Harri Laine

Johdatus sovellussuunnitteluun

Osa 3

Helsingin yliopisto

Tietojenkäsittelytieteen laitos

2002

Sisältö:

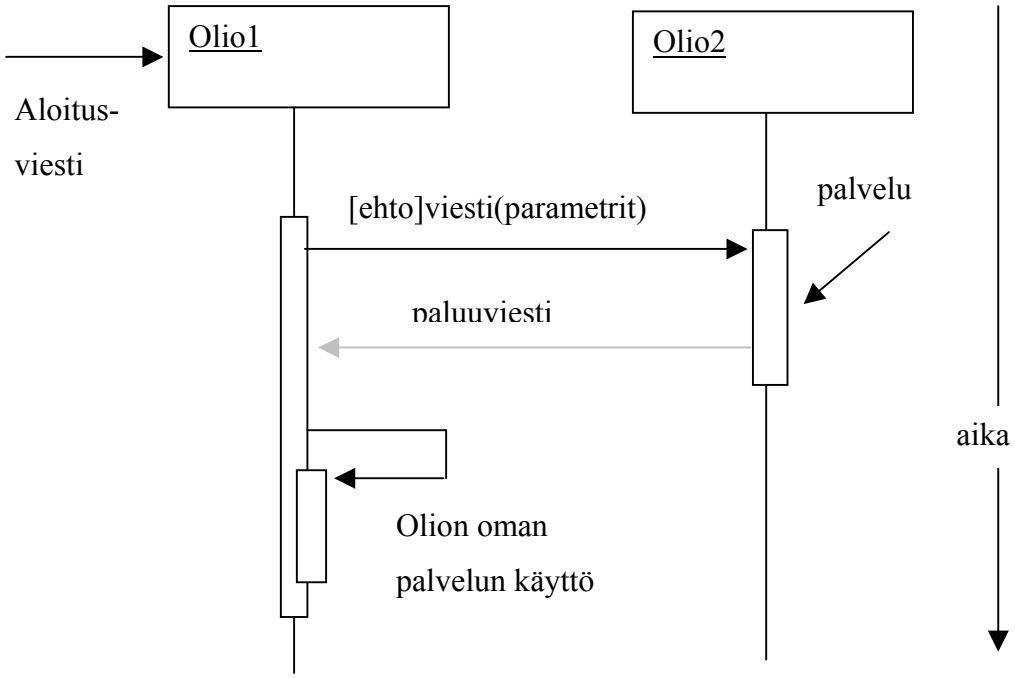
5. Olioiden yhteistyö ja palveluiden määrittely

Oliojärjestelmän toiminta perustuu olioiden yhteistyöhön. Yhteistyö ei tule esiin luokka- tai oliokaavioissa. Luokkakaaviossa esitetään luokan palvelut, mutta siitä ei näy miten palvelun suoritus hyödyntää muita palveluita tai muita olioita. Yhteistyön selvittäminen on kiinteästi sidoksissa olioiden palveluiden määrittelyyn, sillä yhteistyö toteutuu palvelujen kautta. Samalla kun määrittelemme oliolle palveluja meidän tulisi ottaa kantaa siihen millaista yhteistyötä olioiden välillä palvelun suorittamiseen liittyy. Palvelujen ja olioyhteistyön määrittely on ohjelmiston teknisen suunnittelun tehtäviä. Nämä tehdään siis myöhemmin kuin sovellusalueen kannalta keskeisten luokkien määrittely, joka tapahtuu jo ohjelmiston määrittelyvaiheessa.

UML tarjoaa kaksi tekniikkaa olioyhteistyön kuvaamiseen: sekvenssikaavion ja yhteistyörakennekaavion. Sekvenssikaaviossa (yhteistyöpoluissa, sequence diagram) keskitytään kuvaamaan operaatioiden suoritusjärjestystä ja kontrollin etenemistä oliolta toiselle. Yhteistyörakennekaaviossa (collaboration diagram) kuvataan erityisesti sitä, miten yhteistyö perustuu olioiden välisiin yhteyksiin. Suoritusjärjestys ja kontrollin eteneminen kuvataan myös, mutta ei yhtä helppolukuisesti kuin sekvenssikaavioissa.

5.1 Sekvenssikaavio

Sekvenssikaavio kuvaa tietyn palvelun suorittamiseen liittyvän olioiden yhteistyön. Kuvattava palvelu voi olla järjestelmän palvelu (käyttötapaus) tai johonkin luokkaan liittyvä palvelu. Sekvenssikaaviossa esitetään, mitä avustavia olioita palvelun suoritukseen osallistuu ja mitä näiden palveluita käytetään. Kaavio kuvaa myös missä järjestyksessä avustavien olioiden palveluja käytetään. Kuva 5.1 esittää kaaviotekniikassa käytettävät symbolit.



