namespace.

LÄHTEITÄ

Booch, Jacobson, Rumbaugh. 1999. *The Unifield Modeling Language User Guide.* Addison-Wesley Longman Inc. ISBN 0-201-57168-4.

Eriksson, Hans-Erik & Penker, Magnus. 2000. *UML*. Suomentaja Tarmo Toikkanen. Gummerus Kirjapaino Oy. Jyväskylä. ISBN 951-826-026-5. (Alkuperäinen teos UML Toolkit)

Fowler, Martin & Scott, Kendal. 2002. *UML*. Suomentaja Eero Sarkkinen. Docendo Finland Oy. Jyväskylä. ISBN 951-846-168-6. (Alkuperäinen teos UML Distilled)

Haikala, Ilkka ja Märijärvi, Jukka. 2002. *Ohjelmistotuotanto*. 8. painos. Talentum Media Oy. ISBN 952-14-0486-8.

Koskimies, Kai. 2000. *Oliokirja*. 2. painos. Gummerus Kirjapaino Oy. Jyväskylä. ISBN 951-762-720-3.

Object Management Group (OMG): http://www.omg.org/technology/uml/index.htm

Oliopohjainen tietojärjestelmän suunnittelu:
http://hercules.pspt.fi/homepages/jpulkkin/UML/index.html

Oliosanasto: http://www.cs.helsinki.fi/u/laine/oliosanasto/

LIITTEET

Liite 1: Sanalliset kuvaukset työnjohtojärjestelmän käyttötapauskuvauksista

Käyttötapaus: Kirjaa toimeksianto
Ehdot: Asiakas on antanut yritykselle toimeksiannon. Toimeksianto sisältää seuraavat tiedot: asiakkaan nimi, osoite, puhelinnumero, kohteen/kohteiden osoite ja suoritettavat tehtävät.
Aktori: Työnjohto
Kuvaus: Työnjohtaja avaa työnjohtojärjestelmän ja siirtyy toimeksiannon kirjaamiseen. Tiedot syötettyään hän tallentaa tiedot Tallenna-painikkeella. Peruuta-painike tyhjentää tiedot tallentamatta.
Poikkeukset: Työnjohtojärjestelmä ei avaudu.
Jälkitilanne: Toimeksianto jää odottamaan tehtävien lisäämistä.

Käyttötapaus: Lisää tehtävä
Ehdot: Työnjohto tai sihteeri on kirjannut toimeksiannon määrämuotoisena järjestelmään.
Aktori: Työjohto.
Kuvaus: Toimeksiannon ollessa valittuna työnjohto painaa Lisää tehtävä –painiketta. Näkyviin tulee ikkuna Tehtävät. Järjestelmä pyytää täyttämään seuraavat tiedot: suoritettava tehtävä, osoite, tarvikkeet. Käyttäjä tallentaa painikkeella OK tai peruuttaa painikkeella Peruuta. Tallennus on mahdollinen vaikka kaikkia tietoja ei olisikaan kirjattu, mutta lisäys aikatauluihin tapahtuu vasta kun tiedot asentajasta ja ajankohdasta on lisätty.
Poikkeukset: Järjestelmä näyttää virheilmoituksen, jos ajankohta on aikaisempi kuin nykyinen päivämäärä.
Jälkitilanne: Toimeksianto on pilkottu järjestelmässä yhdeksi tai useammaksi tehtäväksi.

Käyttötapaus: Kiinnitä toimeksianto vapaalle asentajalle
Ehdot: Järjestelmään on syötetty vähintään yksi asentaja.
Aktori: Työnjohto
Kuvaus: Työnjohto siirtyy Asentajat-lomakkeeseen ja painaa Haku-painiketta. Haku-laatikossa on pudotusvalikko Työntekijän taidot. Tästä valitaan tehtävän edellyttämät taidot ja valitaan Etsi-painike, jolloin näkyviin tulevat valittujen kriteerien täyttämät asentajat nimettyinä taulukkomuodossa. Kun taulukosta valitaan asentaja, tulevat aikataulutiedot näkyviin aikataulutiedot-taulukkoon. Kun asentaja on valittu, lisätään aikataulutiedot-taulukkoon uusi tehtävä ja sen ajankohta.
Poikkeukset: Työtehtävään soveltuvaa vapaata asentajaa ei löydy hakuehdoilla.
Jälkitilanne: Työnjohto on kiinnittänyt toimeksiannon asentajalle.

