UML OHJELMISTOPROSESSIEN TUKENA

Kimmo Kampman

11.5.2001

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Ohjelmistojen teko muuttuu jatkuvasti vaativammaksi. Ohjelmiston mallintamisen apuvälineeksi on luotu UML, joka on graafinen ohjelmiston mallinnuskieli. Sen ympärille on myös luotu sitä käyttäviä prosessimalleja. UML:n ja prosessimallien avulla saadaan ohjelmistokehityksessä parannettua uudelleenkäytettävyyttä, lyhennettyä ohjelmistojen kehitysaikaa ja nostettua tuottavuutta. UML on kuitenkin vielä sen verran uusi suunnittelun apuväline, ettei sen käyttö ole aivan ongelmatonta. Puutteistaan huolimatta se on tehokas apuväline mallinnettaessa ohjelmistoja.

Tämän tutkielman tarkoituksena on kuvata UML:n käyttöä prosessissa ja kahta prosessimallia, joissa UML:ää käytetään apuvälineenä. Nämä prosessimallit ovat Rational Unified Process ja GSMS:ään kuuluva Object and Component Engineering. Näiden prosessimallien vertailun lisäksi on arvioitu niiden ominaisuuksia, UML:n roolia prosesseissa ja sen mallintamisessa.

Tutkielman teoreettinen osuus perustuu lähdekirjallisuuteen. Prosessimallien ja UML:n arviointi perustuvat käytännön kokemuksiin sekä lähdekirjallisuudesta löytyvään materiaalin.

Avainsanat: UML, ohjelmistoprosessi, UP/RUP, GSMS/OCE

SISÄLLYSLUETTELO

1 JOHDANTO

Ohjelmistot ovat tärkeässä roolissa yhteiskunnassa ja niiden asema vahvistuu jatkuvasti. Ne ovat liiketoiminnan perusta. Ihmiset saavat tietoa helpommin ja nopeammin niiden ansiosta. Ohjelmistot ovat osaltaan auttaneet meitä luomaan, tarjoamaan ja esittämään tietoa monilla eri tavoilla. Ne ovat olleet osatekijöinä sairaiden parantamisessa, niiden avulla on voitu antaa ääni takaisin mykälle tai ne ovat auttaneet liikuntakyvytöntä liikkumaan. Ne ovat suurena osatekijänä siihen, että maailmantalous on kasvanut niihin mittoihin, missä se on nykyään (Kruchten, 2000). Voidaan sanoa, että melkein jokaiseen aikamme kehittyneeseen tuotteeseen tai palveluun kuuluu ohjelmistoja ja/tai se käyttää jotain ohjelmistoa (Fuggetta, 2000).

Vaikka ohjelmistot ovat olleet tärkeässä asemassa jo jonkin aikaa, korostuu niiden rooli entisestään. Jo nyt monessa tilanteessa ohjelmistovirhe voi aiheuttaa ihmishenkien menetyksiä. Koko ajan kasvavat ja monimutkaistuvat ohjelmat asettavat myös kasvavia vaatimuksia menetelmille, joilla niitä tehdään. Samalla tavalla kuin talojen rakentamista ei voisi kuvitellakaan tehtävän ilman rakennuspiirustuksia tai yksityiskohtaisia ohjeita työmenetelmistä, vaatii ohjelmistotuotteenkin tekeminen ohjeita ja piirustuksia. Juuri tätä varten on ohjelmistotuotantoon kehitetty prosesseja.

Ohjelmistotekniikka (software engineering) on tieteenalana uusi. Sen voidaan katsoa alkaneen varsinaisesti 50- ja 60-luvulla. Samaan aikaan alettiin kehittää erilaisia tiedonmallinnuskeinoja. Aluksi käytössä oli vuokaavioesityksiä, jotka muistuttivat paljon sähköteknisiä piirustuksia. Suurin osa tänäkin päivänä käytettävien menetelmien perustoista kehittyi 70- ja 80-luvun aikana. Tällöin syntyivät tiedon loogisia suhteita kuvaavat mallinnustavat, kuten oliosuhde-kaavio. Samoin 80-luvulla alettiin kiinnittää huomiota ohjelmistoprosessiin erilaisissa workshopeissa ja tapahtumissa (Fuggetta, 2000).

Myöhemmin 90-luvulla tuotettiin uusia kehittyneempiä menetelmiä, esimerkiksi olio-suunnittelumenetelmät. Uutena piirteenä näissä menetelmissä oli geneerisyys. Piirteitä alettiin ottaa aikaisemmista menetelmistä. Tällöin kehittyi myös UML (Unified Modeling Language), joka jo nimelläänkin kertoo siitä, että se koottiin ja yhdistettiin aikaisemmista menetelmistä.

Ohjelmistosuunnittelun kehittyminen jatkuu tämän vuosituhannen alkupuolella samoilla linjoilla kuin 90-luvun lopulla. Prosesseja pyritään kehittämään edelleen, mutta mitään mullistavaa ei näyttäisi olevan näköpiirissä. Parempaan laatuun ja luotettavuuteen pyritään

esimerkiksi komponenttitekniikoiden avulla. Uudelleenkäytettävyyden lisäämiseen kiinnitetään myös entistä enemmän huomiota.

Ohjelmiston kehitysprosessissa prosessi määrittää *kuka* tekee *mitä, milloin* ja *miten* saavuttaakseen tietyn tavoitteen (Jacobson & al., 1999). Prosessinmallin avulla pystytään koko ohjelmistoprojektin elinkaaren ajan hallitsemaan, mitä missäkin vaiheessa tehdään. Prosessimalli tarjoaa ohjeita jokaisen vaiheen suorittamiseen. Sen avulla myös kaikki ohjelmistoprosessiin kuuluvat ihmiset tietävät, missä vaiheessa ohjelmiston kehitys on.

Jacobsonin & al. (1999) mukaan ohjelmistoprosessiin kuuluu neljä keskeistä tekijää: teknologia, työvälineet, ihmiset ja organisaation mallit. Prosessin täytyy kehittyä näiden tekijöiden lähtökohdista, ja niiden täytyy olla tasapainossa. Ohjelmistoprosessin tulee välttämättä olla kunnossa ja testattu, kun sitä käytetään käytännössä. Uuden tuotteen kehitykseen liittyy tarpeeksi riskejä ilman, että puutteellinen prosessi aiheuttaisi niitä vielä lisää.

UML luotiin ajatellen sitä, miten ohjelmistoprosessi voidaan viedä paremmin läpi koko sen elinkaaren. Se on graafinen kieli, joka on tarkoitettu ohjelmiston mallintamiseen, määrittelyyn, rakentamiseen ja dokumentointiin. Se on standardi apuväline ohjelmistoprosessin mallintamiseen, lähtien käsitteellisistä asioista eli liiketoimintamalleista ja järjestelmän tehtävistä. Se kattaa myös konkreettiset asiat, kuten luokat, tietokantakaaviot ja komponentit (Booch & al., 1999).

Vaikka UML onkin saavuttanut standardin aseman (se sai OMG standardin vuonna 1997), jatkuu sen kehitys edelleen. UML soveltuu käytettäväksi eri ohjelmistoprosessien kanssa apuvälineenä ja sen rinnalla on kehitetty myös prosessimalleja. Kuitenkaan UML:n tarjoamien kaavioiden käyttäminen hyvinkin erilaisten ohjelmistoprosessien kuvauksessa, ei ole läheskään selvä asia ja helppo sisäistää. Sen käyttöön otto ja tehokas käyttö vaativat yrityksiltä panostusta koulutukseen ja opetteluun. Prosessimallit tarjoavat apua UML:n käyttöönottamiseksi, mutta toisaalta prosessimallin käyttöönottokin vaatii opettelua.UML ei välttämättä vaadi jotain tiettyä suunnitteluprosessia tuekseen, mutta yleisesti sitä käytetään erilaisten prosessien apuvälineenä. Se on suunniteltu tukemaan pääasiassa iteratiivista, kasvua lisäävää ja käyttötapauslähtöistä suunnittelua, jossa keskeisessä asemassa on arkkitehtuuri (Rumbaugh & al., 1999). Sen avulla saadaan mallinnettua järjestelmän staattisia rakenteita ja dynaamisista käyttäytymistä. Järjestelmä on mallinnettu kokoelmana erillisiä olioita, jotka

ovat vuorovaikutuksessa keskenään saavuttaakseen tietyn, käyttäjää hyödyttävän tavoitteen (Rumbaugh & al., 1999).

On tärkeää muistaa, että UML ei ole formaali ohjelmointikieli vaan se on yleiskäyttöinen mallinnuskieli. Sen avulla ei pystytä mallintamaan kaikkea. Monen asian mallintamiseen voidaan tarvita formaalimpia menetelmiä. UML:ää käytettäessä täytyy olla koko ajan selkeästi mielessä, mitä tehtävä malli todella yritetään kuvata. UML:n käyttö vaatii käyttäjältään myös vankkaa olio-ohjelmoinnin menetelmien tuntemusta.

Tämän tutkielman tavoitteena on selvittää, miten UML tukee ohjelmistoprosessia ja miten sitä käytetään erilaisten ohjelmistoprosessien apuvälineenä. Tämä tehdään kahden prosessimallin avulla. Toinen on yleisesti tunnetumpi ja UML:n rinnalla kehitetty Unified Process (UP), jota käsitellään yhdessä Rational Unified Processin (RUP) kanssa. Tämä siksi, koska RUP:ia voidaan pitää UP:n ilmentymänä. Toisena prosessimallina käsitellään EDS Corporationissa käytettävää Global Solution Management System -prosessimallia (GSMS) ja tarkemmin sen OCE -työtyyppiä, joka voidaan ajatella olevan GSMS:n vastine UP:n RUP:lle. GSMS on suuren organisaation sisäisessä käytössä oleva malli ohjelmistotyön tekemiseen.

Tutkielman rakenne etenee siten, että seuraavassa luvussa esitellään UML pääpiirteittäin. Tämän jälkeen kolmannessa luvussa esitellään UP/RUP-prosessimalli, jossa kuvataan UP/RUP:n keskeisiä periaatteita ja sitä, miten prosessin eri vaiheissa käytetään UML:ää. Vertailukohtana edelliselle kuvataan neljännessä luvussa EDS Finland:ssa käytettävä OCE-prosessimalli ja sitä, miten sen yhteydessä käytetään UML:ää. Viidennessä luvussa tarkastellaan prosessien vahvuuksia, UML:n roolia molemmissa prosesseissa ja arvioidaan prosessien hyviä ja huonoja puolia sekä eroja. Toisaalta tarkastelua tehdään myös UML:n näkökulmasta tutkimalla UML:n puutteita, sitä, mitä UML:n käytössä tulisi ottaa huomioon, ja tutustutaan lyhyesti mallinnusvälineisiin liittyvää problematiikkaa. Lopuksi kuudennessa luvussa tehdään lyhyt yhteenveto käsitellyistä asioista sekä mahdollisista jatkotutkimusmahdollisuuksista.

2 UML

Booch & al. (1999) mukaan UML on graafinen kieli, jonka avulla voimme visualisoida, määritellä, rakentaa ja dokumentoida *artefakteja* ohjelmistojärjestelmässä. Artefakti on Jacobsonin & al. (1999) mukaan informaatiokokonaisuus, jota käytetään tai tuotetaan ohjelmiston kehitysprosessin aikana.

Tässä luvussa esitellään lyhyesti UML:n peruskäsitteitä ja kaikki sen tarjoamat kaavioesitykset. Koska UML on eri kaaviotyyppeineen ja yksityiskohtineen laaja kieli, on tässä yhteydessä annettava kuvaus melko suppea. Tarkoituksena ei ole opettaa UML:n käyttöä, vaan pyrkiä antamaan käsitys sen merkintätavasta.

2.1 UML:n historia

UML ei ole syntynyt tyhjästä, vaan se on kehitetty aikaisempien mallinnusmenetelmien pohjalta. UML syntyi 90-luvun lopulla, kun kolme menetelmäkehittäjää, Booch, Rumbaugh ja Jacobson, yhdistivät voimansa ja muodostivat omista kaavioesityksistään universaalin notaatioon.

UML:n kehittäminen on siirtynyt tämän jälkeen OMG- yrityskonsortion (Object Management Group) tehtäväksi. OMG ei ole liikeyritys vaan voittoa tuottamaton organisaatio, joka ylläpitää eri teollisuusstandardeja ja edistää olioteknologioita. UML:stä puhuttaessa on syytä muistaa, että koska UML on kohtuullisen uusi väline (kehitystyö alkoi 1994, OMG standardi 1997), kehittyy se edelleen melko voimakkaasti. Toisaalta perusrakenteen osalta muutoksia ei ole odotettavissa (Jacobson, 1999).

2.2 UML:n perusajatus

UML:n perusajatuksena on tarjota järjestelmän kehittäjille apuväline, jonka avulla jokainen järjestelmän kehityksessä mukana oleva henkilö pystyy ymmärtämään paremmin toistensa työn tulokset. Kun kaikki mukana olevat henkilöt käyttävät samaa mallinnusmenetelmää ja ymmärtävät, miten se toimii, pystyvät he paremmin keskustelemaan keskenään kehitettävästä järjestelmästä. UML:n avulla voidaan kuvata järjestelmän staattinen rakenne ja dynaaminen käyttäytyminen. Staattisen rakenteen kuvaamisessa järjestelmästä esitetään ne oliot, jotka ovat tärkeitä järjestelmän kannalta, ja niiden väliset suhteet. Dynaaminen käyttäytyminen kuvaa, kuinka olio on vuorovaikutuksessa ympäristönsä kanssa.

On tärkeää muistaa, että UML ei ole suunnittelumalli, vaan ohjelmiston mallien kuvaustapa, graafinen suunnittelukieli. Se ei sisällä mitään suunnitteluprosessia. UML tukee ohjelmistoprosessin suunnittelua ja sisältää käsitteet, joiden avulla ohjelmisto voidaan kuvata. Kuvaus tehdään erilaisten kaavioesitysten perusteella, joiden avulla mallinnetaan erilaisia asioita suunniteltavasta ohjelmistosta.

2.3 UML-kaaviot

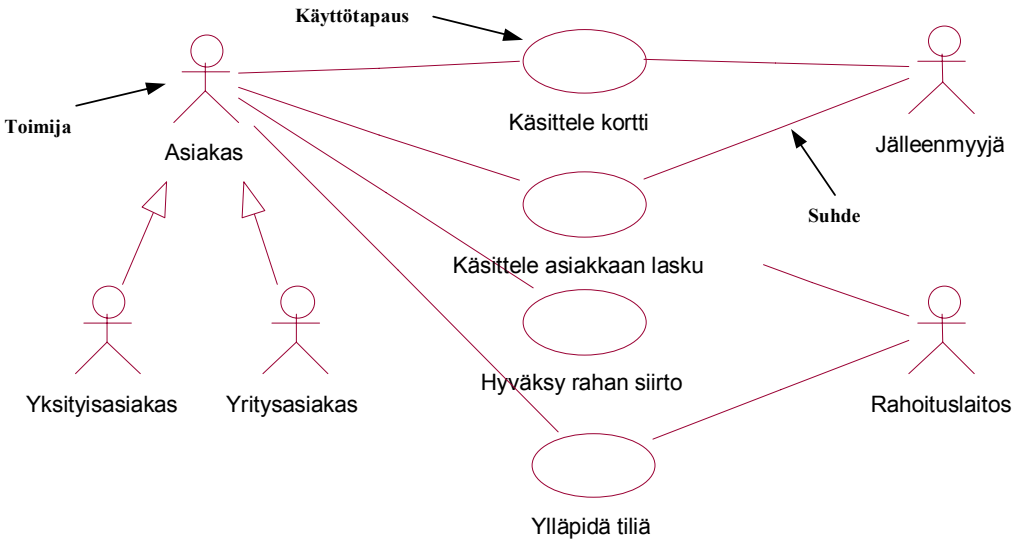
UML:n kuuluu yhteensä yhdeksän eri kaaviota, joiden avulla järjestelmää mallinnetaan. Järjestelmän korkean tason toiminnallisuutta kuvataan *käyttötapauskaavioilla*, järjestelmän staattista ja dynaamista rakennetta *oliokaavioilla*, *luokkakaavioilla*, *sijoittelukaavioilla* ja *komponenttikaaviolla*. Järjestelmän dynaamista käyttäytymistä kuvataan *sekvenssikaavioilla*, *yhteistyökaavioilla*, *tilakaavioilla* ja *aktiviteettikaavioilla*. Näissä kaavioissa esiintyy samoja symboleja. Kaavioesitykset ovat osittain päällekkäisiä (esim. sekvenssikaavio ja yhteistyökaavio kuvaavat asioita melko samalla lailla). Eri kaaviotyypit soveltuvat eri ohjelmistokehityksen vaiheisiin käytettävistä menetelmistä riippuen.

Kaavioesityksiä laadittaessa täytyy koko ajan olla selkeästi mielessä se, mitä tarkoitusta varten kaaviota ollaan laatimassa. Jos ollaan laatimassa analyysivaiheen luokkakaaviota, joka kuvaa aluekohtaiset luokat, ei esityksen tarkoituksena ole antaa yksityiskohtaista kuvaa siitä, miten järjestelmä toteutetaan, vaan tarkoitus on esimerkiksi antaa suunnittelijoille ja järjestelmän tilaajalle mahdollisuus ymmärtää järjestelmän toimintaa paremmin. Tällöin tehtävässä luokkakaaviossa täytyy pysyä tarvittavan korkealla abstraktiotasolla, välttäen tekemästä liian yksityiskohtaista kuvausta asiasta. Booch & al. (1999) mainitsevat, että järjestelmä näytetään useasta eri näkökulmasta ja samaa näkymää tarvitsevat ihmiset tarvitsevat sen usealta eri abstraktiotasolta.

2.3.1 Käyttötapauskaaviot

Käyttötapauskaaviot kuvaavat sitä, miten ulkoinen käyttäjä käyttää järjestelmää. Niiden avulla voidaan tunnistaa järjestelmän ulkoiset käyttäjät ja pääominaisuudet, joita järjestelmän tulee tarjota. Käyttötapauskaaviot soveltuvat järjestelmän vaatimusten kuvaukseen, koska ne keskittyvät järjestelmän kannalta olennaisten asioiden, ominaisuuksien ja toiminnallisuuden kuvaamiseen. Ne ovat myös erityisen tärkeitä järjestelmän käyttäytymisen organisoimisen ja mallintamisen kannalta (Booch & al., 1999).

Yksittäinen käyttötapaus on yhden järjestelmässä olevan toiminnon kuvaus. Kaavioiden keskeiset osat ovat *toimija*, *käyttötapaus* ja niiden välinen *suhde*. Kuvassa 2.1 on kuvattu yksinkertainen käyttötapauskaavio ja nimetty käytetyt peruselementit.



.