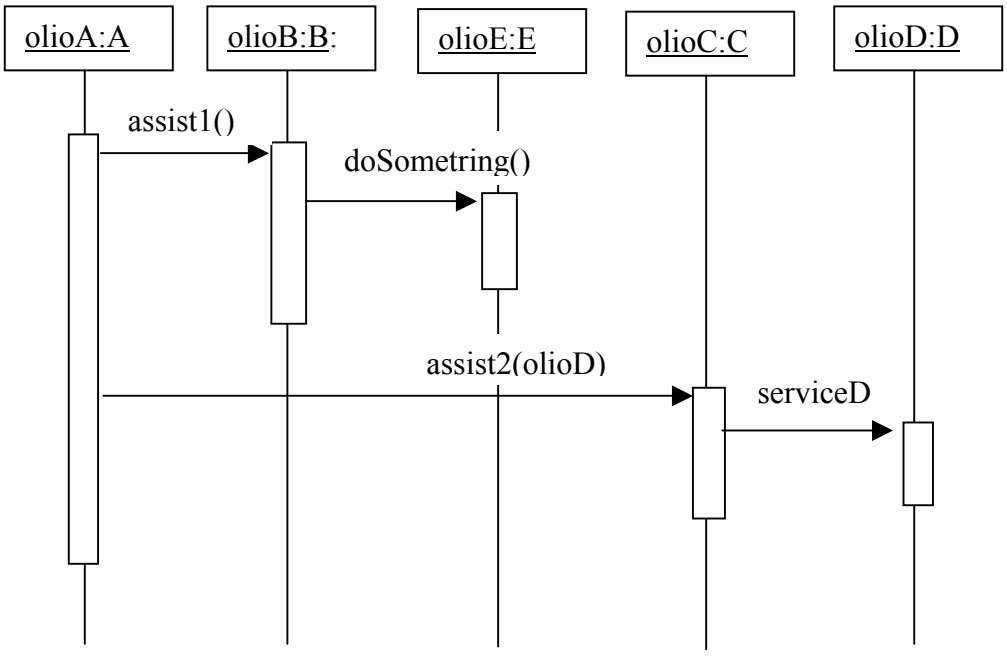
Kuva 5.1: Sekvenssikaavion symbolit

Sekvenssikaaviossa oliot kuvataan suorakaiteina, jonka sisällä on olion tunnus. Suorakaiteesta alaspäin lähtevä viiva kuvaa olion elinkaarta. Viivalla oleva paksunnos (laatikko) kuvaa palvelun suoritusta. Paksunnoksen pituus kuvaa palvelun kestoa lähinnä kontrollin kannalta. Oliolla on kontrolli suoritukseen palvelun keston ajan. Olio pyytää avustavalta oliolta palvelua lähettämällä tälle viestin. Olio-ohjelmassa viestin lähettäminen tarkoittaa yleensä metodikutsua (esim. Javassa). Kaaviossa viestin lähettäminen kuvataan lähettäjältä vastaanottajalle menevänä nuolena. Nuoleen liitetään tekstinä lähetettävä viesti. Yleensä tämä muodostuu pyydettävän palvelun nimestä ja pyynnön yhteydessä välitettävistä parametreista. Ohjelmissa metodit voivat tuottaa paluuarvon. Tällaisia ei yleensä piirretä kaavioon. Palauteviesti voidaan piirtää kaavioon, mutta sitä käytetään lähinnä poikkeuksellisissa tilanteissa, esimerkiksi silloin kun rinnakkain toimiva palvelu palauttaa jo ennen päättymistään jotain tietoa palvelun pyytäjälle. Viestiin voidaan liittää myös ehto, jonka voimassaollessa viesti voidaan lähettää. Ehdon liittäminen ei ole välttämätöntä.

Esimerkki: Tarkastellaan seuraava Java-ohjelman osaa

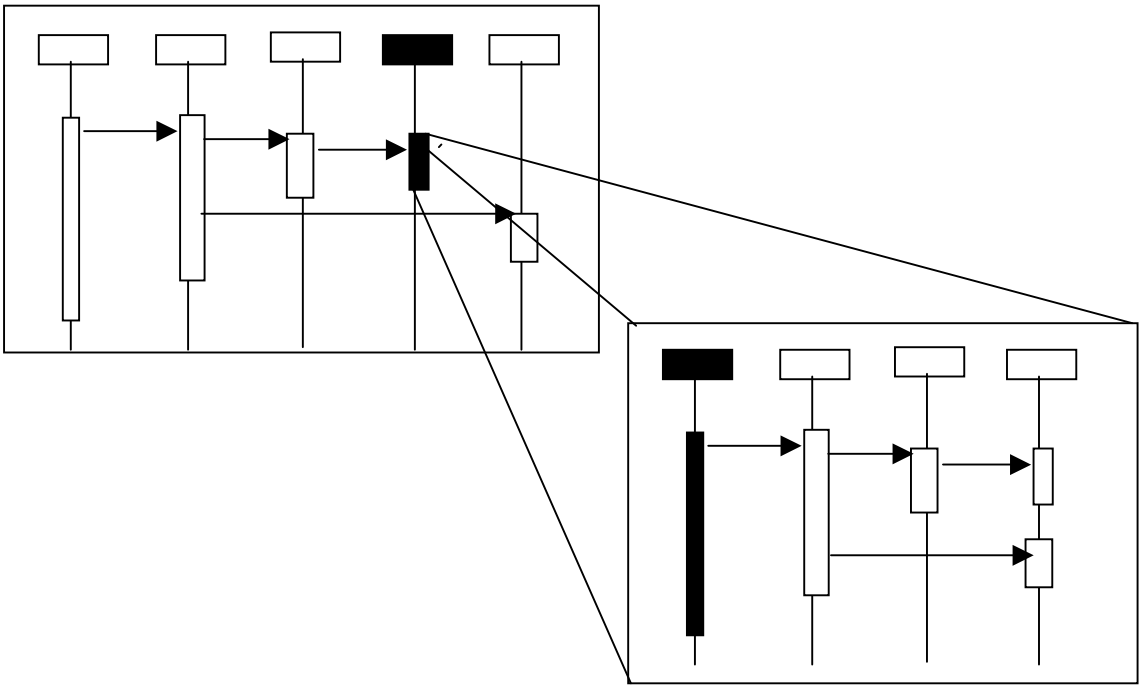
```
class A {
  B olioB;
  C olioC;
  D olioD;

  public int doIt() {
     olioB.assist1();
     olioC.assist2(olioD);
  }
}

class B {
   E olioE;

   public void assist1() {
     olioE.doSomething();
   }
}

class C {

   public void assist2(D od) {
      od.serviceD()
   }
}
```

Palveluun A.doIt liittyvä yhteistyökaavio on esitetty kuvassa 4.2.