Käyttötapaus: Viimeistele laskut
Ehdot: Työntekijä on lähettänyt laskutustiedot tehtäväkohtaisesti eriteltynä palvelimelle
Aktori: Työnjohtaja
Kuvaus: Työnjohtaja avaa työasemaltaan palvelimella sijaitsevan laskutustiedot sisältävän kansion, kerää sieltä tarvitsemansa tiedot ja sulkee kansion. Työnjohtaja avaa laskutusohjelman ja täyttää siihen palvelimelta saamansa tiedot. Tämän jälkeen hän lähettää laskutustiedot tulostettaviksi.
Poikkeukset: Laskuja ei voida viimeistellä, mikäli palvelimelle ei ole tallennettu tarvittavia laskutustietoja.
Jälkitilanne: Laskut ovat valmiita tulostettavaksi ja lähetettäväksi asiakkaalle.

Käyttötapaus: Laadi raportit
Ehdot: Työntekijät ovat lähettäneet raporteissa esiintyvät tiedot palvelimelle
Aktori: Työnjohtaja
Kuvaus: Työnjohtaja avaa työasemaltaan ohjelman, josta haluaa raportin. Hän valitsee raporttiin tarvittavat tiedot: ajankohdan, suoritetut tehtävät, kappalemäärän, laskutussummat ja työtunnit. Raportin hän tallentaa tiedostoon ja sulkee ohjelman.
Poikkeukset: Raportteja ei voida muodostaa, elleivät työntekijät ole toimittaneet tarvittavia tietoja tietokantaan.
Jälkitilanne: Raportit ovat valmiita tulostettaviksi ja toimitettaviksi eteenpäin sähköpostilla, faksilla tai postitse.

Käyttötapaus: Selvitä työntekijän tämänhetkinen toimeksianto
Ehdot: Järjestelmään on syötetty vähintään yksi asentaja
Aktori: Työnjohto
Kuvaus: Työnjohto siirtyy Asentajat-lomakkeeseen ja painaa Haku-painiketta. Haku-laatikon auettua hän syöttää nimi-kenttään asentajan sukunimen tai sen alkukirjaimet. Järjestelmä näyttää kriteerit täyttävien asentajien aikataulutiedot taulukossa. Työnantaja voi myös selata taulukossa asentajan kohdalle. Kun asentaja on valittu, täyttyy toimeksiannot-taulukko asentajan toimeksiannoista.
Poikkeukset: Asentajalle ei ole yhtään toimeksiantoa. Kriteerit täyttävää asentajaa ei löydy.
Jälkitilanne: Työnjohto on saanut näkyviin haluamansa tiedot.

Käyttötapaus: Hae lähin vapaa asentaja
Ehdot: Järjestelmään on syötetty vähintään yksi asentaja. Työnjohto saa kiireellisen hälytyksen tai puhelun.
Aktori: Työnjohto
Kuvaus: Työnjohto siirtyy Asentajat-lomakkeeseen ja painaa Haku–painiketta. Haku-laatikon auettua työnjohto syöttää päivämäärän ja kellonajan. Järjestelmä näyttää tämän jälkeen kriteerit täyttävät vapaat asentajat taulukossa. Työnjohto selaa taulukossa asentajien sijainnin ja valitsee lähinnä olevan asentajan.
Poikkeukset: Ei yhtään vapaata asentajaa.
Jälkitilanne: Työnjohto on saanut näkyviin haluamansa asentajan.

Käyttötapaus: Päivitä aikataulu
Ehdot: Asentaja on suorittanut toimeksiannon loppuun.
Aktori: Asentaja
Kuvaus: Asentaja siirtyy Asentajat-lomakkeeseen ja painaa Haku–painiketta. Haku–laatikon auettua hän syöttää nimi -kenttään oman sukunimensä tai sen alkukirjaimet. Järjestelmä näyttää asentajan rivin taulukossa. asentaja valitsee rivin ja päivittää suoritetut toimeksiannot kestoineen ja mahdollisine viivästyksineen.
Poikkeukset: Viivästymisen johdosta uusi toimeksianto on jo päällä järjestelmässä.
Jälkitilanne: Asentaja on päivittänyt aikataulun.