Kuva 2.1. Luottokorttijärjestelmä -käyttötapauskaavio (Booch & al., 1999) .

Jacobson & al. (1999) mukaan toimija on järjestelmän ulkoinen käyttäjä tai toinen järjestelmä, joka on vuorovaikutuksessa kuvattavan järjestelmän kanssa. Toimija voi toisaalta antaa järjestelmään syötettä tai saada siltä tietoa. Käyttötapaus on sarja tapahtumia, myös vaihtoehtoisia tai poikkeavia, joita järjestelmä suorittaa toimijan käyttäessä järjestelmää. Suhde kuvaa yhteyttä näiden kahden elementin välillä.

Käyttötapauskaavioon sisältyy aina myös kuvaus jokaisesta yksittäisestä käyttötapauksesta. Tämä voidaan kuvata usealla eri tavalla. Kuvan 2.1. kaavion Käsittele asiakkaan lasku -käyttötapauksen kuvaus voisi olla esimerkiksi seuraavanlainen (kuva 2.2).

Käyttötapaus	**Käsittele asiakkaan lasku**
Esitila	Käsittele kortti -käyttötapaus on suoritettu onnistuneesti.
Toimijat	Asiakas, Jälleenmyyjä, Rahoituslaitos
Kulku	Jälleenmyyjä syöttää laskutettavan hinnan järjestelmään. Järjestelmä tarkastaa, että asiakkaalla on korttia vastaava tili rahoituslaitoksessa (E-1). Järjestelmä tulostaa kuitin allekirjoitusta varten (E-2). Jälleenmyyjä voi keskeyttää käyttötapauksen milloin tahansa (E-3).
Vaihtoehtoinen kulku	E-1: Kortin ja rahoituslaitoksen tiedot eivät täsmää. Järjestelmä tuottaa virheilmoituksen. Käyttötapaus keskeytyy. E-2: Kuitin tulostus epäonnistuu. Järjestelmä tulostaa virheilmoituksen. E-3: Järjestelmä ilmoittaa operaation keskeytyksestä ja keskeyttää käyttötapauksen.
Jälkitila	-

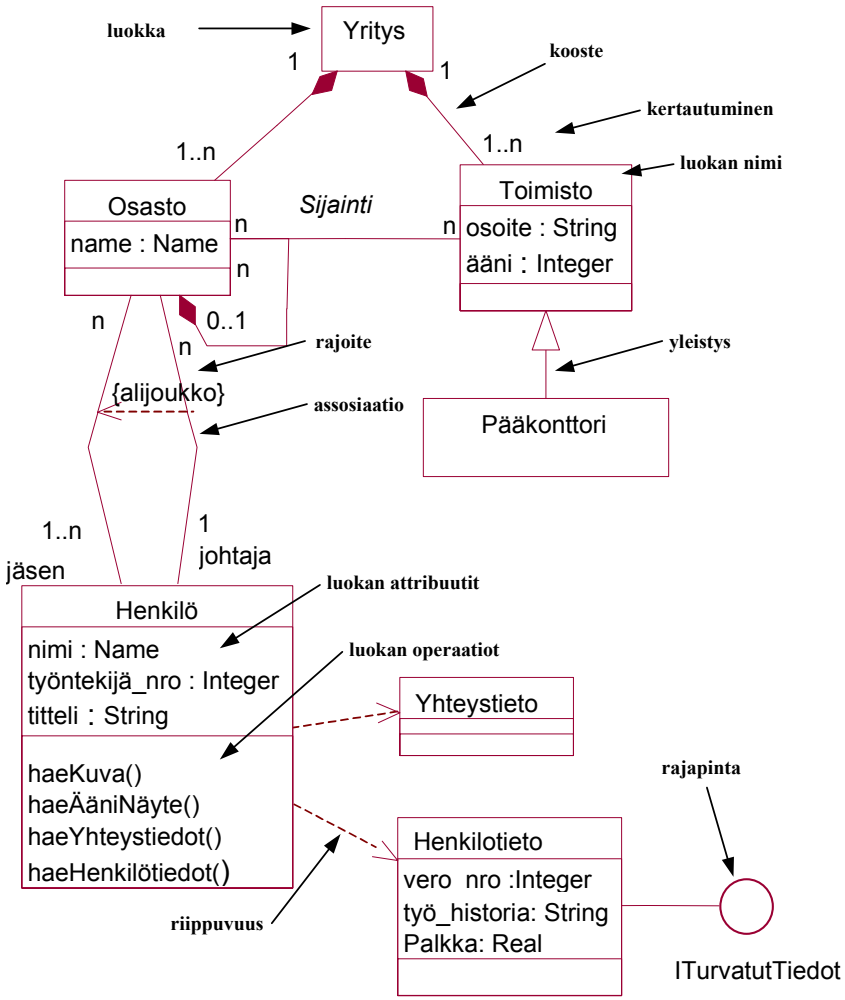
Kuva 2.2. Käyttötapauksen kuvaus.

Käyttötapauksia voidaan kuvata monella eri abstraktiotasolla. Vaatimusmäärittelyä tehdessä ja asiakkaan kanssa järjestelmästä keskusteltaessa ne ovat yleisellä tasolla, kuten kuvassa 2.1. Jos abstraktiotasoa tarkennetaan, voidaan alijärjestelmät (kuvan 2.1 toimijat Jälleenmyyjä ja Rahoituslaitos) kuvata myös omina käyttötapauskaavioinaan.

2.3.2 Luokkakaavio

Kaikkein käytetyin ja tärkein UML-kaavioista on luokkakaavio. Luokkakaaviot kuvaavat järjestelmään kuuluvia luokkia ja niiden välisiä suhteita. Luokat kuvaavat asioita, joita järjestelmä käsittelee. Luokkakaavio on staattinen. Se näyttää tiedon rakenteen lisäksi myös sen, miten tieto käyttäytyy.

Keskeinen käsite luokkakaavioista puhuttaessa on luokkien väliset suhteet. Niistä tärkeimmät ovat *riippuvuus*, *assosiaatio*, *yleistys* ja *toteutus*. Näiden neljän suhteen avulla voidaan käytännössä mallintaa suurin osa luokkakaavioilla mallinnettavista asioista.

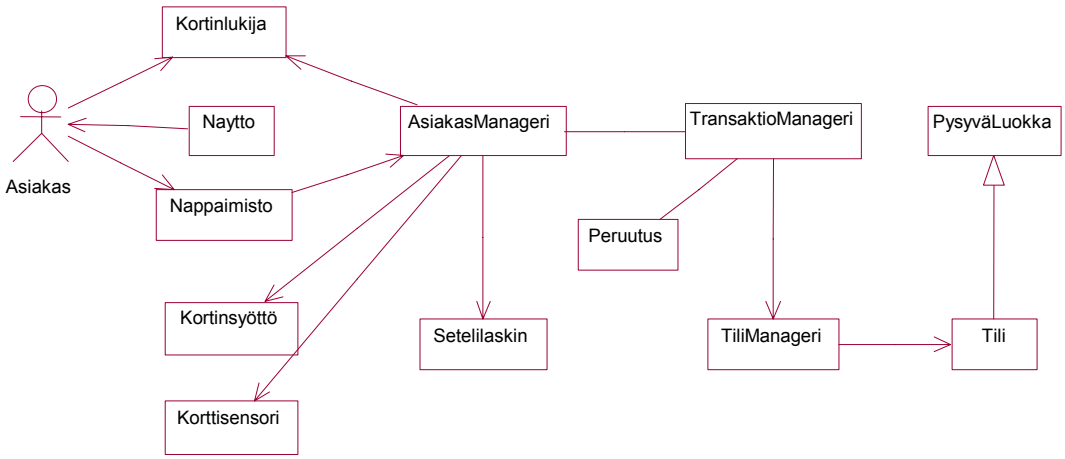


Kuva 2.3. Luokkakaavio (Booch & al., 1999).

Luokan attribuutit kuvaavat luokan ominaisuudet ja luokan operaatiot sen käyttäytymisen. Luokkiin tai niiden suhteisiin voi kuulua myös rajoitteita, joilla voidaan tarkentaa kuvausta. Rajoite on yksi UML:n laajentamismekanismeista. Rajapintojen avulla kuvataan luokkien ulospäin näkyvä ja käsiteltävä käytös.

Luokkakaavioiden avulla kuvattavat asiat voivat myös olla monella eri abstraktitasolla. Tämä vaikuttaa osaltaan myös siihen, millaisia luokkakaavioita käytetään. Luokkien attribuutteja ja operaatioita voidaan jättää näyttämättä, jos niillä ei ole mallinnettavan asian kannalta siinä vaiheessa merkitystä. Luokkakaavio voi esimerkiksi aikaisessa UP/RUP:n mukaisessa suunnitteluvaiheessa sisältää pelkästään luokkien nimet ja assosiaatiosuhteet. Tästä esimerkkinä on kuvan 2.4 luokkakaavio, joka kuvaa erään käyttötapauksen toteuttamisen (use

case realization) osaa. Luokkakaavio esittää samaa asiaa tarkemmalla tasolla kuin kuvassa 2.6 oleva UP/RUP:n analyysin tuottama luokkakaavio.



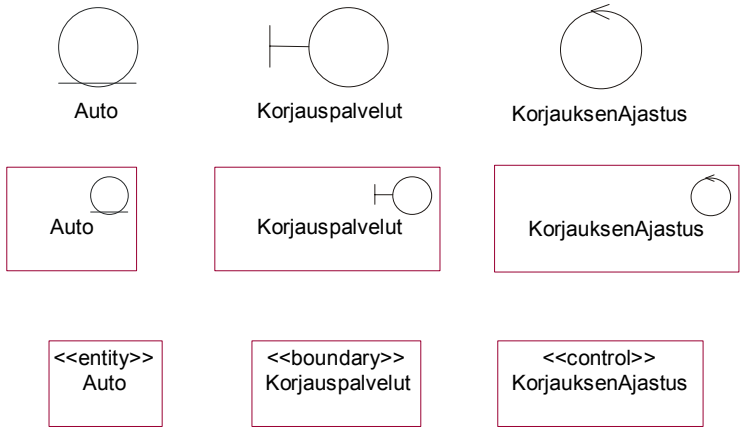
Kuva 2.4. Eräs suunnitteluvaiheen luokkakaavio UP/RUP:n mukaan. (Jacobson & al, 1999)

Mallinnettaessa järjestelmän staattista näkymää, on Boochin & al. (1999) mukaan kolme tyypillistä tapaa käyttää luokkakaavioita. Nämä ovat sanaston mallintaminen, yhteistoiminnan kuvaus ja tietokantakaavion mallintaminen. Sanaston mallintamisen avulla saadaan tehtyä päätöksiä siitä, mitkä abstraktiot ovat osa järjestelmää ja mitkä jäävät sen rajojen ulkopuolelle. Yksinkertaisen luokkien yhteistoiminnan avulla mallinnetaan, miten luokat toimivat yhdessä. Yhtä luokkaa tutkimalla ei voi saada selville, miten ko. systeemin osa toimii. Vasta luokkien yhteistoiminta saa aikaan tietyn toiminnan. Luokkakaavioiden avulla voidaan mallintaa myös järjestelmän looginen tietokantakaavio. Luokkakaavioita käytetään tällöin pysyvän tiedon säilyttämisen mallintamiseen.

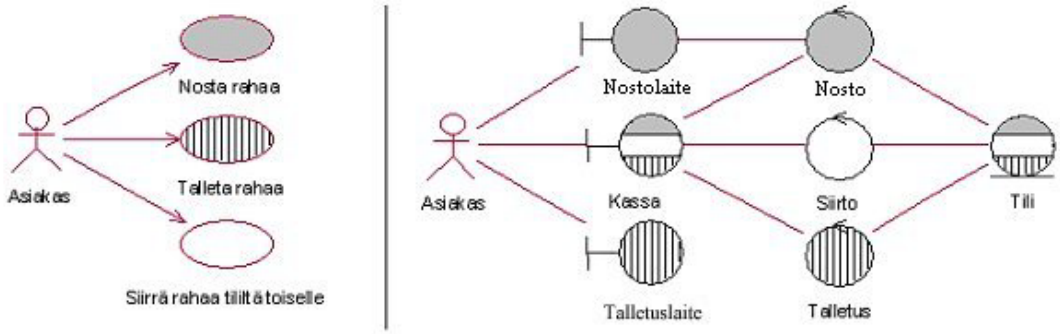
Luokkakaavioiden semantiikkaa voidaan myös laajentaa UML-laajennosmekanismien avulla. UML:ssä on mallinnuselementtityyppi nimeltään stereotyyppi, joka perustuu jo määriteltyihin elementteihin laajentaen niiden merkitystä. Rumbaugh & al (1999) määrittelevät sen eräänlaiseksi virtuaaliseksi metamalliksi, joka on pantu toisen mallin sisään. On katsottu järkevämmäksi tehdä näin kuin muokata aikaisemmin määriteltyä luokan metamallia, jonka UML määrittää. UML sisältää lukuisia, valmiiksi määriteltyjä stereotyyppejä, ja niitä voi määritellä myös itse.

UP:n ja RUP:n yhteydessä on luotu kolme stereotyyppiä luokille, jotka on standardoitu UML:ään. Niitä käytetään UP/RUP prosessin mukaisessa analyysivaiheessa. Nämä ovat *olio-*

(entity), *raja-* (boundary) ja *ohjausluokka* (control). Standardoinnin ansiosta näille stereotyypeille on kehitetty oma notaatio, joka osaltaan aiheuttaa muutoksia luokkakaavioiden ulkonäössä. Näiden stereotyypitettyjen luokkien mahdolliset kuvaustavat ovat kuvassa 2.5. Niiden käyttöä UP:n mukaisesti on kuvattu kuvassa 2.6, jossa on lisäksi mukana kuviointi selvittämässä sitä, mitä käyttötapausta luokka vastaa.



Kuva 2.5. Standardeja luokan stereotyyppejä kuvattuna erilaisilla kuvaustavoilla.



Kuva 2.6. Analyysiluokkakaavion ja sen käyttötapauksien yhteyden kuvaus (Jacobson & al., 1999).

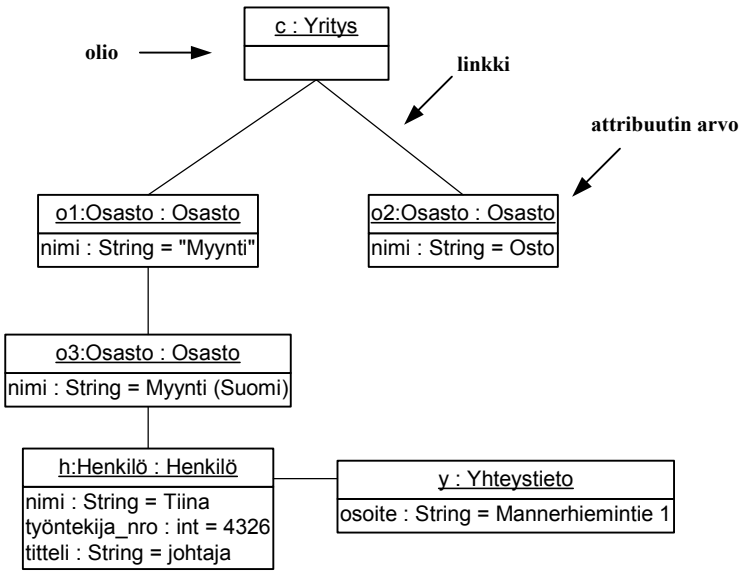
Siihen, millaista luokkakaaviota tulisi milloinkin käyttää ja miten tämä pitäisi toteuttaa, ei ole yksikäsitteistä vastausta. Prosessimalli voi määritellä, millaisia luokkakaavioita käytetään milloinkin. Toisaalta tällaisten mallien ymmärtäminen prosessia tuntemattomien osalta voi tulla mahdottomaksi. Kuvassa 2.6 esitetty notaatio kuuluu UP/RUP:n. Käyttäjä, joka ei tunne prosessimallia, ei ymmärrä mitä kaavio tarkoittaa. Tällaisen notaation käyttö ei ole perusteltua, jos projekti ei käytä kyseessä olevaa prosessimallia. Perusnotaatiota käyttämällä

(kuva 2.3) pystytään mallintamaan sama asia, kuin UML:n laajennosten tarjoaminen uusien kuvaamistapojen avulla.

2.3.3 Oliokaavio

Oliokaavio kuvaa olioita ja niiden välisiä suhteita. Se näyttää otoksen järjestelmän tilasta eli siitä, miltä järjestelmä näyttää tietyllä hetkellä. Oliokaavion notaatio muistuttaa suuresti luokkakaavion notaatiota. Oliokaaviossa on kuitenkin luokkien sijasta luokan ajonaikaisia ilmentymiä.

Kuten luokkakaaviossa luokat, on oliokaaviossa oliot yhdistetty toisiinsa. Nämä yhteydet ovat nimeltään *linkkejä*. Linkki on luokka- ja oliokaavion välisen suhteen vuoksi assosiaation ajonaikainen ilmentymä.



Kuva 2.7. Oliokaavio (Booch & al, 1999).

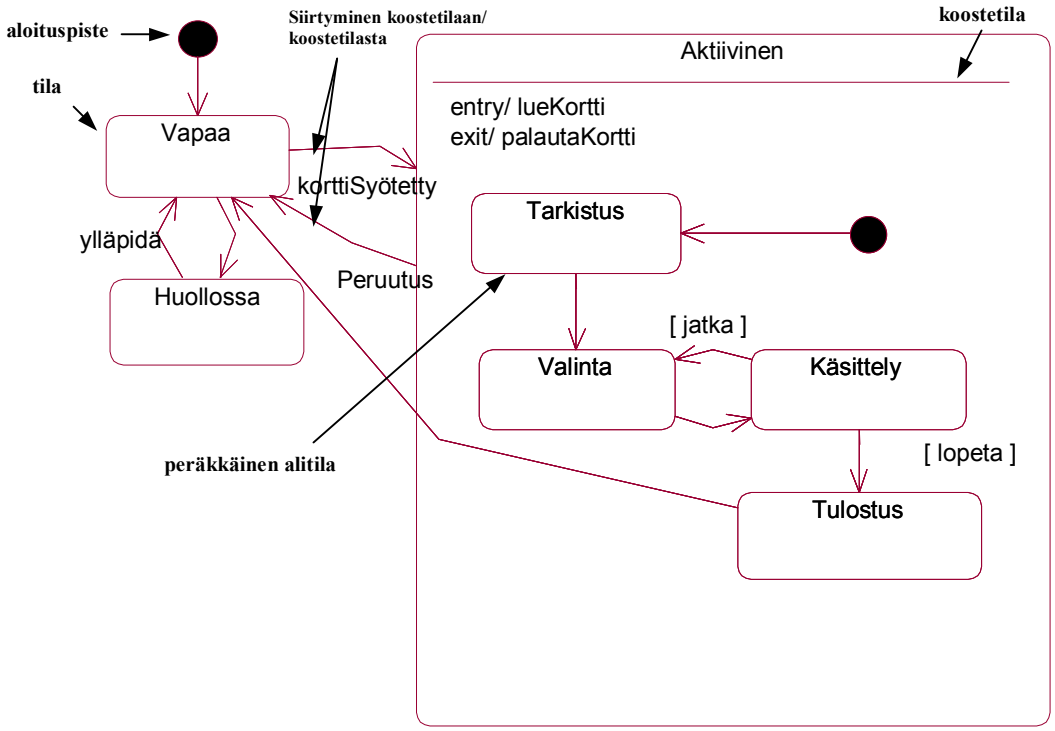
Kuten kuvassa 2.7 nähdään, oliokaaviossa voi olla useita ilmentymiä samasta luokasta ja Olioiden nimet on alleviivattu. Nämä kaksi seikkaa ovatkin ainoat erot luokkakaavion ja oliokaavion notaatioiden välillä. Olioiden attribuuttien arvot näkyvät otoksen hetkellä olioiden sisällä.