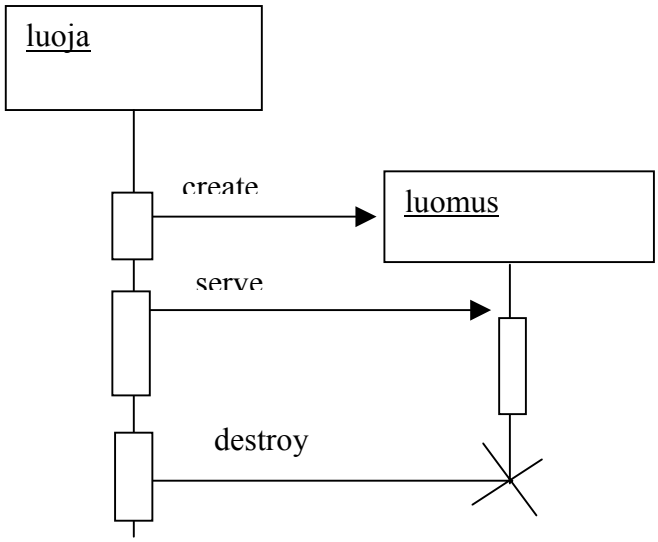
Kuva 5.2: Esimerkkiohjelman palvelun A:doIt sekvenssikaavio.

Sekvenssikaavion tarkoituksena on havainnollistaa olioiden välistä yhteistyötä. Etenkin järjestelmätason palveluita tarkasteltaessa saattaa palvelun suoritukseen osallistua kymmeniä olioita ja yhteistyöpolut muodostuvat pitkiksi. Yhdessä kaaviossa ei ole kuitenkaan tarkoituksenmukaista esittää kovin pitkiä yhteistyöpolkuja. 2-3 palvelupyynnön ketju on sopivan pituinen. Jatko voidaan esittää erillisessä kaaviossa, jos se on tarpeen (kuva 5.3).

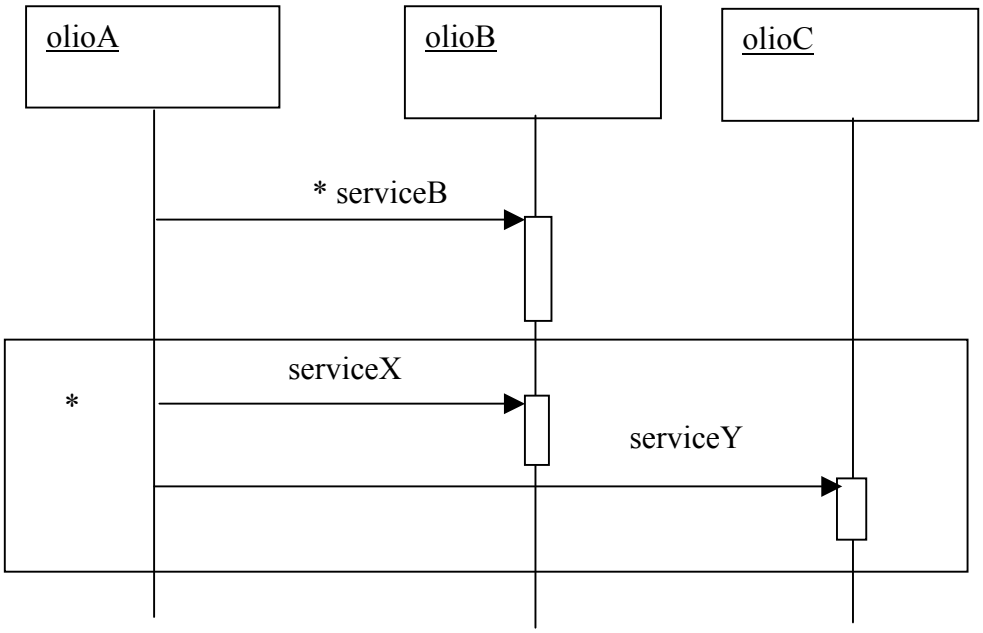


Kuva 5.3: Pitkän yhteistyöketjun ositus eri kaavioihin

Sekvenssikaaviossa voidaan esittää myös olioiden luonti ja tuhoaminen (kuva 5.4). Ohjelmissa esiintyy usein silmukoita, joissa palvelua pyydetään esimerkiksi kaikilta kokoelman jäseniltä. Silmukoiden esittämistä sekvenssikaavioissa havainnollistetaan kuvassa 5.5.



Kuva 5.4: Olion luonti ja tuhoaminen.



Kuva 5.5: Silmukoiden esittäminen

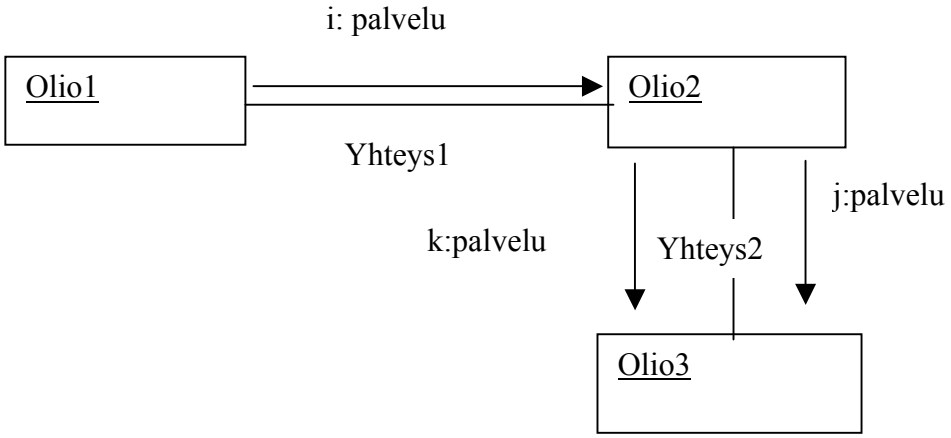
Kuvan 5.5 silmukat voisivat olla muotoa:

```
foreach olioB in kokoelma1 do {
   olioB.serviceB()
}
foreach (olioB, olioC) in kokoelma2 do {
   olioB.serviceX;
   olioC.serviceY()
}
```

Sekvenssikaavioita ei yleensä ole tarpeen laatia kaikille palveluille. Yhteistyön ymmärtämisen kannalta keskeiset palvelut on syytä kuvata. Samantapaisista yhteistyökuvioista riittää yleensä yksi esimerkki. Hyvin yksinkertaisia palveluita ei ole tarpeen kuvata kaaviolla.