Käyttötapaus: Lähetä laskutustiedot
Ehdot: Asentaja on suorittanut laskutettavan toimeksiannon loppuun.
Aktori: Asentaja
Kuvaus: Asentaja avaa työasemaltaan palvelimella sijaitsevan laskutustiedot sisältävän kansion, lisää sinne kannettavansa kautta tehtäväkohtaisesti eriteltyinä laskutustiedot (työtunnit, tarvikkeet).
Poikkeukset: Usea asentaja on toteuttanut toimeksiannon.
Jälkitilanne: Työntekijä on lähettänyt laskutustiedot tehtäväkohtaisesti eriteltynä palvelimelle.

Liite 2: *Tietohakemisto suunnitteluvaiheen luokkakaaviosta*

A. Luokkia:

Asiakas on henkilö, joka tarvitsee yrityksen palveluja.
- Etunimi: Asiakkaan etunimi
- Sukunimi: Asiakkaan sukunimi
- Osoite: Asiakkaan osoite
- Puhelin: Asiakkaan puhelinnumero.

Työnjohto jakaa toimeksiannot työntekijöille.
- Etunimi: Työnjohtajan etunimi
- Sukunimi: Työnjohtajan sukunimi

Toimeksianto on työtehtävä, jonka asiakas antaa yritykselle, työnjohtaja kirjaa ylös ja jonka työnjohtaja antaa asentajalle suoritettavaksi.
- Osoite: Osoite, jossa tehtävä suoritetaan
- Kuvaus: Tarkka kuvaus suoritettavasta tehtävästä

Tehtävä on asiakkaalle suoritettava työtehtävä.
- Alkamisajankohta: Päivämäärä, jolloin tehtävän suorittaminen aloitetaan
- Päättymisajankohta: Päivämäärä, jolloin tehtävä on tullut valmiiksi ja sen suorittaminen lopetetaan
- Kuvaus: Tarkka kuvaus suoritettavasta tehtävästä

Asentaja on yrityksen työntekijä, joka suorittaa annetut työtehtävät.
- Etunimi: Asentajan etunimi
- Sukunimi: Asentajan sukunimi
- Asiantuntemus: Tieto asentajan ammattitaidosta eli tehtävät, johon hänellä on valmiudet.

Laskutustieto on joukko tietoja, joiden perusteella voidaan laatia lasku asiakkaalle suoritetusta tehtävästä.
- Työtunnit: Asentajien työtehtävään käyttämä kokonaistuntimäärä
- Tehtävä: Tehtävä, johon laskutustieto liittyy

Lasku on asiakkaalle toimitettava maksukehote suoritetusta tehtävästä.
- Viitenumero: Laskun yksilöintinumero
- Tuntihinta: Yhden työtunnin hinta
- Materiaalihinta: Hinta, joka veloitetaan käytetyistä materiaaleista
- Loppusumma: Asiakkaan maksettavaksi tuleva laskun summa
- Pvm: Laskun lähettämispäivämäärä
- Tilinumero: Yrityksen tilinumero, johon lasku maksetaan

Seurantaraportti on suoritettujen tehtävien perusteella laadittu raportti, jossa on yhteenveto halutuista tiedoista kuukauden ajalta.
- Kuukausittaiset tehtävät: Ko. kuukauden aikana tehdyt tehtävät
- Työtunnit: Ko. kuukauden aikana tehdyt työtunnit

- Laskutussumma: Ko. kuukauden aikana lähetettyjen laskujen yhteissumma

B. Assosiaatioita:

Antaa: Asiakas antaa työnjohdolle toimeksiannon työtehtävästä. Toimeksiantoja voi olla useita ja useat asiakkaat voivat antaa toimeksiantoja.

Kirjaa: Työnjohto kirjaa asiakkaan antaman toimeksiannon järjestelmään. Yksi työnjohtaja voi kirjata useita toimeksiantoja ja useat työnjohtajat voivat kirjata useita toimeksiantoja.