On tärkeää muistaa, ettei oliokaavioiden avulla voida määritellä systeemin toimintaa. Oliokaavio voi olla käytännöllinen, jos tarvitaan esimerkkiä systeemin toiminnasta. Tällaisia esimerkkejä voivat olla monimutkaisen tietomallin havainnollistaminen tai otosten sarja, jolla

näytetään olioiden käyttäytyminen jollakin ajanjaksolla (Jacobson, 1999). Oliokaavioiden rooli ei ole kuitenkaan tärkeä suhteessa luokkakaavioihin.

2.3.4 Tilakaavio

Luokan kuvaus yksinään ei välttämättä kerro kaikkea olennaista informaatiota luokasta. Tilakaaviota voidaan käyttää apuna lisäinformaation antamiseen. Kaaviossa näytetään *tilat*, joihin oliot voivat joutua, ja *tapahtumat*, jotka aiheuttavat *tilasiirtymät*. Tilasiirtymään voi liittyä myös *toiminto*, joka tehdään, kun siirtymä tapahtuu. Tilakaavio piirretään pääasiassa *aktiivisille luokille*. Aktiivinen luokka on Rumbaughin & al (1999) mukaan luokka, jonka ilmentymät ovat *aktiivisia olioita*. Nämä puolestaan ovat olioita, joilla on oma säikeensä, jonka sisällä ne suoritetaan. Niillä on myös oma pino ja omat tilansa. Niillä on itsenäinen kontrolli toimintoihinsa järjestelmän suorituksen sisällä. Tilakaavioita voidaan aktiivisten luokkien lisäksi piirtää myös koko järjestelmälle.



Kuva 2.8. Pankkiautomaatin tilakaavio (Rational, 2001a).

Tilakaavio (kuva 2.8) voi sisältää myös aloitustilan ja lopetustilan, jotka kertovat tilakaavion alku- ja lopputilan. Näitä pisteitä voi olla useita. Alku- ja lopputila eivät ole kuitenkaan pakollisia. Tilaa kuvaavassa UML mallissa on kolme eri osaa. Ensimmäisessä osassa on tilan nimi. Toinen osa sisältää valinnaiset muuttujat, jossa attribuutteja alustetaan. Tämä osa on

valinnainen. Kolmas osa on toiminto-osa, jossa tapahtumia ja toimintoja luetellaan. Myös tämä osa on valinnainen.

UML:n tilakaavio sisältää kolme standardoitua tapahtumaa, jotka ovat *aloitustapahtuma* (entry), *lopetustapahtuma* (exit) ja *toistotapahtuma* (do). Aloitustapahtuma määrittää toiminnot, jotka suoritetaan tilaan tultaessa. Vastaavasti lopetustapahtuma määrittää ennen tilasta pois siirtymistä suoritettavat tapahtumat. Toistotapahtuman kertoo, mitä tilassa oltaessa tehdään. Kaikkia mainittuja tapahtumia voi tilassa olla useita.

Olion tilan muuttuessa tapahtuu tilasiirtymä. Tilasiirtymä laukeaa, kun siihen liittyvä laukaisin aktivoituu eli jotain tapahtuu. Siirtymä voi olla *normaali siirtymä* tai *itseensä palaava siirtymä*, jolloin tilan lopetus- ja aloitustapahtumat suoritetaan siirtymän tapahtuessa. Siirtymä voi olla myös *sisäinen siirtymä*, jolloin siirtymä tapahtuu kyseistä tilaa jättämättä. Tämä tarkoittaa sitä, ettei aloitus ja lopetustapahtumia suoriteta.

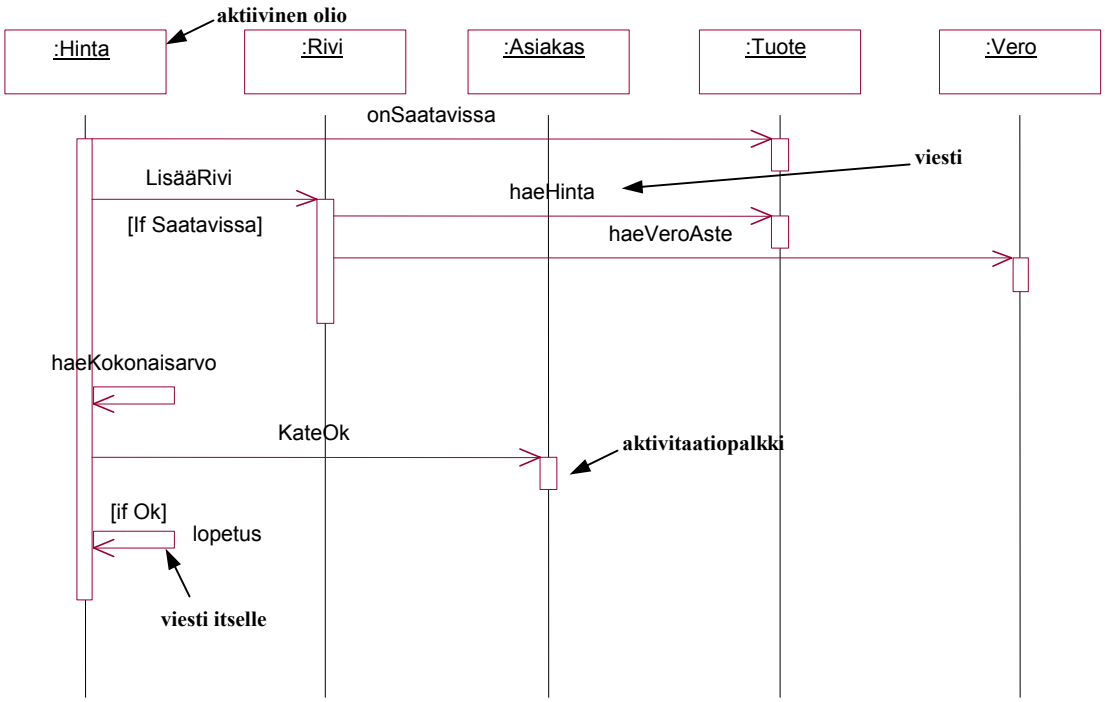
Tiloihin voi liittyä myös hierarkkisuutta. Kuten kuvasta 2.8 voi nähdä, voi tila olla *koostetila*, jolloin se koostuu *peräkkäisistä alitiloista*. Peräkkäiset alitilat muodostavat alitilakaavion, joka sisältää siirtymiä samalla tavalla kuin normaali tilakaavio. On mahdollista, että koostetila sisältää *rinnakkaisia alitiloja*. Rinnakkaiset alitilat suoritetaan nimensä mukaisesti rinnakkain. Tilasta voidaan siirtyä seuraavaan tilaan vasta sen jälkeen, kun kaikki rinnakkaiset alitilat ovat saavuttaneet lopputilansa.

Rumbaugh & al. (1999) antavat tilakaavioille kahdenlaisia käyttötarkoituksia. Tästä syystä ne voidaan myös ymmärtää kahdella tavalla. Tilakaavio voi kuvailla isäntäelementin (master element) käyttäytymisen suorituksen aikana. Tämä on yleensä luokka. Tällöin tilakaavio kuvaa isäntäelementin reagoimisen ulkoisiin laukaisimiin. Laukaisu aiheuttaa siirtymisen tilasta toiseen. Tietyssä tilassa saatu tapahtuma aiheuttaa tietynlaisia tapahtumia ja tietynlaisiin tiloihin siirtymisiä.

Toinen tilakaavioiden käyttötarkoitus on protokollien määrittely, jolloin tilakaavion avulla kuvataan, missä järjestyksessä luokan tai rajapinnan operaatioita voidaan kutsua. Tällöin laukaisimina toimivat kutsut. Laukaisu aktivoi halutun operaation tilassa. Tällä kuvataan sitä, että kutsuja saa kutsua operaatiota tässä kohdassa. Kaavio ei kuvaa toimenpiteitä operaation itsensä käyttäytymisen kuvaamiseksi.

2.3.5 *Sekvenssi- eli viestiyhteyskaavio*

Sekvenssi- eli viestiyhteyskaavio on dynaaminen kaavio, joka kuvaa olioiden välistä yhteistyötä tietyssä tilanteessa. Se on toinen ja yleisemmin käytetty vuorovaikutuskaavio. Usein kuvaus on jonkin käyttötapauksen toteutuessa tapahtuva vuorovaikutus. Kaavio näyttää, miten *viestit* etenevät olioiden välillä. Olioiden välinen vuorovaikutus kuvataan viestinä yhdeltä oliolta toiselle. Viestit ovat monesti yksinkertaisia operaatiokutsuja. Aika kulkee kaaviossa ylhäältä alaspäin, ja vuorovaikutustapahtumat kuvataan vaakatasossa kulkevina nuolina lähettäjältä vastaanottajalle.

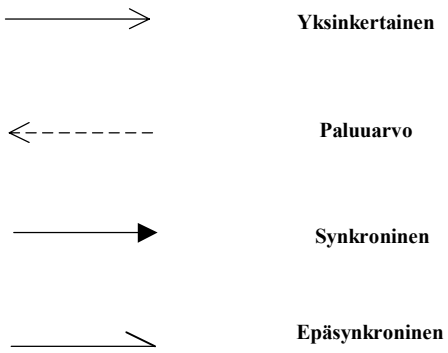


Kuva 2.9. Sekvenssi- eli viestiyhteyskaavio (Marshall, 1999).

Rumbaughin & al. (1999) mukaan viesti on informaation siirto yhdeltä objektilta (lähettäjä) toiselle (vastaanottaja). Tämä kutsu aiheuttaa aina jotain aktiviteettia. Viesti voi olla signaali tai operaatiokutsu, joka kuvataan *aktivitaatiopalkilla*.

Sekvenssikaaviossa ei ole välttämätöntä käyttää algoritmisia ilmaisuja, kuten ehdollisuus ja toisto. Niiden käyttö on kuitenkin mahdollista. Tällöin voidaan vähentää tarvittavien sekvenssikaavioiden määrää, koska samassa kaaviossa voidaan kuvata eri tapahtumasarjoja. Liiallinen algoritmisten ilmaisujen käyttö voi toisaalta aiheuttaa sen, että kaavion luettavuus kärsii.

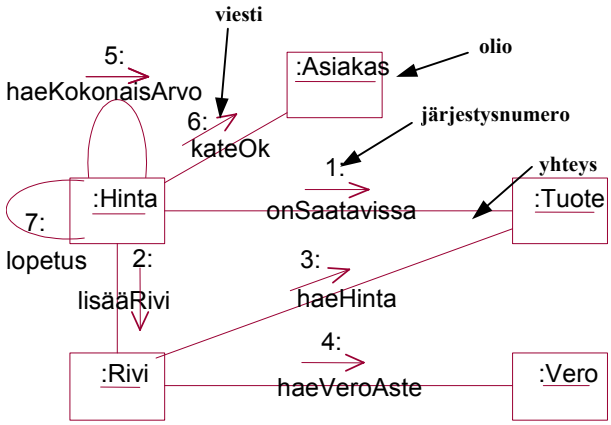
Viestiyhteyskaavio sisältää viestityyppejä, joiden notaatio on kuvattu kuvassa 2.10. Yksinkertainen viestiyhteystyyppi kertoo, että suoritus etenee yhdeltä olion tyypiltä toiselle. Tyyppiä käytetään, kun yksityiskohtia ei ole syytä kertoa tai ne eivät ole tärkeitä. Paluuarvon välittäminen tehdään samanlaisella nuolella kuin viestiyhteystyypillä on, varsi on vain katkoviiva. Synkronisella viestiyhteystyypillä kuvataan suorituksen siirto, joka toteutetaan yleisimmin operaatiokutsuna. Epäsynkronisella viestiyhteystyypillä kuvataan suorituksen siirto, jossa ei ole palautusta kutsujaan. Suoritus jatkuu kutsujan toimesta huolimatta siitä, palautetaanko kutsutusta oliosta jotain vai ei. Myös joitain viestiyhteyksien laajennoksia on määritelty (Rumbaugh & al., 1999). Yksi on esitetty kuvassa 2.9 (viesti oliolta itselle).



Kuva 2.10. Eri viestityyppien merkintätavat.

2.3.6 Yhteistyökaavio

Yhteistyökaavio on toinen vuorovaikutuskaavioista. Yhteistyökaavio kuvaa myös olioiden välistä yhteistyötä, kuten sekvenssikaaviokin, mutta erona on se, ettei siinä aikaa edusta mikään suunta. Se näyttää lisäksi olioiden väliset suhteet toisiinsa. Yhteistyökaaviossa osallistujat voidaan asettaa mielivaltaisiin paikkoihin ja tämän avulla saadaan ilmaistua, mitkä oliot kuuluvat loogisesti yhteen.



Kuva 2.11. Yhteistyökaavio.

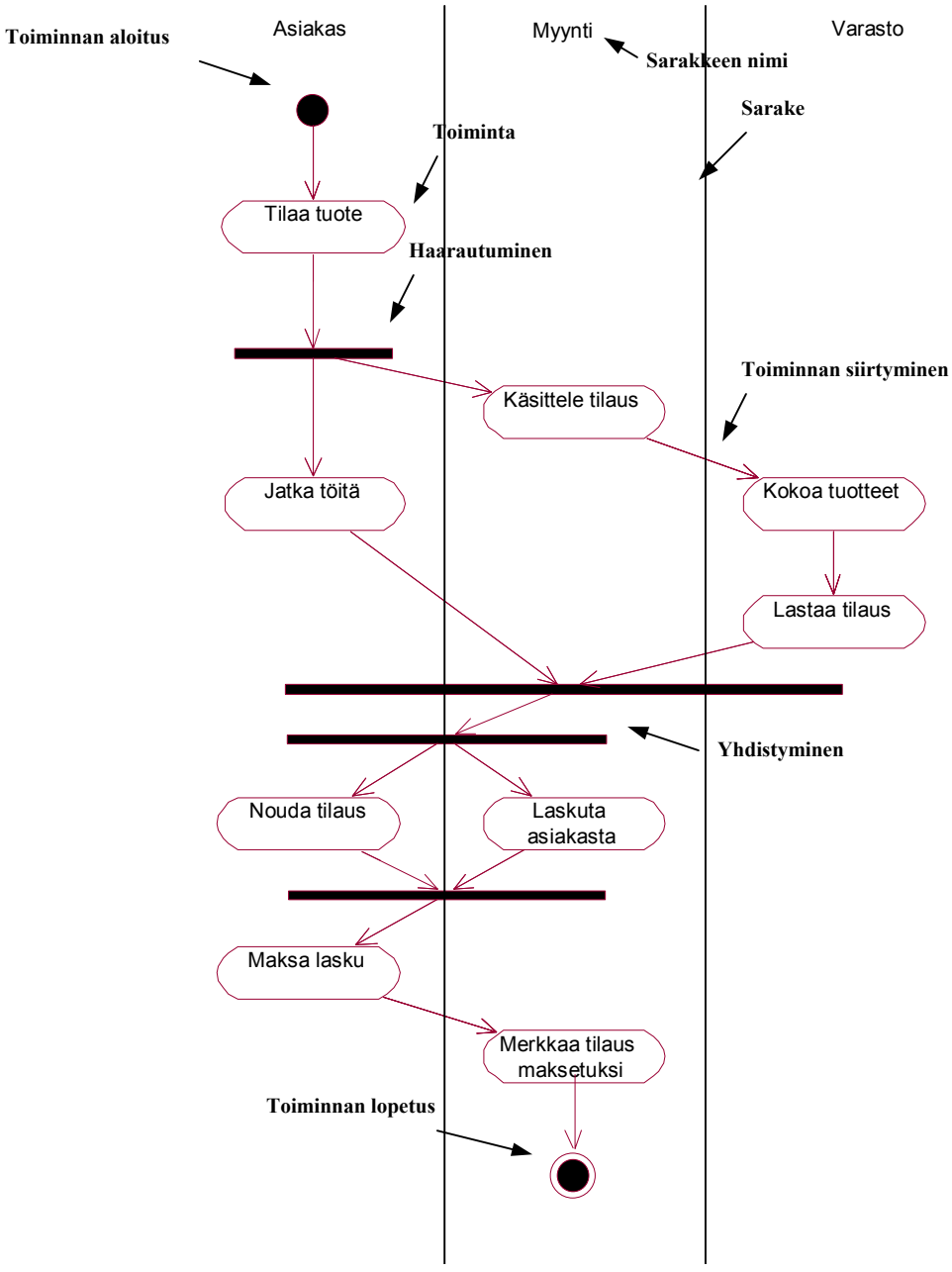
Kuvassa 2.11 on kuvattu kuvassa 2.10 esiintyvää viestiyhteyskaaviota vastaava yhteistyökaavio. Kaavio kuvaa polun läpi operaation suorituksen. Oliot on yhdistetty yhteydellä toisiinsa. Suorituksen etenemisen seuraamista helpottamaan viesteille on lisätty järjestysnumero. Järjestysnumerot on kuvassa 2.11 näytetty yksinkertaisesti juoksevien numeroiden sarjana. Ne voitaisiin esittää myös alanumeroiden avulla siten, että sisäiset viestit saisivat saman pääluvun. Esimerkiksi kuvan 2.11 haeHinta -viestin järjestysnumero voisi olla 2.1 ja haeVeroAste -viestin 2.2, koska ne ovat lisääRivi -viestin (2) sisäisiä viestejä.