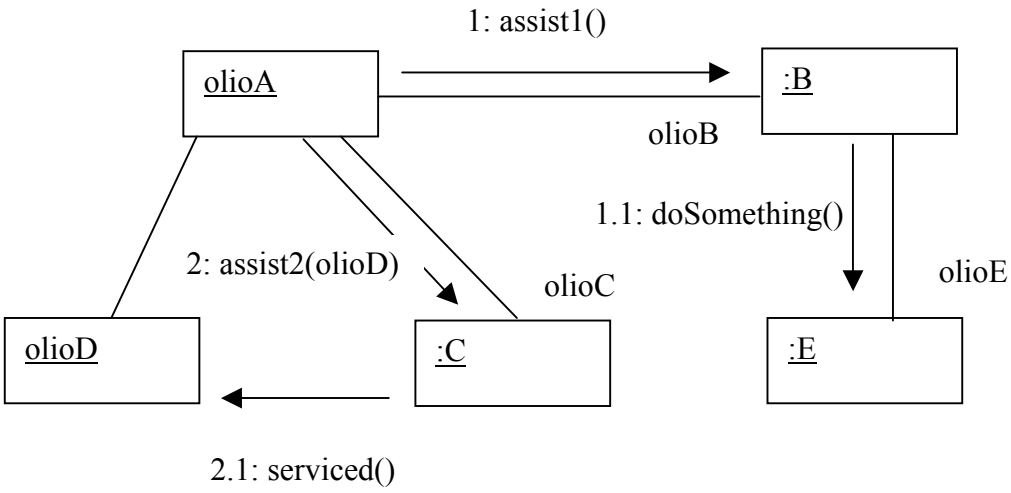
5.1 Yhteistyörakennekaavio

Yhteistyörakennekaavioissa esitetään miten yhteistyö käyttää hyväkseen olioiden välisiä yhteyksiä. Yhteistyö perustuu olioiden välisiin yhteyksiin, sillä olio voi pyytää palvelua vain sellaiselta oliolta, jonka olemassaolon se tietää. Tämä tietäminen kuvataan luokkakaaviossa yhteyksien avulla. Yhteistyörakennekaaviossa käytettävät symbolit selviävät kuvasta 5.6.



Kuva 5.6: Yhteistyörakennekaavio

Yhteistyörakennekaaviossa palvelujen suoritusjärjestys kuvataan palvelun nimen eteen sijoitettavan järjestysnumeron avulla. Järjestysnumero voidaan antaa tasoittain. Jos oliolta A pyydetyn palvelun X järjestysnumero olisi *i*, niin ensimmäinen olion A palvelun X sisällä pyytämä palvelu olisi järjestysnumeroltaan *i.1* ja seuraava *i.2*. Kuvan 5.2 sekvenssikaaviossa esitetty yhteistyö kuvataan yhteistyörakenteena kuvan 5.7 mukaisesti.



Kuva 5.7: Yhteistyörakennekaavio

5.2 Palvelujen määrittelystä

Luokkien palveluja määriteltäessä voidaan luokille aluksi antaa attribuuttien kysymiseen ja niiden arvojen asetukseen liittyvät peruspalvelut (set- ja get –metodit). Monipuolisempien palvelujen kohdalla on otettava kantaa yhteistyöhön

- tarvitseeko joku muu olio suunniteltua palvelua
- millaisiin yhteistyöketjuihin palvelu osallistuu.

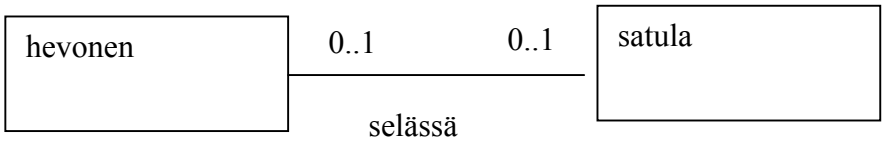
Yhteistyön määrittelyn luonnollinen aloituskohta ovat järjestelmän palvelut. Järjestelmän palvelujen toteutukseen osallistuvat

- käyttöliittymäoliot (ikkunat, valikot, napit, jne), joiden tehtävänä on hoitaa yhteys käyttäjän ja sisältöolioiden välillä

- sisältöoliot, jotka toteuttavat ’reaalimaailman simulointimallin’
- tekniset apuoliot (tietokantarakenteet, tietoliikenneoliot, jne)

Sisältöolioiden ja käyttöliittymäolioiden väliseen kytkentään on esitetty monenlaisia ratkaisumalleja. Yleisperiaatteena useissa malleissa on käyttöliittymän ja sisällön erottaminen. Tämä tarkoittaa sitä, että sisältöolioiden pitäisi tarjota käyttöliittymästä riippumattomia palveluja sisällön käsittelyyn. Näitä palveluja aktivoidaan käyttöliittymän kautta. Tällainen ratkaisu tekee mahdolliseksi toteuttaa helposti erilaisia käyttöliittymiä samaan sisältöön perustuen. Ratkaisumallissa käyttöliittymän muuttaminen ei myöskään vaikuta sisältöluokkiin.

Oliot tarjoavat ensisijaisesti omaan tietosisältöönsä perustuvia palveluja. Yhteyksien kautta olioon kytkeytyvät yhteyskumppanit sisältyvät olion tietosisältöön. Olio voi käyttää näitä apuna omissa palveluissaan. Palvelujen liittäminen luokkiin ei ole aivan yksinkertainen asia. Suunnittelija joutuu usein miettimään, mille luokalle hän palvelun määrittelisi. Kirjallisuudessa on yleisesti käytetty esimerkkinä palvelun määrittelyn vaikeudesta hevosen satulointia. Oletetaan, että sekä hevonen että satula ovat luokkia ja satuloituna olo puolestaan hevosen ja satulan välinen yhteys (kuva 5.8).



Kuva: 5.8: Hevonen ja satula

Tarkastellaan hevosen satulointi tehtävää. Tarjolla on ainakin kolme mahdollisuutta liittää palvelu luokkiin:

- hevoselle määritellään palvelu *satuloidu*, jolle annetaan parametrina satula,
- satulalle määritellään palvelu *kiinnity*, jolla annetaan parametrina hevonen,
- otetaan käyttöön ylimääräinen luokka *ratsastaja* ja määritellään tälle palvelu *satuloi*, parametreina hevonen ja satula.