Annetaan: Työtehtävän anto asentajalle. Yksi työtehtävä annetaan yhdelle asentajalle.

Lähettää: Asentaja lähettää työtehtävän tiedot laskutustietoihin. Yksi laskutustieto liittyy yhteen tehtävään.

Liittyy: Laskutustieto liittyy tehtävään. Yksi laskutustieto voi liittyä yhteen tehtävään.

Siirtää: Laskutustiedot siirretään työnjohtoon jatkokäsittelyyn. Yksi tieto tulee yhdelle työnjohtajalle.

Viimeistelee: Työnjohto viimeistelee lähetettävät laskut asiakkaille lähetettävään kuntoon. Yksi työnjohtaja voi viimeistellä usean laskun ja saman laskun voi viimeistellä usea työnjohtaja.

Laaditaan: Laskujen perusteella laaditaan kuukausittaiset seurantaraportit. Useasta laskusta laaditaan yksi raportti ja yhtä laskua voidaan käyttää usean raportin laatimiseen.

Liite 3: *Luokkamääritykset suunnitteluvaiheen luokkakaaviosta*

A Luokan nimi: Toimeksianto

Kuvaus:
Kuvaus tarkoittaa työtehtävää, jonka asiakas antaa yritykselle. Työnjohtaja kirjaa toimeksiannon ylös ja antaa asentajalle suoritettavaksi.

Attribuutit:
Osoite on paikka, jossa tehtävä suoritetaan.
Kuvaus tehdään suoritettavasta tehtävästä tarkasti.
Tehtävä_REF on viiteattribuutti tehtävään.

Operaatiot:
TallennaTiedot tallentaa asiakkaan antaman toimeksiannon tiedot. Toimeksiannon kohteen osoitetiedot sekä toimeksiannon kuvaus saadaan käyttäjän täyttämästä tekstikentästä. Palauttaa tiedon onnistuneesta tallennuksesta.

B Luokan nimi: Ohjain

Kuvaus:
Ohjain on ohjausluokka, jonka tehtävänä on välittää käyttäjän syöttämät tiedot käyttöliittymältä liiketoimintaluokille (Toimeksianto, Tehtävä, Asentaja).

Attribuutit:
Toimeksianto_REF on viiteattribuutti toimeksiantoon
Asentaja__REF on viiteattribuutti asentajaan
Tehtävä_REF on viiteattribuutti tehtävään

Operaatiot:
VälitäTiedot
Ohjain-olio suorittaa seuraavia tehtäviä:

1) Olio välittää työnjohdon syöttämät hakuehdot asentajista Asentaja-luokalle, joka hakee tehtävään sopivat asentajat.
2) Olio välittää työnjohdon käyttöliittymään syöttämät tiedot toimeksiannosta Toimeksianto-luokalle, joka huolehtii tietojen tallentamisesta.
3) Olio välittää työnjohdon käyttöliittymään syöttämät tiedot toimeksiantoon kuuluvasta tehtävästä Tehtävä-luokalle, joka huolehtii tietojen tallentamisesta.
4) Olio ilmoittaa Asentaja-luokalle käyttöliittymässä valitun asentajan.
5) Olio välittää Tehtävä-luokalle tiedon valitun tehtävän ajankohdasta ja Asentaja-luokalle valittuun tehtävään kiinnitetystä asentajasta.

C Luokan nimi: Tehtävä

Kuvaus:
Kuvaus on työnjohdon toimeksiannosta erittelemä tehtävä.

Attribuutit:
Alkamisajankohta on päivämäärä, jolloin tehtävän suorittaminen aloitetaan.
Päättymisajankohta on päivämäärä, jolloin tehtävän suorittaminen lopetetaan.
Kuvaus on kuvaus siitä mitä tehtävässä tehdään.

Operaatiot:
LuoTehtävä tallentaa työnjohdon toimeksiannosta erittelemän tehtävän ja palauttaa tiedon onnistuneesta tallennuksesta.
KiinnitäAjankohta tallentaa valitun tehtävän ajankohdan, joka saadaan käyttöliittymältä ohjaimen välityksellä.
HaeTehtäväni hakee Asentaja-luokan pyytämän tehtävän ja palauttaa sen Ohjain-luokalle.