Yhteistyökaavio mahdollistaa lisäksi iteraation ja haarautumisen kuvaamisen. Iteraatiossa järjestysnumeron yhteyteen lisätään hakasulkuihin toistojen määrä. Tämä annetaan hakasulkujen sisällä ja se voisi olla esimerkiksi 3.5 *[x := 1..5]: kasvataLkm. Tällöin operaatio toistetaan viisi kertaa. Haarautuminen kuvataan myös hakasulkujen sisällä. Esimerkiksi merkintä 2.3 [y>=3]: kasvataLkm tarkoittaa sitä, että operaatio suoritetaan ehdossa kuvatun lauseen ollessa tosi. Yhteyksissä voidaan näyttää lisäksi olioiden roolinimet ja tarkentimet. Näitä roolinimiä voidaan käyttää myös luokkakaavioissa. Yhteistyökaaviossa, kuten viestiyhteyskaaviossakin, voidaan näyttää lisäksi yhteistyön aikana luotuja uusia olioita, tuhottuja olioita tai molempia.

Sekä viestiyhteys että yhteistyökaavio kuvaavat olioiden välistä viestintää ja vuorovaikutusta, mutta viestiyhteyskaavio keskittyy kuvaamiseen enemmän ajankulun näkökulmasta. Yhteystyökaavioiden näkökulmasta tärkeämpi tekijä on ympäristö ja olioiden väliset suhteet. Molemmat vuorovaikutuskaaviot ovat semanttisesti niin samanlaisia, että niiden käyttötilanteet ovat myös samantyyppisiä. Booch & al. (1999) nimeävät kaksi tapaa käyttää eri vuorovaikutuskaavioita, kun mallinnetaan järjestelmän dynaamisia ominaisuuksia. Ensimmäinen tapaus on tapahtumavirran mallintaminen aikajärjestyksessä. Tähän

tarkoitukseen käytetään viestiyhteyskaaviota. Tämä on erityisen käyttökelpoinen mallinnettaessa käyttötapausten skenaarioiden dynaamista käyttäytymistä. Toinen tapaus, jolloin käytetään mieluummin yhteistyökaaviota, on tapahtumavirran kulun kuvaus järjestelmässä. Tällöin pääpaino on enemmän ilmentymien yhteistyön kuvaamisessa.

2.3.7 Aktiviteetti- eli toimintokaavio



Kuva 2.12. Aktiviteetti- eli toimintokaavio.

Aktiviteetti- eli toimintokaavio kuvaa tietyn tehtävän sisäisen logiikan. Tehtävä voi olla yksittäinen operaatio, käyttötapaus tai viestin välitys. Se on tilakaavion (kuva 2.8)

erikoistapaus. Toimintokaavio koostuu tiloista, jotka sisältävät suoritettavan tapahtuman määritelmän, ja niitä yhdistävistä siirtymistä. Tilasta siirrytään heti pois, kun toiminto on saatu suoritettua, toisin kuin tilakaaviossa, jossa tilasta poistutaan vasta, kun viesti on vastaanotettu.

Booch & al. (1999) kuvaavat toimintokaavion ja vuorovaikutuskaavion eron: kun vuorovaikutuskaavioilla mallinnetaan kontrollivirtaa oliolta toiselle oliolle, toimintokaavio keskittyy kuvaamaan kontrollivirtaa toiminnolta toiselle toiminnolle. Toimintokaavion voi käsittää ympärikäännetyksi vuorovaikutuskaavioksi. Vuorovaikutuskaavio näyttää oliot välittämässä viestejä. Toimintokaavio näyttää operaatiot, jotka välitetään olioiden kesken.

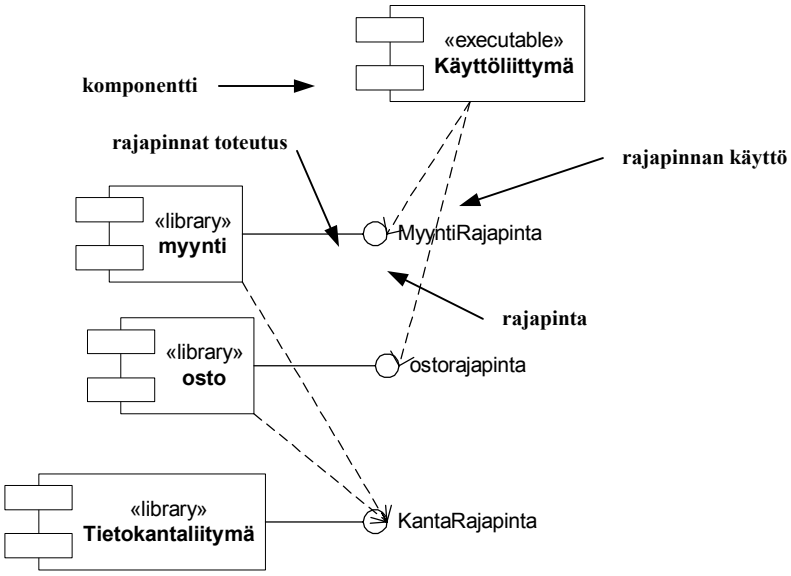
Kuten kuvassa 2.12 nähdään, toimintokaavio sisältää aloitus- ja lopetuspisteen, kuten tilakaaviokin. Kontrollin siirtyminen on kuvattu yksinkertaisilla nuolilla. Toiminnot on kuvattu sivureunoilta pyöristetyissä suorakulmioissa. Kontrolli voi haarautua välillä useaksi eri säikeeksi, jolloin toiminta voi olla rinnakkaista. Nämä rinnakkain tapahtuvat toiminnat voivat taas toisaalla yhtyä eli kummankin toiminnan täytyy olla suoritettu, ennen kuin seuraavaan toimintoon voidaan siirtyä.

Toimintokaaviot sisältävät usein *päätössymboleja*, joilla voidaan kuvata kontrollin haarautumista. Tällöin ehto, jolla haaraan päästään, annetaan hakasuluissa siirtymän yhteydessä. Sarakkeita voidaan käyttää jakamaan toiminnot ryhmiin. Näin saadaan jaettua vastuu toiminnosta toimijoiden tai olioiden kesken.

Toimintokaavioita käytetään Boochin & al (1999) mukaan tyypillisesti kahdella eri tavalla, työnkuvauksen ja operaation mallintamiseen. Työnkuvauksen mallintamisen yhteydessä toimintokaaviolla kuvataan järjestelmää toimijan näkökulmasta. Sen avulla visualisoidaan, määritellään, rakennetaan ja dokumentoidaan liiketoimintaprosesseja. Operaation mallintamisen yhteydessä toimintokaavioilla kuvataan järjestelmän toiminnan yksityiskohtia.

2.3.8 Komponenttikaavio

Komponenttikaavio kuvaa järjestelmän komponentit ja niiden väliset suhteet. Komponenttikaavioiden sisältö voi olla hyvinkin erilainen. Se voi olla suoritettava, se voi myös sisältää binääri- tai lähdekoodia. On myös mahdollista, että komponentti on esimerkiksi dokumenttitiedosto. Lähtökohta on se, että komponentin on oltava jokin järjestelmästä fyysisesti erotettavissa oleva osa. Booch & al. (1999) määrittävät komponenttikaavion siten, että se on pohjimmiltaan luokkakaavio, joka keskittyy järjestelmän komponentteihin.



Kuva 2.13. Komponenttikaavio.

Kuten kuvassa 2.13 voi nähdä, komponenttikaavion komponenttien suhteita kuvaavat rajapintojen toteutukset ja rajapintojen käyttö. Ensimmäinen kuvataan yhtenäisellä viivalla komponentista rajapintaan. Rajapinta tarjoaa komponentin palvelut saataville. Rajapinnan käyttö kuvataan katkoviivanuolella käyttävästä komponentista rajapintaan. Tällöin komponentti on riippuvainen rajapinnasta.

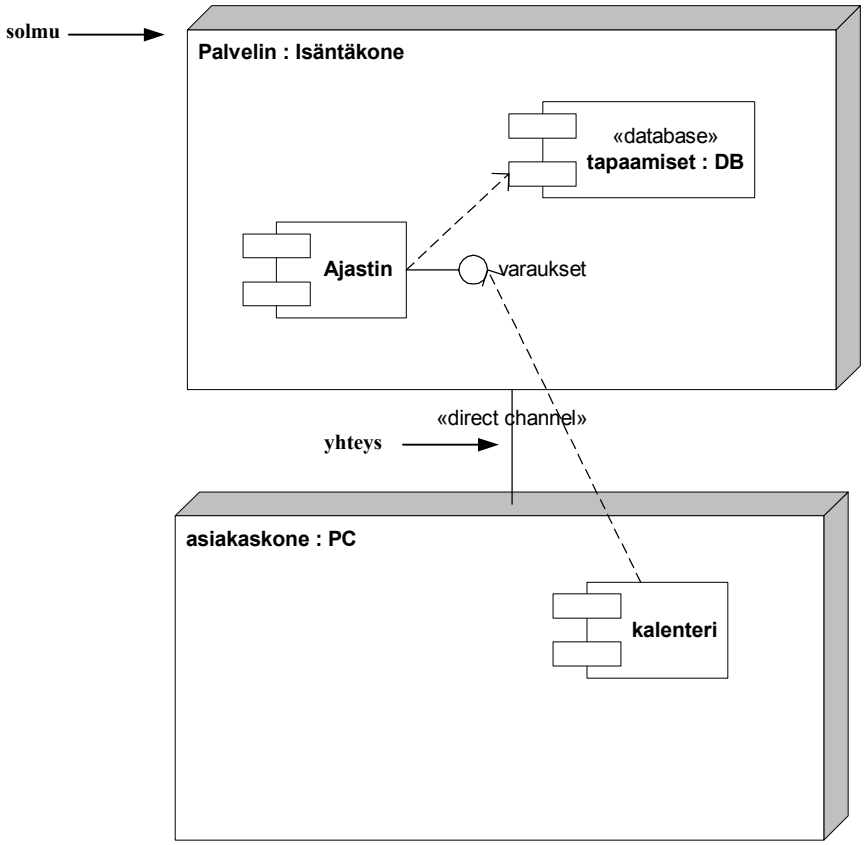
Komponenttikaavioiden käytetään Booch & al. (1999) mukaan neljällä eri tavalla. Niiden avulla mallinnetaan lähdekoodien fyysistä sijoittelua tiedostoihin. Toinen käyttökohde on mallintaa suorituskelpoisia versioita järjestelmästä. Tällöin keskitytään niihin osiin, jotka ovat suorituskelpoisen ohjelman kannalta olennaisia. Kolmanneksi niitä voidaan käyttää fyysisen tietokannan mallintamiseen. Neljäs käyttökohde on erilaisten muuntuvien järjestelmien osien mallintaminen.

2.3.9 Sijoittelukaavio

Sijoittelukaavio kuvaa järjestelmän ajonaikaisia prosessoreja ja laiteita sekä sitä, miten komponentit on sijoiteltu niihin. Se on komponenttikaavioiden ohella fyysinen kuvaus järjestelmästä. Järjestelmä voi olla rakenteeltaan sellainen, ettei sijoittelukaaviolle ole käyttöä ollenkaan. Tällaisessa järjestelmässä ei ole kuin yksi prosessointia suorittava laite.

Sijoittelukaaviolla kuvataan näkymää järjestelmästä staattisesti. Järjestelmän täydelliseen kuvaamiseen tarvitaan useita kaavioita. Tällöin kaaviot kuvaavat järjestelmää erilaisista näkökulmista. Yksi voi keskittyä kuvaamaan komponenttien sijoittelua eri *solmuihin*.

Monikerroksisessa järjestelmässä ne voivat esimerkiksi kuvata sovelluskerroksien sijoittelua, niiden fyysisiä yhteyksiä toisiinsa ja niiden kommunikoinnin loogisia reittejä.



Kuva 2.14. Ajankäytönhallinnan sijoittelukaavio (Jacobson & al., 1999).

Sijoittelukaavion keskeisimmät osat ovat solmu ja assosiaatio, jolla solmut yhdistetään toisiinsa. Sijoittelukaavio on monella tavalla luokkakaavion erikoistapaus, keskittyen järjestelmän solmuihin (Booch & al., 1999). Niinpä solmut ovat kuten luokat, ja sen vuoksi niiden välinen suhde on assosiaatio. Kuvasta 2.14 voi nähdä, että sijoittelukaaviot sisältävät usein myös komponentteja, joiden väliset suhteet on kuvattu normaalilla komponenttikaavioiden notaatiolla. Solmut voivat sisältää myös paketteja ja alijärjestelmiä, joiden yhteydessä käytetään usein stereotyyppejä määrittämään niitä.

Sijoittelukaavioita käytetään Boochin & al. (1999) mukaan kolmeen eri käyttötarkoitukseen. Ensimmäinen näistä on sulautettujen järjestelmien mallinnus. Sijoittelukaaviot tarjoavat hyvän mahdollisuuden mallintaa sulautettuun järjestelmään kuuluvien laitteiden ja niiden sisältämien ohjelmien yhteyksiä ja sijoittumista toisiinsa nähden. Toinen käyttötarkoitus on asiakas/palvelin järjestelmien mallinnus. Siinä sijoittelukaaviolla mallinnetaan verkkoyhteyksiä asiakkaan ja palvelimen välillä ja ohjelmistokomponenttien sijoittelua eri

solmuihin. Kolmanneksi sijoittelukaaviot soveltuvat hajautettujen ympäristöjen mallintamiseen. Yleensä tällaiset järjestelmät sisältävät useita palvelimia, jotka on hajautettu laajalle alueelle.

2.4 Arkkitehtuuri

Ohjelmistoprojektissa on mukana suuri määrä ihmisiä, joilla on eri rooli projektin kannalta. He voivat olla analysoijia, suunnittelijoita, testaajia, käyttäjiä, projektipäälliköitä yms. Erilaisissa rooleissa olevat ihmiset tarvitsevat erilaista tietoa järjestelmästä. Projektipäällikkö ja loppukäyttäjä ovat kiinnostuneita hyvinkin erityyppisistä malleista, joita projektista tarjotaan.

Kaaviot voidaan jakaa ryhmiin niiden tarjoaman tiedon perusteella. Näitä ryhmiä kutsutaan UML:ssä *näkymiksi.* Näkymä kuvaa järjestelmää tietyltä kannalta. Näkymät on jaettu viiteen eri luokkaan. Tätä jakoa kutsutaan ”4+1”-näkymäksi. UML:n mukaan näkymät on nimetty *käyttötapausnäkymäksi* (use case view), *suunnittelunäkymäksi* (design view), *prosessinäkymäksi* (process view), *toteutusnäkymäksi* (implementation view) ja *sijoittelunäkymäksi* (deployment view) (Booch & al., 1999). Eri prosessimalleissa nimeäminen poikkeaa hieman tästä, mutta idea pysyy samana. Näkymät ovat käsitelty tarkemmin ohjelmistoprosesseja kuvaavissa luvuissa, koska ne ovat niissä keskeisessä asemassa.

Ohjelmistoarkkitehtuurilla on rooli ohjelmiston korkean tason suunnittelussa ja toteutuksessa. Se on tulos, kun yhdistetään tietty määrä arkkitehtuurisia elementtejä hyvin valitussa muodossa, jotta saadaan kuvattua järjestelmän tärkeimmät toiminnalliset ja suorituskyvylliset vaatimukset. Tämän lisäksi täytyy kuvata myös ei-toiminnalliset vaatimukset, kuten luotettavuus, skaalautuvuus, siirrettävyys ja saatavuus (Kruchten, 1995).

3 UNIFIED PROCESS JA RATIONAL UNIFIED PROCESS

UML kanssa rinnan on kehitetty saman kolmikon (Booch, Jacobson, Rumbaugh) toimesta myös prosessimalli, vaikkakin UML:ää pystytään kuitenkin käyttämään yhtä hyvin minkä tahansa muun prosessimallin kanssa tai ilman prosessimallia.

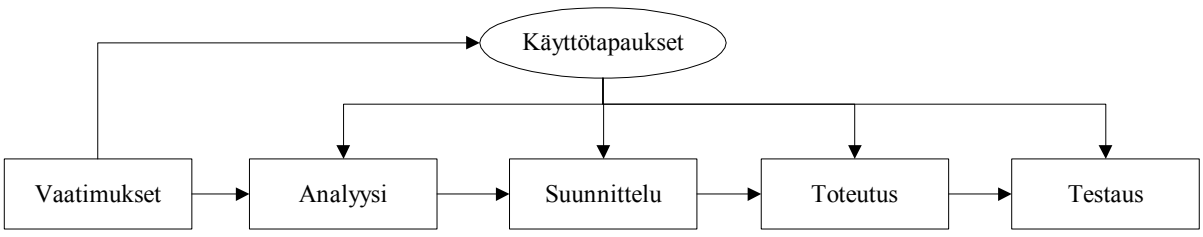
Tässä luvussa kuvataan *Unified Software Development Process* tai lyhemmin *Unified Processin* (UP) ja *Rational Unified Processin* (RUP) pääpiirteet ja keskeiset ajatukset lyhyesti. Käsittely tehdään yhdessä, koska RUP on geneerisemmin prosessia esittävän UP:n osajoukko, eikä niitä näin ollen ole ollut syytä erotella toisistaan.

UP/RUP on käyttötapauslähtöinen ja arkkitehtuurikeskeinen ohjelmiston kehitysprosessi. Sen keskeisiin ajatuksiin kuuluu myös se, että ohjelmiston kehitys on iteratiivista ja lisäävää. Prosessin tarjoamat termit eivät sinänsä ole uusia ohjelmistokehityksessä. Käytännössä se perustuu Booch:n OMT ja OOSE/Objectory menetelmien ajatuksiin. Iteratiivinen ja lisäävä ohjelmistoprosessi ovat myös olleet ajatuksena esillä jo 80-luvulla (Graham, 1989).

3.1 Käyttötapauslähtöinen järjestelmä

Käyttötapaukset ovat yksi UP:n peruskäsitteistä. Käyttötapausten avulla kuvataan järjestelmän toiminnalliset vaatimukset. Yksi käyttötapaus kuvaa toimintaa, jossa käyttäjä keskustelee järjestelmän kanssa saavuttaakseen jonkun päämäärän. Yhdessä käyttötapaukset muodostavat *käyttötapausmallin*, joka kuvaa koko järjestelmän toiminnallisuuden.

Käyttötapaukset sisältävät järjestelmältä vaadittujen toimintojen kuvauksia, ja näin ollen ne vaikuttavat kaikkiin prosessin vaiheisiin (kuva 3.1). Analyysivaiheessa niillä kuvataan tarvittava toiminnallisuus, ja ne toimivat apuvälineenä keskustelussa asiakkaan kanssa. Suunnittelu- ja toteutusvaiheen aikana käyttötapaukset realisoidaan toteutettaviksi luokiksi ja paketeiksi. Ne ovat myös testitapausten perusta ja testausvaiheen aikana niiden avulla todennetaan järjestelmän toimivuus.



Kuva 3.1. Käyttötapausten vaikutus prosessin työnkulkuihin.