Reaalimaailmassa viimeisin vaihtoehto olisi luonnollinen tapa hahmottaa ilmiö, mutta olio-ohjelmissa joudutaan usein antamaan palveluja myös olioille, joiden reaalimaailmavastineet ovat sellaisiin kykenemättömiä. Niinpä esimerkin tapauksessa ei voida sanoa, mikä näistä ratkaisuista olisi yksiselitteisesti paras. Ratkaisu, mikä aluksi näyttää hyvältä, saattaa myöhemmin osoittautua huonoksi valinnaksi tuottamalla esimerkiksi runsaasti ohjelmakoodia. Huono valinta taas aiheuttaa tarpeen muuttaa suunnitelmaa.

Kun luokalle määritellään palvelu on selvitettävä pystyykö luokan olio hoitamaan sen itsenäisesti vai tarvitaanko avustavia olioita. Jos avustavia olioita tarvitaan, on suunniteltava miten yhteistyö toimii.

Esimerkki:

Tarkastellaan lipunvarausjärjestelmää ja sen palvelua '*Selvitä annettuna päivänä annetulla aikavälillä tarjolla olevat elokuvanäytökset*'. Seuraavassa tarkastellaan vain sisältöolioiden palveluita. *Ohjelmakartta*-luokan olioilla on tieto näytöksistä. Oletetaan, että yksi *Ohjelmakartta*-ilmentymä tietää kaikki näytökset. Se pystyy tämän tietonsa pohjalta toteuttamaan tehtävän. Määritellään *Ohjelmakartalle* tehtävää varten palvelu

näytöksetAikavälillä(päiväys, välin_alku, välin_loppu).

Palvelu tuottakoon tuloksenaan merkkijonotaulukon, jonka kukin alkio kuvaa yhden näytöksen. Koska *Ohjelmakartta*-olio tietää vain näytösten olemassaolon, se ei omin avuin kykene tuottamaan tulostaulukon alkioita. Sen on käytettävä näiden tuottamisessa apuna *Näytös*-olioita, joilla on enemmän tietoa näytöksistä.

Ennen *Näytös*-olioiden hyväksikäyttöä on kuitenkin tarkasteltava *Ohjelmakartta*-olion ja *Näytös*-olioiden välistä yhteyttä: *Onko ohjelmakartta-oliosta suora pääsy näytös-olioihin?*. Suora pääsy kaikkiin *Näytös*-olioihin on mahdollinen, jos kaikki *Näytös*-oliot ovat keskusmuistissa tai käytössä on oliotietokanta, joka toimii tavallaan keskusmuistin laajennoksen kaltaisesti. Tällöin *Näytös*-luokkaan voidaan liittää näytöksen kuulumisen oikealle aikavälille testaava palvelu tai palvelut esimerkiksi *esitetäänPäivänä(päiväys)* ja *alkaaVälillä(alku, loppu)*. Ellei suoraa pääsyä ole, vaan *Näytös*-olioiden

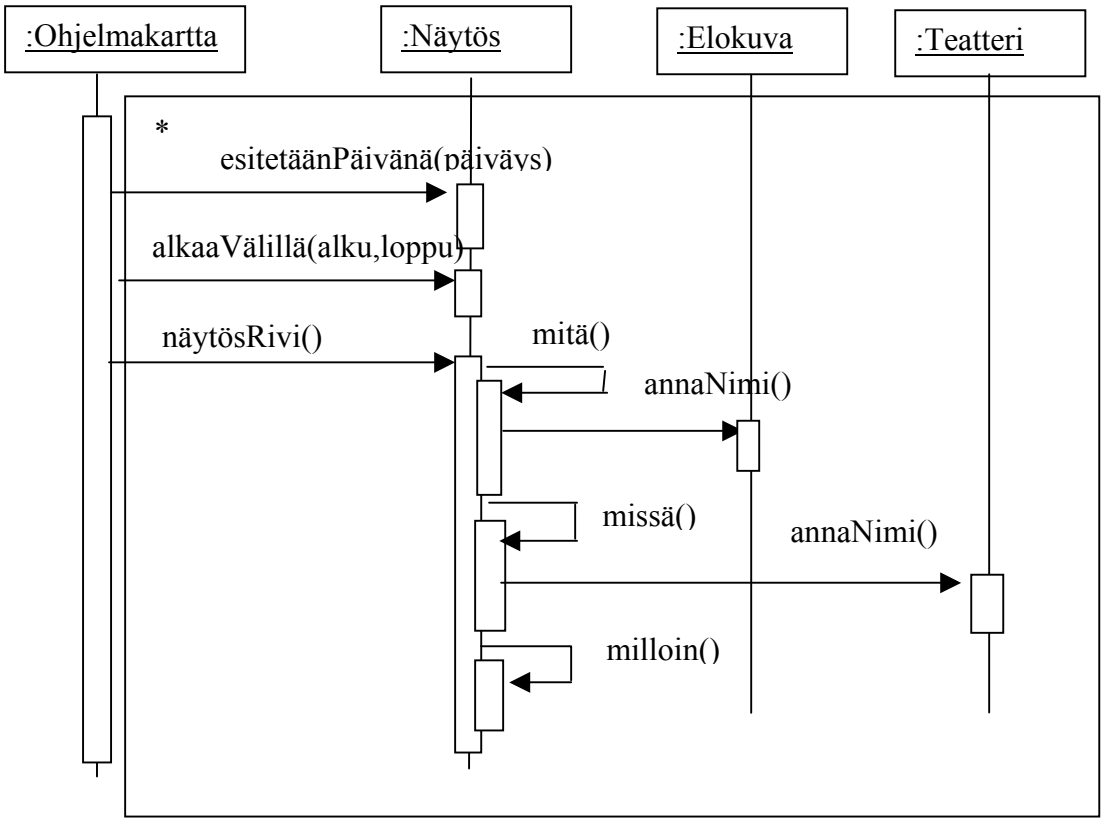
tiedot olisivat vaikkapa relaatiotietokannassa, tarvitaan apuoliona tietokanta-kysely, joka luo keskusmuistiin tarvittavat *Näytös*-oliot. Tällainen voisi olla luokan *NäytösHaku* olio, jolla olisi palvelu *haeNäytökset(päiväys, välin_alku, välin_loppu)*. Kysely hakisi tiedot tietokannasta, loisi Näytös-oliot vastauksen perusteella ja palauttaisi kokoelman, joka sisältää suorat viitteet luotuihin olioihin.

Näytös-luokalle voidaan määritellä palvelu *näytösRivi*, joka tuottaa tulok-senaan näytöstä kuvaavan merkkijonon. Tämän muodostukseen *Näytös* voisi tarjota apupalveluja:

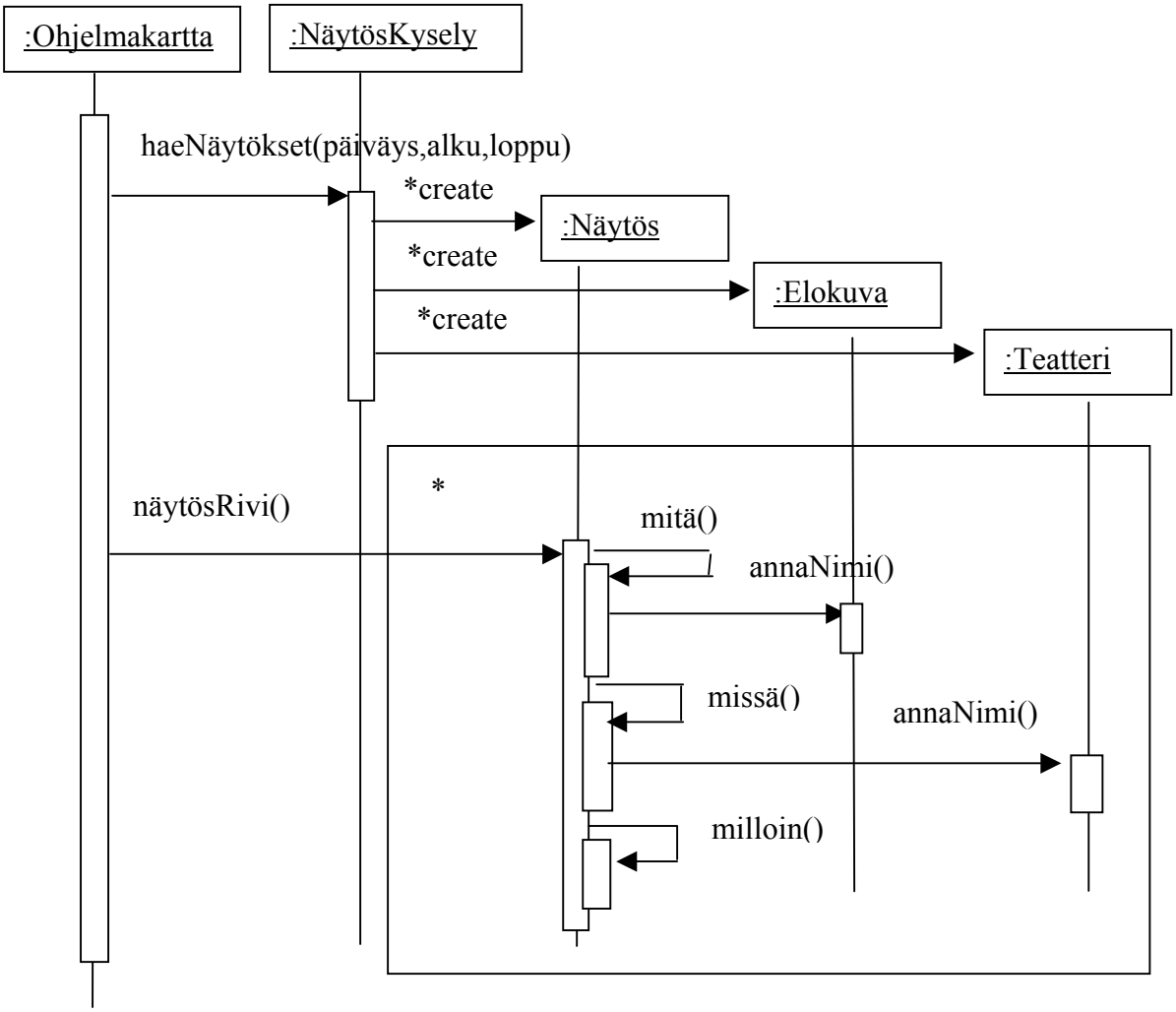
- *milloin*-palvelu antaisi päiväyksen ja kellonajan,
- *missä*-palvelu antaisi teatterin nimen ja
- *mikä*-palvelu antaisi elokuvan nimen.

Kahta viimeksimainittua *Näytös* ei kykene hoitamaan omin avuin, vaan jälleen tarvitaan yhteistyötä. Teatterin nimen selvittämiseksi pyydetään *Teatteri*-oliolta palvelua *annaNimi*. Elokuvan nimi voitaisiin saada *Elokuva*-olion *annaNimi*-palvelulla. Jos *Näytös*-oliosta ei ole suoraa pääsyä siihen kytkettyyn *Elokuva*- ja *Teatteri*-olioon, on pääsy mahdollistettava ennen palvelujen pyytämistä vaikkapa tietokantakyselyiden avulla. Tämä voidaan hoitaa vaikkapa määrittelemällä yllä mainittu *haeNäytökset-palvelu* toimimaan siten, että se luo myös *Elokuva*- ja *Teatteri*-oliot, mikäli niitä ei vielä ole keskusmuistissa.

Kuvassa 5.9 on kuvattu tarkastellun järjestelmäpalvelun toteutukseen liittyvä yhteistyö tilanteessa, jossa ohjelmakartasta on suora pääsy kaikkiin näytöksiin. Kuvassa 5.10 on kuvattu yhteistyö tilanteessa, jossa joudutaan käyttämään tietokantakyselyä luomaan tarvittavat Näytös-, Elokuva- ja Teatteri- oliot.