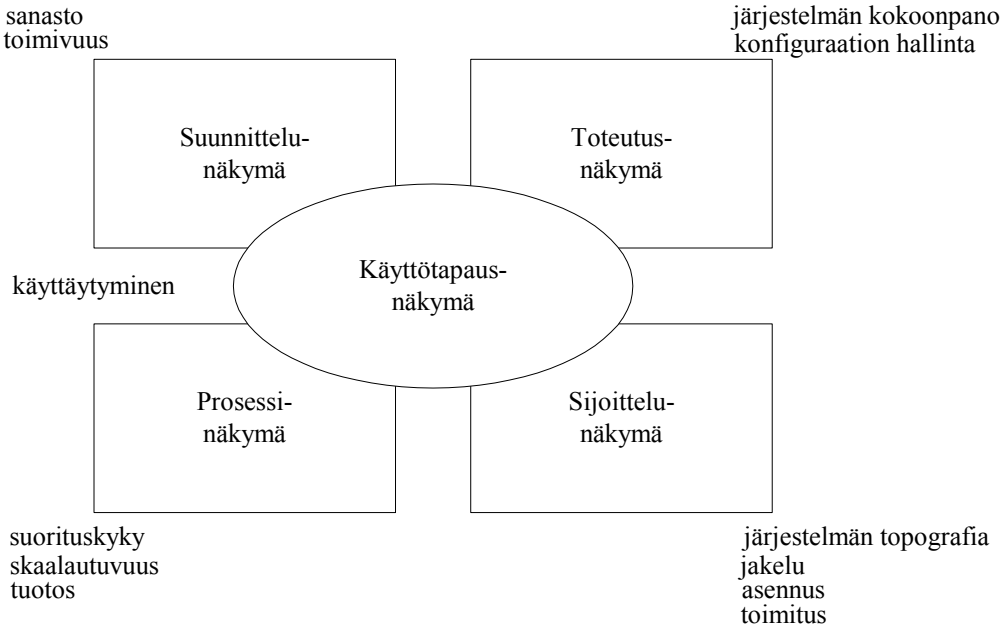
Jacobsonin & al. (1999) mukaan käyttötapauksilla on kaksi päämäärää: löytää todelliset vaatimukset ja esittää ne tarkoituksenmukaisella tavalla käyttäjille, asiakkaille ja suunnittelijoille. Todelliset vaatimukset ovat vaatimuksia, jotka toteutettuina tuottavat odotettua lisäarvoa käyttäjälle. Tarkoituksenmukainen esitystapa tarkoittaa sitä, että käyttäjät ja asiakkaat todella ymmärtävät, mitä suunnittelijat yrittävät kuvata.

3.2 Arkkitehtuurikeskeinen järjestelmä

Toinen peruskäsite prosessimallissa on sen arkkitehtuurikeskeisyys, jonka mukaan perustason järjestelmäarkkitehtuuri on tärkeä ja on muodostettava prosessin alkuvaiheessa. Tätä kutsutaan järjestelmän *perusarkkitehtuuriksi* (baseline architecture). Arkkitehtuurin avulla järjestelmästä kuvataan tärkeimmät staattiset ja dynaamiset aspektit. Vaikka ohjelmistoa voi ajatella kokonaisuutena, on sen rakentamisen kannalta välttämätöntä, että siitä on useita erilaisia näkymiä. Asian tarkasteleminen erilaisista perspektiiveistä helpottaa suunnittelua.

Ohjelmistoarkkitehtuuri kuvaa asioita abstraktilla tasolla. Se identifioi järjestelmän eri osat, näiden väliset suhteet ja vuorovaikutuksen toisiinsa, viestintämekanismit ja osien lisäämisen ja poistamisen yleissäännöt. Ohjelmistoarkkitehtuuri on apuna korkean tason suunnittelussa ja toteutuksessa. Sen tarkoituksena on Kruchtenin (1995) mukaan kuvata järjestelmän tärkeimmät toiminnallisuus- ja suorituskykyvaatimukset, sekä ei-toiminnalliset vaatimukset, kuten luotettavuus, skaalautuvuus, siirrettävyys ja saatavuus.

Arkkitehtoonisen suunnittelun perustan muodostaa UP/RUP:ssa ns. ”4+1” näkymämalli, joka on kuvattu eräässä muodossaan alla olevassa kuvassa. Siihen kuuluvat *käyttötapausnäkymä* (use case view), *toteutusnäkymä* (implementation view), *prosessinäkymä* (process view) ja *sijoittelunäkymä* (deployment view). Nämä näkymät ovat kanssakäymisessä keskenään. Ne voidaan myös erottaa toisistaan, jotta eri prosessissa osallisena oleva voi kiinnittää huomionsa niihin kysymyksiin, jotka ovat tärkeitä hänen kannaltaan (Booch & al., 1999).



Kuva 3.2. ”4+1”-näkymä (Booch & al., 1999).

Booch & al. (1999) ovat kuvanneet kunkin näkymän pääpiirteittäin seuraavalla tavalla:

- Käyttötapausnäkymä käsittää käyttötapaukset, jotka kuvaavat järjestelmän siten, kuin käyttäjät, analysoijat ja testaajat ne näkevät.

- Suunnittelunäkymä käsittää luokat, rajapinnat ja yhteistoiminnan, jotka muodostavat ongelman sanaston ja sen ratkaisun. Tämän näkymän pääasiallinen tarkoitus on tukea järjestelmän toiminnallisia vaatimuksia.

- Prosessinäkymä käsittää säikeet ja prosessit, jotka muodostavat järjestelmän samanaikaisuus- ja synkronisointimekanismit. Näkymä kuvaa pääasiassa suorituskykyä, skaalautuvuutta ja tuotosta.

- Toteutusnäkymä käsittää komponentit ja tiedostot, joista muodostetaan lopullinen versio. Näkymä mallintaa pääasiassa konfiguroinnin hallintaa järjestelmän eri versioissa.

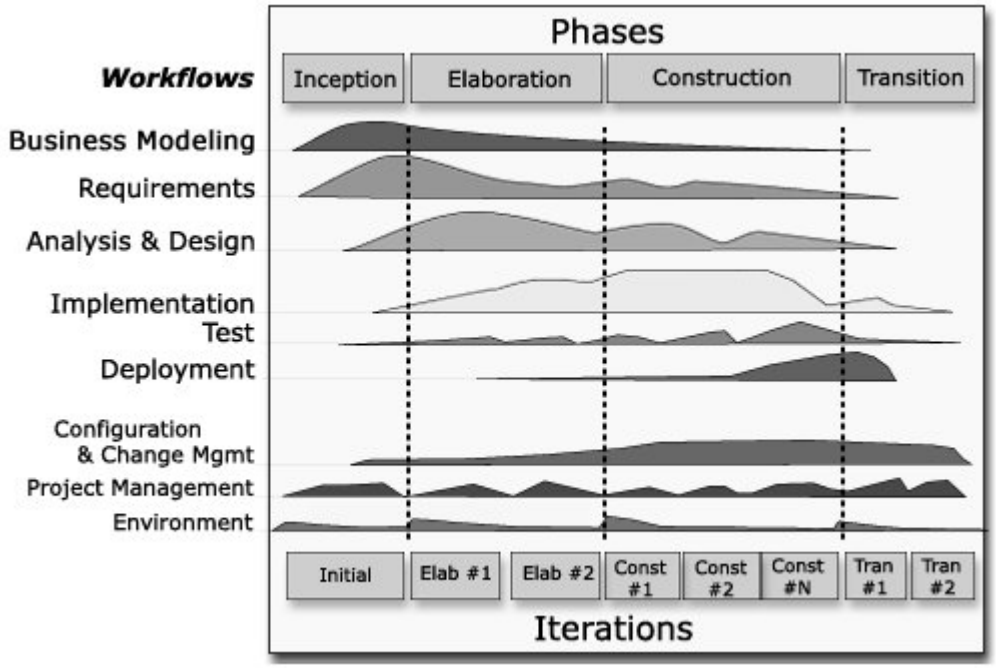
- Sijoittelunäkymä käsittää solmut, jotka muodostavat järjestelmän laitetopologian ja joissa järjestelmä suoritetaan. Näkymä kuvaa pääasiassa jakelua, toimitusta ja asennusta fyysisen systeemin osalta.

3.3 Iteroiva ja lisäävä prosessi

Kolmas peruskäsite UP:ssa on iteroiva ja lisäävä prosessi. Iteroiva ohjelmistoprosessi tarkoittaa sellaista prosessia, jossa eri *työnkulut* (workflow) käydään läpi useasti (kuva 3.3). Jokaisessa iteraatiosta valmistuu versio, joka kehittyy iteraatio iteraatiolta valmiiksi ohjelmistoksi. Samalla lailla kaikki mallit ja kaaviot kehittyvät pienin askelin.

Iteroiva kehittäminen tarjoaa strategian kehittää ohjelmistoja pienissä, helpommin hallittavissa olevissa palasissa. Iterointi myös elää prosessin aikana. Aikaisissa *vaiheissa* (phases) (kuva 3.3) iteraatioissa keskitytään löytämään ohjelmiston rajat, poistamaan kriittisiä riskejä ja löytämään arkkitehtuurin peruslinja. Kun iteraatioita jatketaan läpi projektin eli vähennetään asteittain jäljellä olevaa riskiä ja implementoidaan komponentteja, iteraation muoto muuttuu enemmän järjestelmän kasvua lisääväksi (Jacobson & al., 1999).

UP/RUP:ssa puhutaan ohjelmiston elinkaaresta, joka on jaettu sykleihin. Elinkaari kuvaa koko ohjelmiston elämän syntymästä kehitystyön päättymiseen. Syklit on jaettu neljään eri vaiheeseen, jotka ovat *aloitus* (inception), *kehittely* (elaboration), *rakennus* (construction) ja s*iirtymä* (transition). Syklin sisällä tapahtuvat työnkulut, jotka ovat UP:ssa *vaatimusten määrittely* (requirements), *analyysi* (analysis), *suunnittelu* (design), *toteutus* (implementation) ja *testaus* (test). RUP:ssa ne on jaoteltu hieman eri lailla, ja siinä ne ovat *liiketoiminnan mallintaminen* (business modeling), *vaatimusten määrittely* (requirements), *analyysi & suunnittelu* (analysis & design), *toteutus* (implementation), *testaus* (test), *sijoittelu* (deployment), *konfigurointi ja muutosten hallinta* (configuration & change management), *projektin hallinta* (project management) ja *ympäristö* (environment) (kuva 3.3).



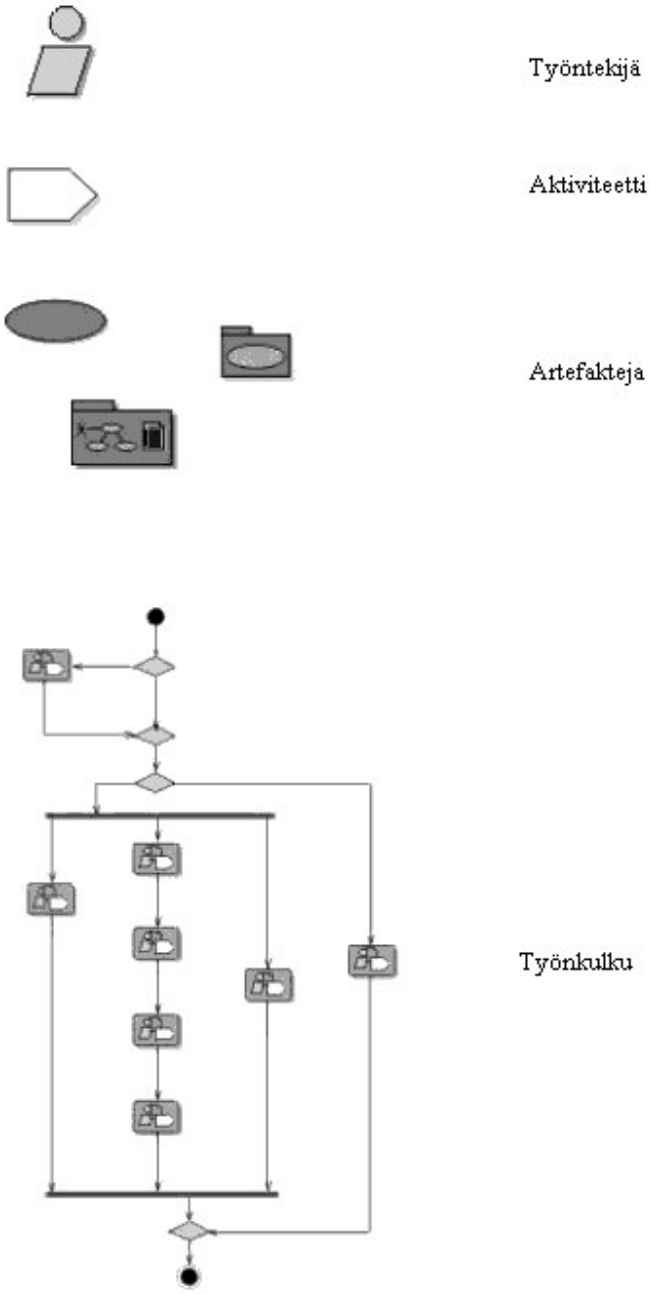
Kuva 3.3. Ohjelmistokehityksen elinkaari (Rational, 2001a)

Jokaisen iteraation on tarkoitus antaa lisäarvoa järjestelmälle. Tämä lisäys on yksi askel järjestelmän kehityksessä, ja sitä kutsutaan *toimitukseksi* (release). Iteraatiot on suunniteltava etukäteen ja jokaiselle niistä on asetettava etukäteen tavoitteet. Asetettuja tavoitteita kutsutaan *merkkipaaluiksi* (milestone) ts. merkkipaalu on saavutettu, kun sille asetetut tavoitteet on saavutettu.

Grahamin (1989) mukaan inkrementoiva ohjelmistokehitys on suunta hieman ”pehmeämpään” ohjelmistokehitykseen. Tämä aiheuttaa kuitenkin huomattavasti lisävaatimuksia projektin hallinnan kannalta. Suunnittelua on tehtävä jokaisessa vaiheessa huomattavasti enemmän. Myös iteratiivinen prosessi lisää projektinhallintaan käytettävää työmäärää (Kruchten, 2000), mutta se auttaa hallitsemaan paremmin monimutkaisten järjestelmien kehittämistä.

3.4 RUP:n terminologia

RUP:n mukaan prosessi kuvaa *kuka* tekee *mitä*, *miten* ja *milloin.* RUP:n kuuluu neljä elementtiä, jotka ovat työntekijät (workers) ⇒ kuka, aktiviteetit (activities) ⇒ miten, artefaktit (artifacts) ⇒ mitä ja työnkulku (workflows) ⇒ milloin. Kuvassa 3.4 on kuvattu näiden elementtien notaatioita RUP:ssa.



Kuva 3.4. RUP:n elementit.

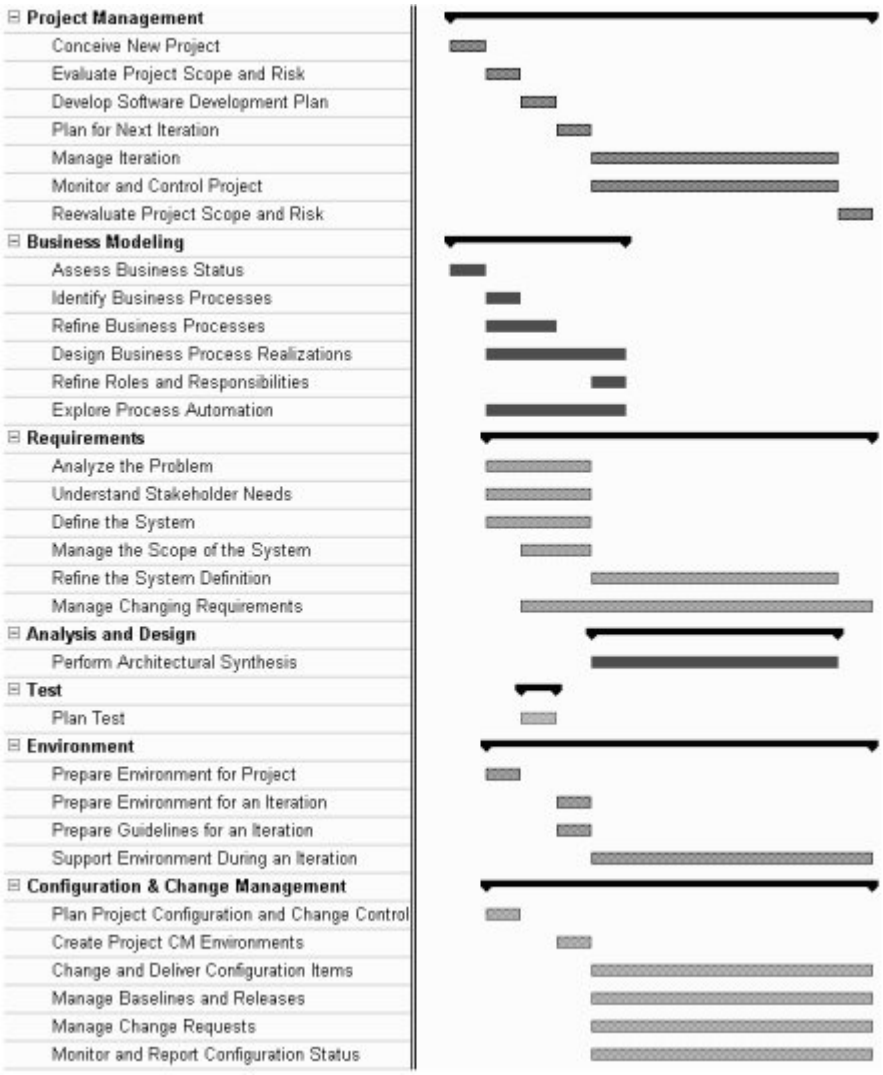
Työntekijällä ja aktiviteetilla on standardi kuvaustapa, mutta artefakteja on kuvattu usealla erilaisessa symbolilla. Työnkulku sisältää työntekijöitä ja aktiviteetteja yhdistettynä toisiinsa nuolilla kuvaamaan työvaiheiden etenemistä.

3.5 Prosessin vaiheet

Vaiheet ovat RUP:ssa keskeisessä asemassa, kun se kuvaa ohjelman elinkaarta. Rational (2001a) määrittelee vaiheen siten, että se kahden merkkipaalun välinen aika, jolloin määritelty

osa tavoitteista on saatu toteutettua, artefaktit on saatu valmiiksi ja päätös siitä, siirrytäänkö seuraavaan vaiheeseen vai ei, on saatu tehtyä.

RUP:ssa vaiheet on kuvattu siten, että jokaisesta annetaan kuvaus sen päämääristä, välttämättömät toiminnot vaiheen suorittamiseksi, merkkipaalu, joka suoritetaan vaiheen lopuksi ja sen kuvaus sekä esimerkinomainen iterointisuunnitelma vaiheelle. Kuvassa 3.5 on kuvattu aloitusvaiheen iterointisuunnitelma.



Kuva 3.5. Aloitus-vaiheen iterointisuunnitelma (RUP, 2001).

Seuraavissa luvuissa on annettu lyhyt kuvaus RUP:n vaiheiden sisällöstä.

3.5.1 Aloitusvaihe

Aloitusvaiheen aikana keskitytään määrittämään projektin laajuus, kuvaamaan pahimpia riskejä projektin kannalta, kuvailemaan alustavia liiketoimintamalleja ja miettimään

liiketoiminnan kannalta, kannattaako koko projektia aloittaa. Myös arviot hinnasta, aikataulusta ja tuotosta tehdään aloitusvaiheen aikana, ennen kuin päätetään siitä, aloitetaanko projekti. Projektille kerätään ja muodostetaan perustavat tiedot tulevan järjestelmän arkkitehtuurista ja sovellusalueesta.

3.5.2 Kehittelyvaihe

Kehittelyvaiheen tehtäviin kuuluu määritellä projektin perusarkkitehtuuri ja toiminnallisuus. Suurin osa vaatimuksista on tässä vaiheessa selvillä. Rakennettava järjestelmä analysoidaan ja suunnitellaan. Toteutettavasta järjestelmästä voidaan tehdä useita prototyyppejä. Myös toteutusympäristö kootaan kehittelyvaiheen aikana.

3.5.3 Rakennusvaihe

Rakennusvaiheessa järjestelmä toteutetaan määritellyn perusarkkitehtuurin perusteella. Järjestelmä toteutetaan iteroiden läpi analyysin, suunnittelun, toteutuksen ja testauksen. Rakennusvaiheen lopuksi tuotettava järjestelmä on siinä kunnossa, että se on valmis beta-testaukseen.

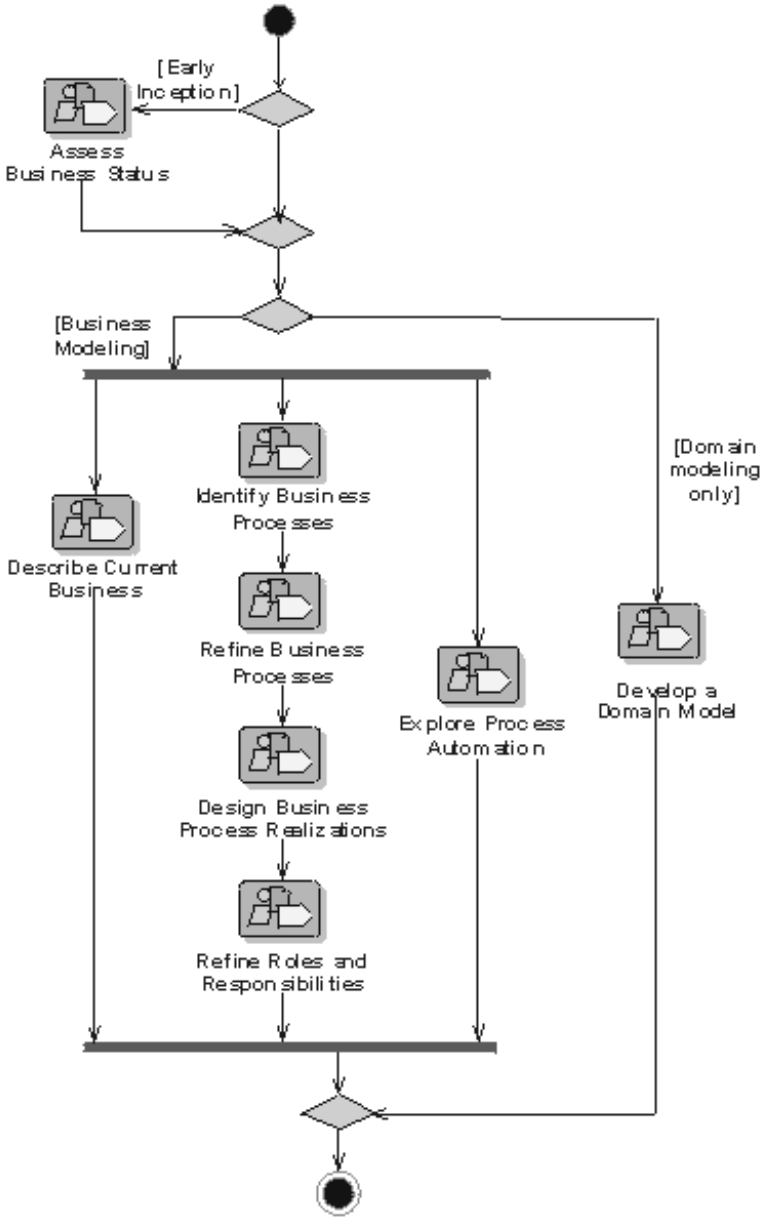
3.5.4 Siirtymävaihe

Siirtymävaiheessa järjestelmä laitetaan toimituskuntoon, jotta se voidaan toimittaa asiakkaalle dokumentteineen ja koulutuksineen. Järjestelmä asennetaan ja otetaan käyttöön tuotantoympäristössä. Se myös versioidaan siirtymävaiheen aikana. Tällöin ei enää tehdä mitään suuria muutoksia vaan pelkästään pientä hienosäätöä käyttäjäkokemusten perusteella.

3.6 Prosessin työnkulut

Työnkulut kuvaavat ohjelmistoprosessin etenemistä tutumpien, monessa aikaisemmassakin prosessissa olleiden ohjelmistoprosessin termein. Rational (2001a) määrittelee työnkulun siten, että ne esittävät työntekijöiden ja aktiviteettien jakamista loogisiin ryhmiin kiinnostusalueiden tai alojen mukaan.

RUP:ssa työnkulut on kuvattu siten, että jokainen niistä sisältää esittelyn, kaaviokuvauksen, siihen kuuluvat aktiviteetit, artefaktit, suoritusohjeet ja työnkulkuun liittyvien käsitteiden kuvauksen. Kuvassa 3.6 on esimerkki tästä kuvauksesta. Siitä voi huomata, että RUP:ssa työnkulun kuvaus on havainnollistettu UML toimintokaaviota mukailevalla notaatiolla.



Kuva 3.6. Liiketoiminnan mallintaminen työnkulun kuvaus (Rational, 2001a).

3.6.1 Liiketoiminnan mallintaminen

Työnkulun aikana selvitetään, mitä ovat kohdeorganisaation liiketoimintaprosessit ja organisaatiorakenne. Tarkoituksena on ymmärtää ongelmat ja löytää niihin parannusehdotuksia. Tärkeää on myös se, että kaikki projektiin osallistujat, niin ohjelman tulevat käyttäjät kuin suunnittelijat ja projektipäällikötkin, ymmärtävät, miten kohde-organisaatio toimii.

3.6.2 Vaatimusten määrittely

Työnkulun tarkoituksena on pyrkimys siihen, että kaikki projektiin osallistujat ovat yhtä mieltä siitä, mitä järjestelmän tulisi tehdä. Järjestelmän kehittäjät saavat tietoa vaatimuksista, joita järjestelmälle asetetaan. Samalla saadaan käsitys siitä, mikä kuuluu järjestelmään ja mikä ei. Työnkulun aikana saadaan myös käsitystä siitä, miten iterointeja tulisi suorittaa ja pystytään arvioimaan projektin kustannuksia ja siihen tarvittavaa aikaa ja resursseja.

3.6.3 Analyysi & suunnittelu

Analyysi & suunnittelu –työnkulku kuvaavat, miten järjestelmä toimii. Järjestelmän tulee täyttää kaikki käyttötapauksissa määritellyt tehtävät ja vaatimusmäärittelyssä esitetyt vaatimukset. Työnkulku toimii suunnitelmana toteutukselle ja sen pohjalta pystytään toteuttamaan sen kuvaama järjestelmän toteutus–työnkulussa.

3.6.4 Toteutus

Toteutus-työnkulun aikana järjestelmä toteutetaan komponenteiksi. Nämä komponentit myös yksikkötestataan vaiheen aikana. Uuden järjestelmän toteutuksessa käytetään hyväksi aikaisemmin suunniteltuja komponentteja. Uudet komponentit suunnitellaan ajatellen niiden mahdollista uudelleenkäytettävyyttä tulevissa projekteissa.

3.6.5 Testaus

Työnkulun aikana testataan, että oliot kommunikoivat oikein keskenään ja että komponentit toimivat oikein yhdessä. Testauksen aikana tarkastetaan, että kaikki järjestelmälle asetetut vaatimukset on oikein toteutettu. Rational:n (2001a) mukaan testeillä on kolme laadullista ulottuvuutta, jotka ovat luotettavuus, toiminnallisuus sekä ohjelman ja järjestelmän suorituskyky.

3.6.6 Sijoittelu

Työnkulun aikana tuotetaan versio tuotteesta ja toimitetaan järjestelmä sen loppukäyttäjille. Versio pakataan ja jaellaan asiakkaille. Se voidaan tarvittaessa myös asentaa tuotantoympäristöön. Loppukäyttäjille tarjotaan apua järjestelmän käyttöönotossa. Sijoittelu-työnkulkuun voi kuulua myös beta-testauksen tekeminen ja muodollinen hyväksymistestaus.

3.6.7 Konfigurointi ja muutosten hallinta

Varsinaisten työnkulkujen lisäksi RUP sisältää työnkulkuja, jotka tukevat ohjelmiston kehitystä (supporting workflows). Konfigurointi ja muutosten hallinta on yksi näistä työnkuluista. Se kuvaa toiminnot, joiden avulla artefaktit saadaan pysymään järjestyksessä. Työnkulku varmistaa, ettei kukaan ei tee artefakteihin päällekkäisiä päivityksiä ja kaikki ovat tietoisia niihin tulleista muutoksista.

3.6.8 Projektin hallinta

Projektin hallinta -työnkulku kuuluu myös kehitystä tukeviin työnkulkuihin. Sen tarkoituksena on auttaa hallitsemaan riskejä, jotta ohjelmisto voitaisiin toimittaa onnistuneesti asiakkaalle. Se keskittyy hallintaan pääasiassa ohjelmiston näkökannasta ja ei ota kantaa sellaisiin asioihin kuten työntekijöiden hallinta, budjetti tai sopimukset.

3.6.9 Ympäristö

Ympäristö–työnkulku on viimeinen kehitystä tukevista työnkuluista. Sen tarkoituksena on varmistaa, että kehitysorganisaatiolla on kehitysympäristö prosesseineen ja työkaluineen, jotta se voisi toteuttaa projektin onnistuneesti. Se kuvaa myös tehtävät, jotka on suoritettava projektin ohjeistuksen suunnittelussa. Ympäristö–työnkulkuun kuuluu myös mahdollisesti tarvittava prosessin muokkaus.

3.7 UML käyttö UP/RUP:ssa

Prosessimalli sisältää artefakteja, jotka voivat olla muodostuneita malleista. Nämä mallit puolestaan voivat sisältävät erilaisia UML-kaavioita. Seuraavassa on tutkittu kaavio kerrallaan, missä yhteydessä ja millaisessa muodossa kaaviot esiintyvät prosessin mukaan.

3.7.1 Käyttötapauskaavio

Käyttötapaukset on jaettu kahteen eri ryhmään. Ne ovat *liiketoimintakäyttötapaukset* (business use cases) ja *käyttötapaukset* (use cases). Molemmat käyttötapauskaaviot kuuluvat käyttötapausmalliin, eli on *liiketoimintakäyttötapausmalli* (business use case model) ja *käyttötapausmalli* (use case model). Eri mallit mallintavat järjestelmää hiukan eri näkökulmasta. Molemmat keskittyvät vaihejaon mukaan pääasiassa aloitusvaiheen ajalle.

Liiketoimintakäyttötapausmalli kuvaa pääasiassa sitä, mitkä aktiviteetit koskevat asiakasta. Se kuvaa liiketoimintaan kuuluvia tehtäviä. Sitä käytetään apuna, kun tunnistetaan rooleja ja

toimitettavia tuotteita tai palveluita organisaatiossa. Se kuuluu liiketoiminnan mallintaminen – työnkulkuun.

Käyttötapausmalli kuvaa puolestaan järjestelmän toimintoja ja sen ympäristöä. Se toimii eräänlaisena sopimuksena asiakkaan ja suunnittelijoiden välillä siitä, mitä toimintoja järjestelmään kuuluu ja mitä järjestelmän tulee tehdä. Käyttötapausmalli on tärkeänä apuna, kun tehdään analysointia, suunnittelua ja testausta.

Käyttötapauskaavioita käytetään lisäksi *käyttötapausten toteutuksen* (use case realization) kuvaukseen. Siinä kuvataan, mikä käyttötapauksen toteutus toteuttaa käyttötapauksen. Tämä tehdään analyysi & suunnittelu -työnkulun aikana ja on nimetty *analyysivaiheen käyttötapausten toteuttamiseksi* (use case realization-analyze) ja *suunnitteluvaiheen käyttötapausten toteuttamiseksi* (use case realization-design). Vaihejaon mukaan tämä sijoittuu kehittely- ja rakennusvaiheisiin.

3.7.2 Luokkakaavio

Prosessimalli sisältää kolmenlaisia luokkakaaviota. Ne ovat *liiketoimintaluokkakaaviot*, *analyysiluokkakaaviot* ja *suunnitteluluokkakaaviot*. Näistä liiketoiminta- ja analyysiluokkakaaviot kuvataan UML:n laajennusmekanismeja käyttäen. Näitä on kuvattu luvussa 2.3.2.

Jokainen luokkakaavioista kuuluu *oliomalliin* (object-model) tai *malliin* (model). Liiketoimintaluokkakaaviot kuuluvat *liiketoimintaoliomalliin* (business object model), analyysiluokkakaaviot kuuluvat *analyysimalliin* (analysis model) ja suunnitteluluokkakaaviot kuuluvat *suunnittelumalliin* (design model). Kaikki näistä ovat nimeämisestä huolimatta käsitteellisesti olio-malleja. Oliomalliin kuuluvat luokkadiagrammi, yhteistyökaavio, viestiyhteyskaavio, tilakaavio ja toimintokaavio.

Liiketoiminta-oliomalli kuvaa, miten liiketoimintakäyttötapaukset toteutetaan. Analyysiluokkamalli kuvaa sitä, miten käyttötapaukset toteutetaan. Se toimii pohjana suunnittelumallille. Analyysimalli voidaan jättää myös tekemättä. Suunnittelumalli kuvaa sitä, miten käyttötapaukset todennetaan. Se toimii puolestaan pohjana *toteutusmallille* (implementation model).

Mallin vaihejaon mukaisesti liiketoiminta-oliomalli luodaan aloitusvaiheen aikana ja sen kehittäminen jatkuu kehittelyvaiheessa. Analyysimalli luodaan puolestaan kehittelyvaiheessa

ja täydennetään rakennusvaiheessa. Suunnittelumallin voidaan katsoa kehitettävän aloitus-, kehittely- ja rakennusvaiheen aikana.

Työnkulkujen mukaan tarkasteltuna luokkakaavioita käytetään varsinaisesti kahden työnkulun aikana, jotka ovat liiketoiminnan mallintaminen ja analyysi & suunnittelu. Myös vaatimusten määrittely–työnkulun aikana mallinnetaan luokkia (raja–stereotyypillä varustetut luokat, joiden tehtävänä on kuvata toimijan ja järjestelmän kanssakäymistä), mutta tällöin keskitytään pelkästään luokkien löytämiseen luokkakaavioiden tekemisen sijasta.

3.7.3 Tilakaavio

Tilakaavio kuuluu oliomalliin. Tilakaavioita käytetään liiketoiminta-oliomallissa kuvaamaan sitä, mitä tiloja sen eri osatekijöillä voi olla. Se kuvaa tapahtumat, jotka aiheuttavat tilasiirtymät. Tilakaavioiden pääasiallinen tehtävä on käyttötapausten toteuttamisen ja suunnitteluluokkien kehittämisen apuna. Tilakaaviota käytetään pääasiassa analyysi & suunnittelu-työnkulun aikana ja kehittelyvaiheen aikana.

3.7.4 Viestiyhteyskaavio

Myös viestiyhteyskaavio on osa oliomallia. Viestiyhteyskaaviota käytetään liiketoiminnan mallintamisessa liiketoiminta-oliomallissa ja suunnitteluluokkien mallintamisessa. Ne toimivat myös apuna integrointitestitapausten muodostamisessa. Sen käyttö sijoittuu siten kehittely- ja rakennus-vaiheiden ajalle vaihejaon mukaan ja liiketoiminnan mallintaminen-, analyysi & suunnittelu- ja testaus -työnkulkujen ajalle työnkulkujaon mukaan.

3.7.5 Yhteistyökaavio

Yhteistyökaavio on myös oliomallin osa. Yhteistyökaaviota käytetään samoin kuin viestiyhteyskaaviota eli liiketoiminnan mallintamisessa liiketoiminto-oliomallissa ja suunnitteluluokkien mallintamisessa. Myös niitä voidaan käyttää apuna integrointitestitapausten muodostamisessa. Niiden käyttö sijoittuu kehittely- ja rakennus-vaiheiden ajalle vaihejaon mukaan ja liiketoiminnan mallintaminen ja analyysi & suunnittelu- ja testaus–työnkulkujen ajalle työnkulkujaon mukaan.

3.7.6 Toimintokaavio

Toimintokaavio on myös osa oliomallia. Toimintokaavioita käytetään mallinnettaessa liiketoimintakäyttötapausten työnkulkua. Toimintokaavio kuvaa tehtävät tai toiminnot, jotka saavat aikaan tietyn halutun tuloksen. Niiden käyttö sijoittuu kehittely- ja rakennus-vaiheiden

ajalle vaihejaon mukaan ja liiketoiminnan mallintaminen ja analyysi & suunnittelu ja testaus–työnkulkujen ajalle työnkulkujaon mukaan.

3.7.7 Komponenttikaavio

Komponenttikaaviota käytetään esitettäessä, miten informaatio on sijoitettu ohjelmakoodissa komponentteihin tai miten informaatio on jaettu erillisiin tiedostoihin. Niitä käytetään myös ns. *testikomponenteissa*, jotka helpottavat tai automatisoivat testien suorittamista. Komponenttikaavioiden käyttö sijoittuu rakennus-vaiheen ajalle vaihejaon mukaan ja toteutus- sekä testaus-työkulun ajalle työnkulkujaon mukaan.

3.7.8 Sijoittelukaavio

Sijoittelukaavio kuvaa järjestelmän fyysisen jakautumisen prosesseja sisältäviin solmuihin. Ne kehittyvät iteraatioiden kuluessa. Sijoittelukaavioiden käyttö ajoittuu rakennusvaiheen ajalle vaihejaon mukaan ja analyysi ja suunnittelu-työnkulun ajalle työnkulkujaon mukaan.