Kuva 5.9: Palveluun *Ohjelmakartta.näytöksetAikavälillä* liittyvä yhteistyö kun kaikkiin olioihin on suora pääsy.

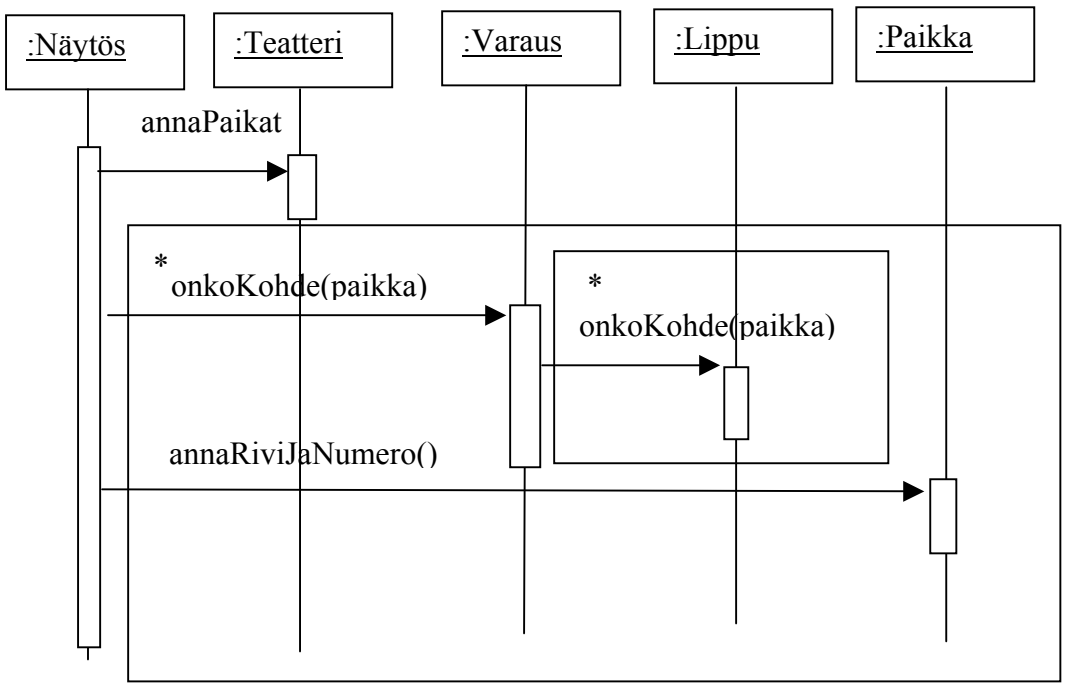


Kuva 5.10: Palveluun *Ohjelmakartta.näytöksetAikavälillä* liittyvä yhteistyö kun avustavat oliot täytyy luoda tietokantakyselyn avulla.

Esimerkki:

Tarkastellaan järjestelmäpalvelua '*Selvitä näytöksen vapaat paikat*'. Palvelu näyttäisi sopivan luokalle Näytös. Seuraavassa esitetään 2 vaihtoehtoista ratkaisua.

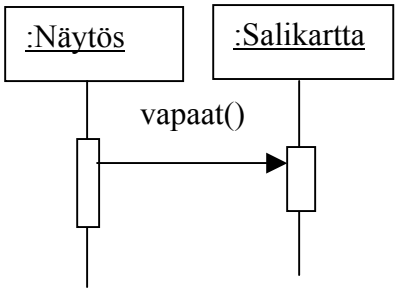
Vaihtoehto 1: Näytös pyytää teatteriltaan kaikki paikat ja käy ne sitten yksitellen läpi kysyen varauksiltaan kohdistuvatko ne kyseiseen paikkaan. Ratkaisu toimii, mutta vaikuttaa hitaalta (koska varaukset joudutaan lataamaan keskusmuistiin). Ratkaisuun perustuvan yhteistyökaavion hahmotelma on kuvassa 5.11. Kuvassa ei ole esitetty mahdollisesti tarvittavia tietokanta-kyselyjä.



Kuva 5.11: Olioiden yhteistyö ja palvelut selvitettäessä näytöksen vapaita paikkoja (vaihtoehto 1).

Vaihtoehto 2: Otetaan käyttöön apurakenne *Salikartta*. *Teatteri* osaa muodostaa salikartan, jossa kaikki paikat ovat vapaita. *Salikartalla* on palvelu *varaa(rivi,paikka)* ja *peru(rivi,paikka)*. *Salikartta*-olio liitetään *Näytös*-olioon sen luonnin yhteydessä ja jokainen varaus merkitään salikarttaan. *Salikartta*

osaa kertoa vapaat paikat vaikkapa metodilla *vapaat()*. Tämä tuottaa (rivi, paikka) –pareista muodostuvan taulukon. *Näytös* tuottaa tiedot vapaista paikoista kysymällä paikat salikartaltaan. Ratkaisu on nopea ja yhteistyöltään yksinkertainen, mutta vaatii jonkin verran muistitilaa. Vapaan paikan kysymiseen liittyvä yhteistyötä on hahmoteltu kuvassa 5.12. Tässä vaihtoehdossa paikan varaus ja varauksen peruutus muodostuvat olioiden yhteistyöratkaisultaan erilaisiksi kuin vaihtoehdossa 1.