4 GSMS

GSMS (Global Solution Management System) on joukko prosessimalleja tietotekniikkaprojektin toteutukseen. Se on toteutettu Electronic Data Systemsin (EDS) toimesta. GSMS on EDS:n sisällä käytössä oleva globaali prosessijärjestelmä. Sen perustana ovat yrityksen vanhat menetelmät ja työvälineet kuten myös kokemukset nykyisistä projekteista (EDS, 2001).

GSMS on luokiteltu Capability Maturity Model (CMM) -kypsyystasomallin mukaan. CMM on Software Engineering Instituten (SEI) kehittämä malli, jonka avulla voidaan mitata ja luokitella ohjelmistokehityksen kypsyystaso. Se sisältää viisi eri tasoa (Curtis ja Paulk, 1993). Jokainen taso sisältää *avainprosessialueen* (key process area, KPA). Ne ovat vaatimuksia, jotka on täytettävä tason saavuttaakseen. GSMS-prosessimalli täyttää toisen ja kolmannen tason vaatimukset (EDS, 2001).

Tämän luvun on tarkoitus esitellä GSMS-prosessimallia ja kuvata tarkemmin sen työvaiheita siihen kuuluvan Object and Component Engineering -*työtyypin* (OCE) mukaan. Lisäksi tarkastellaan, missä vaiheessa eri UML-kaavioita käytetään prosessin aikana.

4.1 Työtyypit

GSMS:n sisällä prosessit on jaettu ns. työtyyppeihin (Work Type) ja organisaatiotasoisiin prosesseihin (Organizational Process). Työtyyppejä on neljä erilaista ja ne ovat Development+, joka keskittyy perinteisempään ohjelmistoprosessiin, Object and Component Engineering, joka on suunnattu olio- ja komponenttipohjaisen ohjelmistoprosessin apuvälineeksi, tuotannon tuki (Production Support) ja projektin hallinta (Project Management). Organisaatiotasoisia prosesseja on myös kolme ja ne ovat prosessin hallinta (Process Management), laadun varmistus (Quality Assurance) ja koulutus (Training).

4.2 Olio- ja komponenttitekniikka

Olio- ja komponenttitekniikka (Object and Component Engineering, OCE) –työtyyppi on olio- ja komponenttipohjaiseen järjestelmänkehitykseen tarkoitettu prosessimalli. Työtyyppi sisältää ohjeiston, mitä työvaiheita ohjelmiston elinkaaren eri vaiheisiin kuuluu, tehtäviin kuuluvat henkilöt ja mitä tuotteita eri vaiheet tuottavat. Lisäksi järjestelmä sisältää mallipohjia dokumentoinnin tueksi, lähdeviitteitä ja apuohjelmia. OCE on työtyyppinä uusi ja sitä kehitetään vielä eteenpäin.

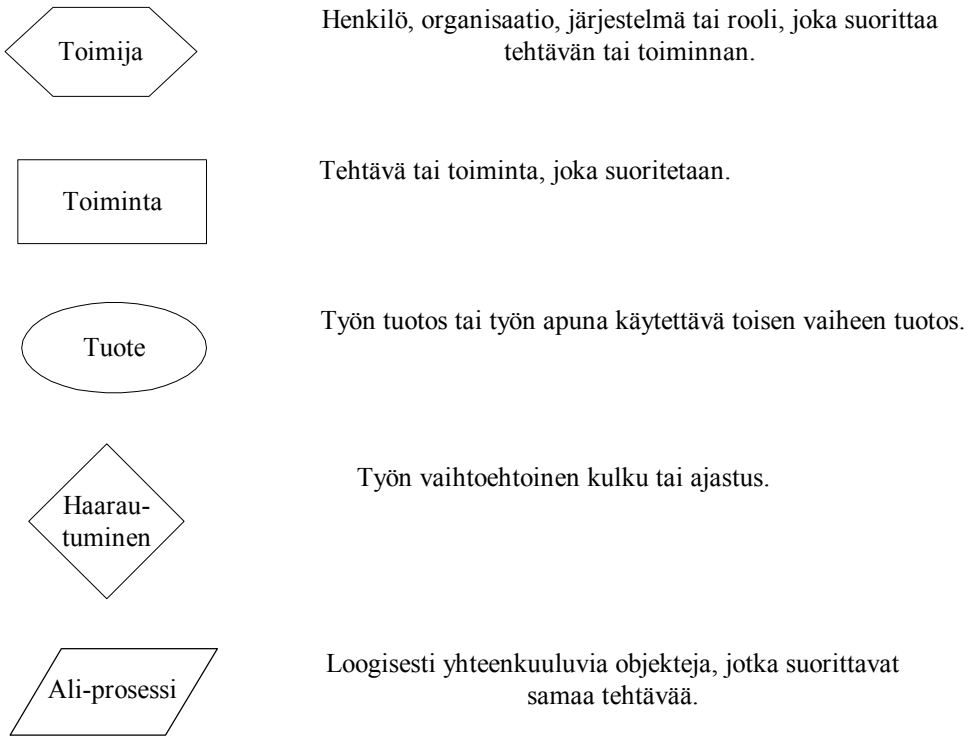
Malli on kuvattu EDS:n intranetiin Process Sourcerer -välineellä, joka mahdollistaa sen, että prosessimalli on kokonaisuudessaan HTML-muodossa. HTML:n mahdollistaa sen, että mallissa voidaan liikkua usealla eri hierarkiatasolla. Elementit voivat olla linkkejä syvemmälle tasolle järjestelmään. Visuaalisen esityksen lisäksi kaikki eri elementit ja tapahtumat on kuvattu yksityiskohtaisesti.

Aliprosessien yhteydessä on kuvattu onnistumisen kannalta kriittiset asiat, prosessikuvaus, työohjeet prosessin suorittamiseksi, selvitys siitä, miksi kyseinen vaihe tehdään ja aiheeseen mahdollisesti liittyvää muuta tietoa. Samasta yhteydestä voi löytyä linkkejä esimerkkimateriaaliin, kuinka sama prosessi on toteutettu esimerkkiprojekteissa sekä aiheeseen liittyvää muuta tietoa, kuten linkkejä työvälineisiin, tekniikoiden esittelyyn, lähdemateriaaliin tai harjoituksiin. Linkkejä löytyy myös standardeihin ja yleisiin käytäntöihin, jotka kuuluvat prosessin yhteyteen.

Hierarkkinen järjestelmä mahdollistaa sen, että ylemmällä tasolla prosessia on kuvattu korkeammalla tasolla ja syvemmälle mentäessä kuvaus tarkentuu. Kolmannella tasolla ja tätä syvemmällä prosessin kuvaus alkaa esittää yksityiskohtaisempia ohjeita kunkin työvaiheen suorittamisesta.

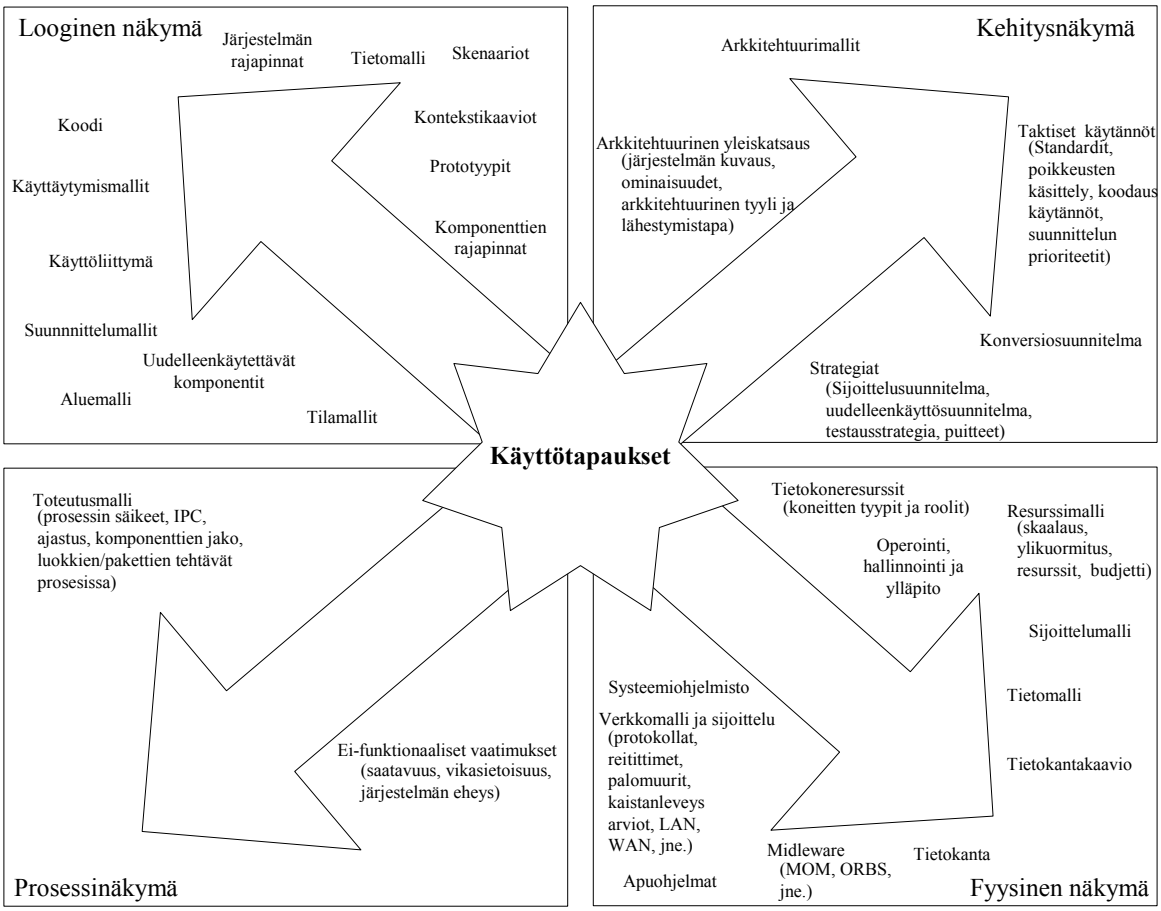
4.3 OCE-mallin terminologia ja keskeiset käsitteet

OCE:n niin kuin kaikkiin muihinkin työtyyppeihin kuuluu viisi erilaista elementtiä. Ne on kuvattu kuvassa 4.1.



Kuva 4.1. Työtyyppien elementit (EDS, 2001).

OCE mallissa keskeisessä asemassa on arkkitehtuuri ja erityisesti ”4+1”-malli. Tämä on kuvattu kuvassa 4.2, josta voi huomata, että malli on kuvattu hieman laajemmin kuin UP/RUP:ssa. Projektissa luodaan määrittelyvaiheen jälkeen malli, jota kehitetään koko prosessin ajan valmiiksi tuotteeksi.



Kuva 4.2. OCE:n ”4+1”-malli (EDS, 2001).

Käyttötapausten tarkoitus on auttaa tunnistamaan järjestelmän rakennetta ja validoimaan sen toimintaa. Ne ovat järjestelmän suunnittelun kannalta keskeisessä asemassa koko projektin läpi ja ne muodostavat eräänlaisen sateenvarjon muiden näkymien yläpuolelle. Käyttötapaukset varmentavat, että kaikki muut näkymät toimivat yhdessä. On tärkeää, että käyttötapauksilla kuvataan ainoastaan tärkeimmät toiminnot, koska muuten niitä tulisi liian suuri määrä. Ne toimivat lisäksi lähtökohtana testaussuunnitelman ja testitapausten tekemiselle.

Looginen näkymä kuvaa kuinka prosessissa toteutetaan toiminnallisen määrittelyn mukaiset toiminnot. Sen tarkoituksena on osoittaa kuinka arkkitehtuuri tukee liiketoimintavaatimuksia. Sen tulee myös näyttää, kuinka oliot ryhmittyvät ongelma-alueella pääabstraktiotasoille toteutuksen vaatimien mekanismien kanssa. Loogisen näkymän tulee tarjota järjestelmästä korkean tason näkymä tarkemman tason kuvauksien kanssa menemättä liikaa yksityiskohtiin, jotka voisivat sotkea kokonaiskuvan.

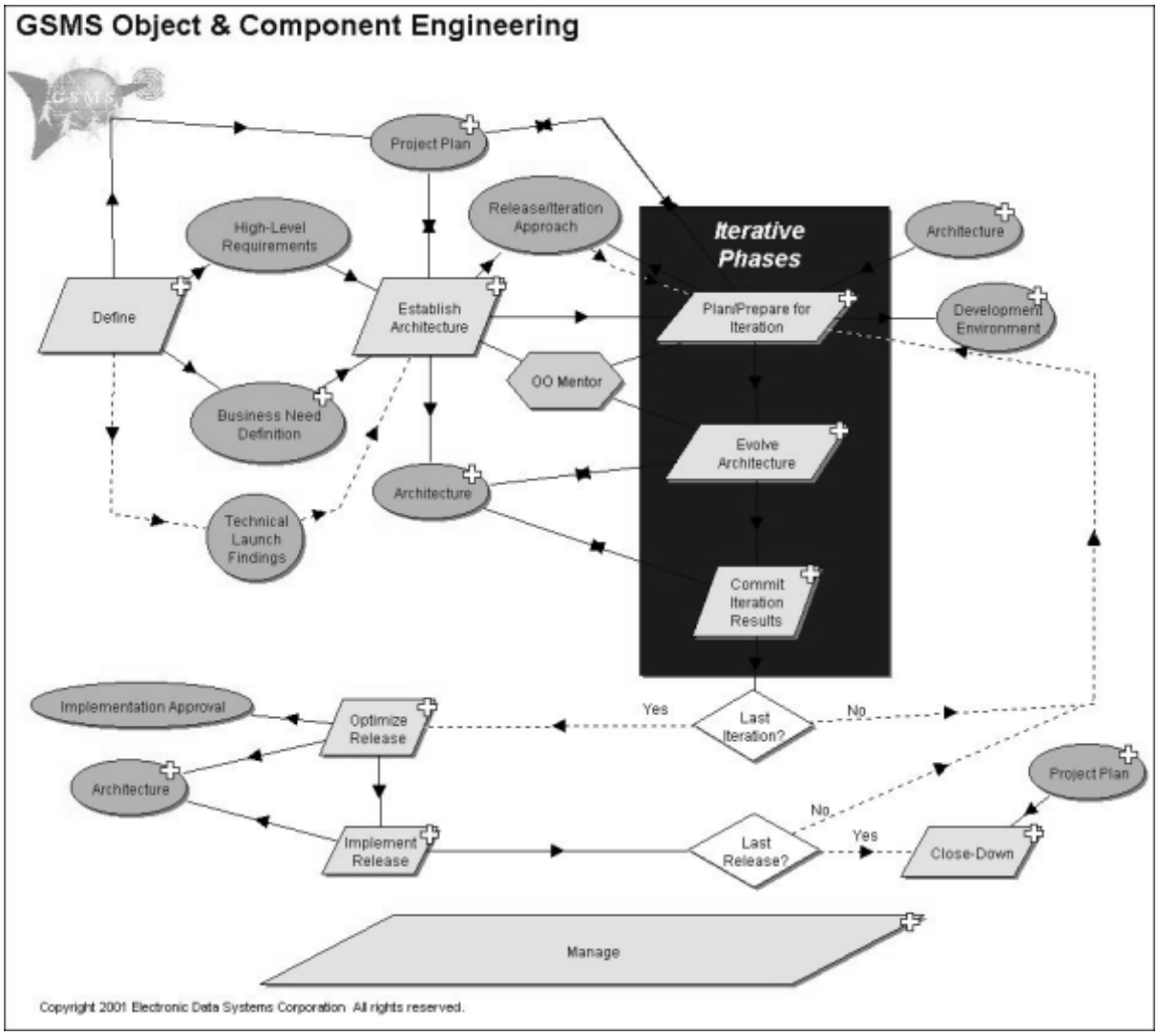
Prosessinäkymän avulla kuvataan järjestelmän suorittamisen kannalta tärkeät piirteet ja ei-toiminnalliset vaatimukset. Näihin kuuluvat järjestelmän saatavuus, samanaikaisuus, jakelu ja suoritusmalli. Suoritusmalli kuvaa järjestelmän prosessit ja sen miten ne on sijoiteltu eri paikkoihin. Prosessi voi olla ohjelman suoritus tai säikeen suoritus ohjelman sisällä. Prosessinäkymä voidaan kuvata usealla eri abstraktiotasolla.

Kehitysnäkymän pääasiallinen tarkoitus on kuvata materiaalia, joka on sovelluskehityksen tukena eli kehitysstrategioita, suunnitelmia ja standardeja. Tukea tarjotaan sekä toiminnalliselta, että ei-toiminnalliselta kannalta. Materiaali tukee kaikkia suunnittelun osa-alueita. Näkymän vaatima työmäärä riippuu paljolti siitä, mitä kokemuksia suunnitteluryhmällä on kohdealueesta. Jos kohdealue on uusi työmäärä voi kasvaa suureksi. Toisaalta aihealueen ollessa tuttu, voidaan aikaisempien projektien materiaalia käyttää uudelleen.

Fyysinen näkymä kuvaa järjestelmän infrastruktuurin ja ohjelmiston. Siihen kuuluu verkon rakenteet, laitteistot, ohjelmistot, systeemiohjelmistot, siirtomekanismit, yhteydet, tietokannanhallinjärjestelmä, operaatioiden hallinta ja ylläpito. Jos verkon rakenne on monimutkainen, voidaan tehdä moniulotteisia malleja.

4.4 Prosessin vaiheet

OCE-mallin mukaan ohjelmistoprosessi on jaettu vaiheisiin (aliprosesseihin). Nämä vaiheet on muodostettu hieman erilailla kuin ”perinteisempää” linjaa noudattavissa prosesseissa, kuten vesiputousmallissa (Royce, 1970) tai spiraalisessa mallissa (Boehm, 1996). Tämän huomaa siitä, ettei termejä kuten analyysi, suunnittelu, toteutus näy suoraan mallissa. Kaikki perinteisen ohjelmistoprosessin elinkaaren vaiheet kuitenkin ovat löydettävissä. Kuvassa 4.3 on kuvattu OCE-mallin päätaso.



Kuva 4.3. OCE mallin päätaso (EDS, 2001).