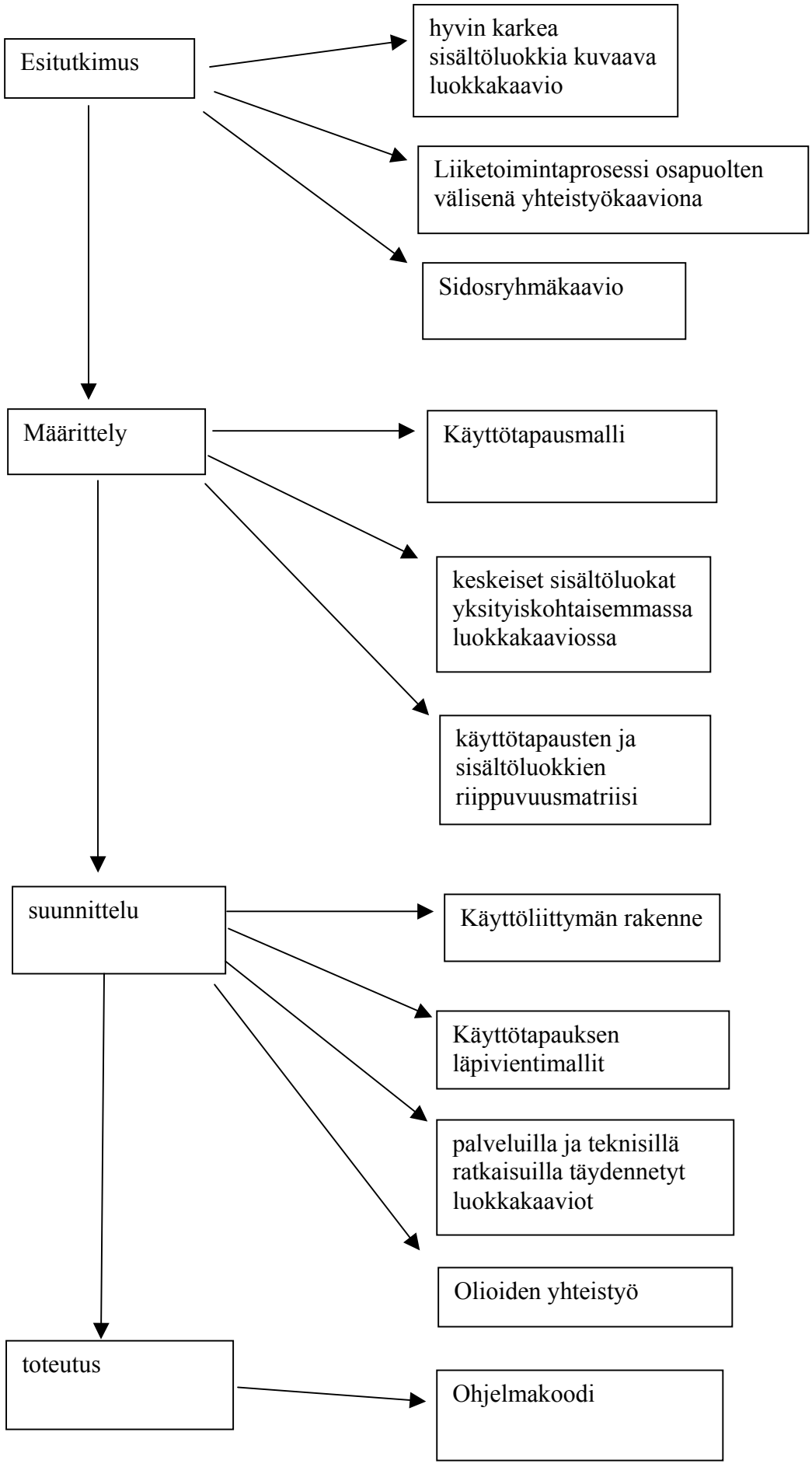


Kuva 5.12: Olioiden yhteistyö ja palvelut selvitettäessä näytöksen vapaita paikkoja (vaihtoehto 2).

Yllä olevissa esimerkeissä on palvelua ja yhteistyötä suunniteltaessa päädytty esittelemään uusia teknisen tason luokkia. Nämä eivät ole välttämättömiä ongelman ymmärtämisen kannalta, mutta ovat toki oleellisia teknisen ratkaisun kannalta siinä vaiheessa kun järjestelmän määrittelystä siirrytään tekniseen suunnitteluun.
Palveluiden määrittely saattaa myös aiheuttaa tarpeita muuttaa luokkiin liitettyjä attribuutteja ja luokkien välisiä yhteyksiä. Muutokset voivat hyvinkin olla sen tasoisia, että niiden tulisi näkyä myös tietosisältömallissa. Tästä syystä järjestelmän luokkakaaviokaan ei ole ’valmis’ ennenkuin palvelut ja yhteistyö on suunniteltu.

6. Kuvaukset ja kehittämisprosessi

Kuvauksia järjestelmästä tuotetaan sen kehityksen eri vaiheissa. Kuvassa 6.1 on kuvattu miten tällä kurssilla esitetyt kuvaukset suhtautuvat kehittämisprosessiin.



Kuva 6.1: Kuvaukset järjestelmäkehityksen vaiheissa

Esitutkimuksessa tuotettava karkea luokkakaavio pitää sisällään vain järjestelmän keskeiset luokat. Attribuutteja ei ole yleensä tarpeen määritellä vielä tässä vaiheessa. Keskeiset yhteydet otetaan mukaan. Ne voidaan kuitenkin yleensä jättää tässä vaiheessa nimeämättä eikä osallistumisrajoitteita määritellä.

Määrittelyvaiheessa tuotettavan käyttötapausmallin tulee olla kattava. Määrittelyvaiheen luokkakaavioon tulisi ottaa kaikki järjestelmän tietosisällön kannalta oleelliset luokat. Tietosisällön kannalta oleelliset attribuutit otetaan myös mukaan luokkakaavioon, samoin kuin yhteyksien osallistumisrajoitteet. Kaavion tarkkuuden tulisi olla sellainen, että sen perusteella pystytään toteamaan käyttötapausten toteutettavuus.

Suunnitteluvaihe jakautuu moniin alivaiheisiin, jotka tuottavat erilaisia kuvauksia. Järjestelmän tietosisältöä kuvaavan luokkakaavion perusteella tuotetaan esimerkiksi järjestelmän tietokannan rakenteen kuvaus. Tämän esittämiseen on oma kuvaustekniikkansa. Suunnitteluvaiheen tuloksena saatavat luokkakaaviot kuvaavat ohjelmiston teknistä rakennetta. Näissä kaavioissa ovat mukana myös luokkien tarjoamat palvelut. Tässä vaiheessa on tarpeen kuvata myös olioiden yhteistyö.
ja yhteistyö on suunniteltu.