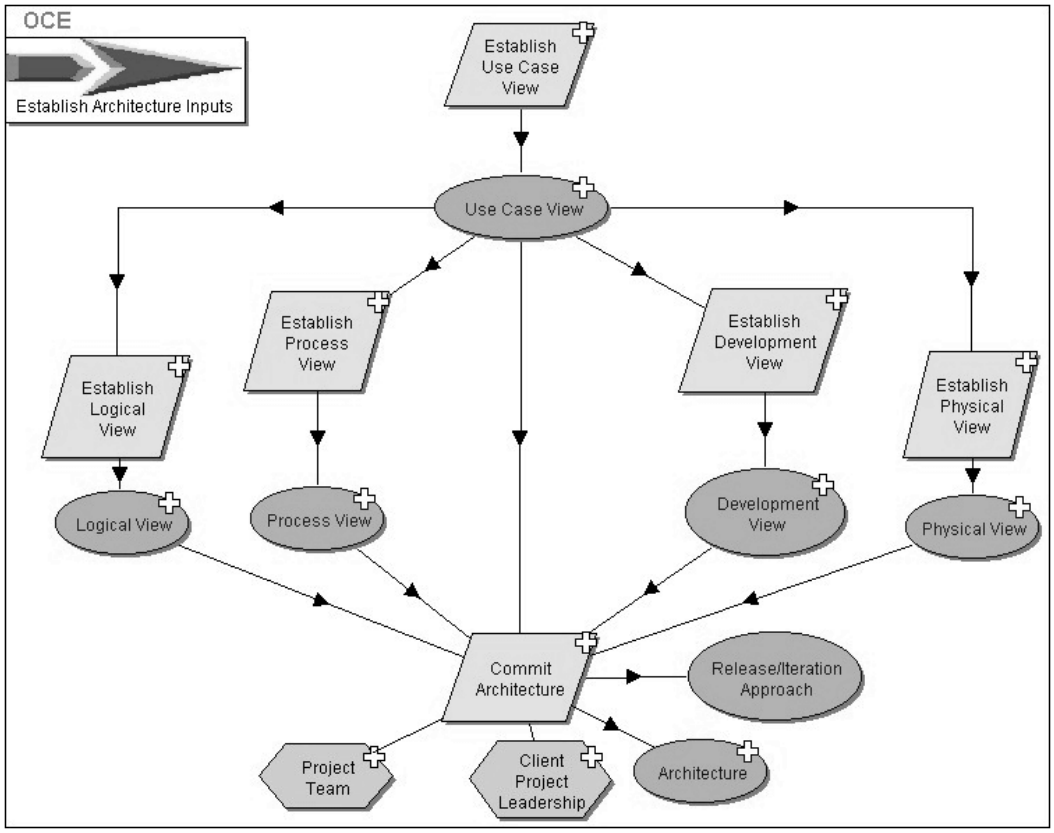
Kuten kuvasta 4.3 voi huomata, ei siinä näy analyysiä, suunnittelua ja toteutusvaihetta päätasolla. Ne sisältyvät *iteraatio-vaiheiden* (iterative phases) sisään. Tärkeää on myös huomata, että *projektin hallinta* (manage) on erotettu omaksi kokonaisuudekseen. GSMS sisältää myös projektinhallintaa varten oman työtyypin.

4.4.1 Tarvemäärittely

OCE alkaa tarvemäärittely-vaiheella (define). Vaiheen tarkoituksena on selvittää asiakkaan liiketoiminnalliset tarpeet (business need) ja tehdä projektisuunnitelma. Asiakkaan ongelmiin tehdään vaihtoehtoisia ratkaisumalleja, joista valitaan yhdessä paras. Projektisuunnitelmassa määritellään mm. aikataulut, käytettävät resurssit, budjetti, toteutusympäristö, riskit, yhteydenpitotavat ja projektin rajaus Tämä jako perustuu Project Management Instituten tarjoamaan malliin (Project Management Institute, 2001).

4.4.2 Arkkitehtuurin perustaminen

Arkkitehtuurin perustamisen (establish architecture) keskeinen tarkoitus on tuottaa alustava arkkitehtuurinen visio järjestelmästä, jonka pohjalta kehitystä voidaan alkaa iteroida. Vaiheeseen kuuluu ”4+1”-jaon mukaisten dokumentaation ja näkymien teko järjestelmästä.



Kuva 4.4. Arkkitehtuurin perustaminen (EDS, 2001).

Kuvasta 4.4. voi nähdä, aliprosessit ovat *käyttötapausnäkymän perustaminen*, *loogisen näkymän perustaminen*, *prosessinäkymän perustaminen, kehitysnäkymän perustaminen* ja *fyysisen näkymän perustaminen*. Jokainen aliprosessi sisältää yksityiskohtaiset ohjeet vaiheen suorittamiseksi. Prosessimalli ei pakota tekemään eri vaiheita missään määrätyssä järjestyksessä.

4.4.3 Iteraation suunnittelu ja siihen valmistautuminen

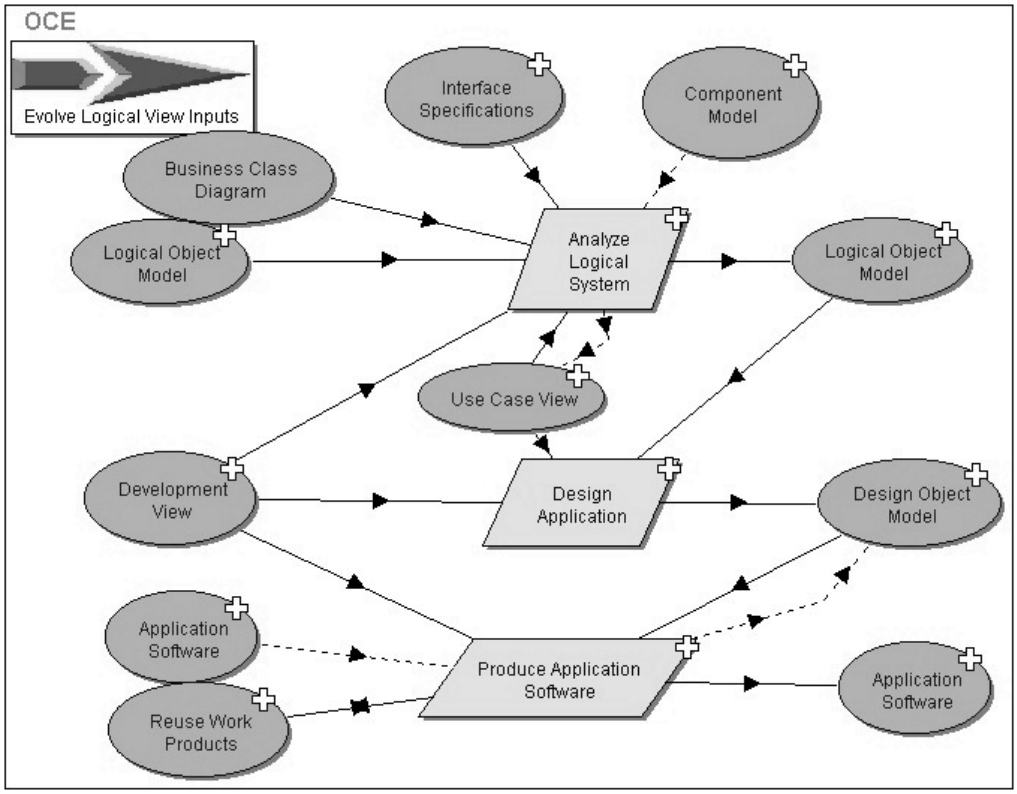
Kun järjestelmän alustava arkkitehtuuri on tehty, jatkuu sen kehitys iteraatioissa. Jokainen iteraatio on suunniteltava etukäteen. Tämä tehdään iteraation suunnittelu ja siihen valmistautuminen–vaiheessa (plan/prepare for iteration). Iteraatiolle kirjataan tavoitteet ja kuvaus, miten nämä tavoitteet saavutetaan. Tarvittavien iteraatioiden määrä voi vaihdella

erilaisissa projekteissa. OCE-mallin mukaan käytännössä usein sopivaksi määräksi on havaittu kolmeen kertaan suoritettu iterointi (EDS, 2001). Tämä on kuitenkin ainoastaan suositus.

Prosessimalli ei määritä onko iteraatioiden oltava lisäävää tai evolutiivista (vrt. Graham, 1989). Tämä voidaan määritellä projektisuunnitelmassa, joka sisältää tiedon siitä, montako iteraatioversiota ohjelmistosta tehdään projektin aikana ja montako iteraatiota sen teon aikana suoritetaan. Jokainen iteraatiotavoite on kuitenkin määriteltävä erikseen.

4.4.4 Arkkitehtuurin kehittäminen

Arkkitehtuurin kehittäminen (evolve architecture) tehdään iteraatioiden aikana. Se sisältää käytännössä suurimman osan koko järjestelmän kehittämisestä. Siinä kehitetään eteenpäin ”4+1”-mallin mukaisesti arkkitehtuurin perustaminen -vaiheessa tehtyä työtä. Sen aikana määritellään järjestelmäarkkitehtuuria. Analysoinnin, suunnittelun ja toteutuksen avulla kehitetään näkymiä eteenpäin ja järjestelmästä toteutetaan uusi iteraatioversio.



Kuva 4.5. Loogisen näkymän kehittäminen (EDS, 2001).

4.4.5 Iteraation tulosten vahvistaminen

Iteraation lopuksi sen tulokset katselmoidaan ja saavutettuja tuotoksia verrataan ja arvioidaan suunniteltuihin tuotoksiin. Tämä tehdään iteraation tulosten vahvistaminen –vaiheessa (commit iteration results). Tässä vaiheessa projektista voidaan myös etsiä ja kerätä uudelleenkäytettäviä osia. Valmiit tuotokset asetetaan muutostenhallinnan alaisuuteen. Vaiheen lopuksi siirrytään uudelle iteraatiokierrokselle tai optimointivaiheeseen.

4.4.6 Optimointi

Kun iteraatio on suoritettu loppuun, kokonainen iteraatioversio ohjelmistosta on valmis. Tässä vaiheessa ohjelmiston pää-toiminnallisuus on testattu ja se täyttää vaatimusmäärittelyssä määritellyt vaatimukset. Optimointi-vaiheen (optimise release) aikana voidaan parantaa sen suorituskykyä, asettaa se toimitettavaan pakettiin. Tällöin tehdään myös järjestelmän hyväksymistestaus.

4.4.7 Toimitus

Toimitus–vaiheen (implement release) aikana lopullinen versio toimitetaan ja/tai asennetaan asiakkaalle tuotantoympäristöön. Vaiheeseen kuuluu myös sovittua koulutusta ja uuden järjestelmän käyttöönotossa tarvittavaa tukea. Kaikki edellä mainitut toimenpiteet tehdään projektisuunnitelman ja vaatimusten mukaan. Jos versio ei ole viimeinen, jatkuu kehitys iteraatioissa.

4.4.8 Projektin lopetus

Kun todetaan, että kaikki järjestelmän osat on toimitettu ja ne toimivat kunnolla, on vuorossa *Projektin lopetus*–vaihe (close-down). Projektin lopputulos ja tehdyt tuotokset arvioidaan. Projektista kerätään tietoja tulevia projekteja varten. Tarvittavia muutoksia prosessiin ja käytettäviin työmenetelmiin arvioidaan ja tarvittaessa tallennetaan. Myös asiakkaan tyytyväisyys arvioidaan.

4.4.9 Projektin hallinta

Projektin hallinta–vaihe (manage) sisältää projektin aikaisen projektinhallinnan toimenpiteet. Projektin hallintaa tehdään koko projektin elinkaaren ajan. Tämä takaa sen, että projektiin kuuluu asianmukaista ylläpitoa, valvontaa, johtamista ja kommunikointia.

4.5 UML:n käyttö OCE:ssa

OCE prosessimalli määrittää *työtuotteet* (work product), joita käytetään syötteenä aliprosessille ja joita ne voivat tuottaa tai täydentää. Yksi työtuotteista (arkkitehtuuri) sisältää erilaisia UML kaaviota. Seuraavassa on tutkittu kaavio kerrallaan, missä yhteydessä ja millaisessa muodossa ne prosessissa esiintyvät.

4.5.1 Käyttötapauskaavio

Käyttötapauskaavio kuuluu OCE-prosessimallissa *käyttötapausmalliin* (use case model). Se on osa käyttötapausnäkymää. Käyttötapausmalliin kuuluu käyttötapauskaavio ja käyttötapauksen kuvaus. Käyttötapaukset on kuvattu käyttötapausmallissa siten, että toimijat ja käyttötapaukset esitellään erikseen ja ilman suhteita, minkä lisäksi käyttötapaukset kuvataan käyttötapauskaaviossa.

OCE:n käyttötapausmallia käytetään kolmeen tarkoitukseen. Ne ovat järjestelmän toiminnallisten vaatimusten dokumentointi, järjestelmää käyttävien ulkoisten entiteettien (antavat syötettä järjestelmään ja saavat siltä palautetta) roolien dokumentointi ja käyttötapausten keskinäisen uudelleenkäytön dokumentointi ts. se, miten käyttötapaukset jakautuvat kokonaisuuksiin siten, että toiset käyttötapauksiin käyttävät samaa käyttötapausta omassa työnkulussaan.

Käyttötapausmalli sijoittuu OCE-mallissa korkean tason vaiheisiin arkkitehtuurin perustaminen, arkkitehtuurin kehittäminen ja toimitus. Arkkitehtuurin perustaminen- ja arkkitehtuurin kehittäminen -vaiheissa se kuuluu ”4+1”-mallin mukaan käyttötapausnäkymään.

4.5.2 Luokkakaavio

OCE sisältää kolmenlaisia luokkakaavioita. Ne ovat *liiketoimintaluokkakaavio* (business class diagram), *analyysiluokkakaavio* (analyze class diagram) ja *suunnitteluluokkakaavio* (design class diagram). Kaikki kaaviot kuvataan standardin UML-notaation avulla.

Liiketoimintaluokkakaavion tarkoitus on kuvata liiketoimintaluokat, joiden avulla saadaan mallinnettua asiakkaan järjestelmälle asettamat vaatimukset. Kaavio kuvaa ainoastaan liiketoimintaluokat, mutta ei luokkia, jotka ovat tärkeitä järjestelmän teknisen toteutuksen kannalta. Näiden luokkien avulla saadaan ymmärrystä asiakkaan liiketoimintaongelmasta ja saadaan kerättyä järjestelmässä esiintyviä termejä.

Analyysiluokkakaavio helpottaa järjestelmän toteuttamisessa tarvittavien luokkien rakenteen analysointia. Kaavio ei sisällä luokkia, joita tarvitaan teknisen toteutuksen vuoksi. Analyysiluokat on kehitetty liiketoimintaluokkia eteenpäin kehittämällä. Niitä käytetään apuna analyysissä ja suunnittelussa samoin kuin liiketoimintaluokkiakin. Lisäksi ne ovat apuna määriteltäessä tietokantakaaviota.

Suunnitteluluokkakaavio kuvaa luokkia, joiden avulla saadaan täytettyä asiakkaan järjestelmälle asettamat vaatimukset. Se sisältää kaikki luokat, joita tarvitaan järjestelmän toteuttamiseen. Ohjelmakoodi voidaan luoda suunnitteluluokkien pohjalta.

Luokkakaavioista osa kuuluu ns. *oliomalliin* (object model). Oliomalli on jaettu *loogiseen oliomalliin* (logical object model), *suunnitteluoliomalliin* (design object model) ja *toteutusoliomalliin* (implementation object model). Näin analyysiluokkakaavio kuuluu loogiseen oliomalliin ja suunnitteluluokkakaavio kuuluu suunnittelu-oliomalliin. Toteutusoliomalliin kuuluvat toteutusluokat ovat ohjelmoijan kirjoittamaa koodia.

Luokkakaaviot sijoittuvat prosessin korkean tason vaiheissa siten, että liiketoimintaluokat luodaan pääosin arkkitehtuurin perustaminen -vaiheen aikana ja analyysi- ja suunnitteluluokkakaaviot arkkitehtuurin kehittäminen -vaiheen aikana. Ne kuuluvat loogisen näkymään.

4.5.3 Tilakaavio

Tilakaavio kuuluu luokkakaavion tavoin oliomalliin. Tilakaavioita tehdään pääasiassa vain pienelle osalle liiketoimintaluokista. Ne ovat kuitenkin sellaisia luokkia, jotka ovat keskeisiä järjestelmän toiminnan kannalta. Tilakaaviot sijoittuvat arkkitehtuurin kehittänen -vaiheeseen. Ne kuuluvat loogiseen näkymään.

4.5.4 Viestiyhteyskaavio

Viestiyhteyskaavion käytetään OCE:n mukaan mallintamaan ohjausvuota (flow of control) ja havainnollistamaan tyypillisiä skenaarioita. Viestiyhteyskaaviot sijoittuvat Arkkitehtuurin kehittäminen -vaiheeseen. Ne kuuluvat loogiseen näkymään.

4.5.5 Yhteistyökaavio

Yhteistyökaaviota käytetään OCE:n mukaan samoin kuin viestiyhteyskaaviota eli mallintamaan kontrollivirtaa ja havainnollistamaan tyypillisiä skenaarioita. Se on myös

avainasemassa suunnitteluoliomallissa. Yhteistyökaaviot sijoittuvat Arkkitehtuurin kehittäminen -vaiheeseen ja ne kuuluvat loogiseen näkymään.

Vuorovaikutuskaaviot (viestiyhteyskaavio ja yhteystyökaavio) ovat avainasemassa suunnittelun aikana suunnitteluoliomallissa. Niitä suositellaankin tehtävän suurimmalle osalle käyttötapauksista ja skenaarioista.

4.5.6 Toimintokaavio

Toimintokaavio on oliomallin viimeinen osa. Se liittyy mallin kautta aina luokkaan, käyttötapaukseen, pakettiin tai operaation toteutukseen. Toimintokaavio keskittyy kuvaamaan tietyn operaation suoritukseen liittyviä tapahtumia. Toimintokaaviot sijoittuvat arkkitehtuurin kehittäminen -vaiheeseen. Ne voivat kuulua loogiseen näkymään tai käyttötapausnäkymään.

4.5.7 Komponenttikaavio

Komponenttikaavio kuuluu *komponenttimalliin*, johon kuuluvat myös vuorovaikutuskaaviot. Komponenttimalli kuvaa alijärjestelmät, jotka toteuttavat ratkaisun ongelmaan. OCE:n mukaan komponenttikaavioiden tarkoitus on määrittää jokainen komponentti, määrittää rajapinta jokaiselle komponentille ja näyttää yhteydet komponenttien välillä. Komponentit sijoittuvat arkkitehtuurin kehittäminen -vaiheeseen ja ne kuuluvat prosessinäkymään.

4.5.8 Sijoittelukaavio

Sijoittelukaaviot eivät ole saaneet OCE:ssa kovinkaan suurta huomiota. Ne kuvataan vain eräänä tekniikkana tehtäessä ohjelmiston toteutussuunnitelmaa ja näin ollen ne kuuluvat useaan eri korkean tason vaiheeseen. Näitä vaiheita ovat arkkitehtuurin perustaminen, arkkitehtuurin kehittäminen, optimointi ja toimitus. OCE-mallin ”4+1”-näkymässä ne sijoittuvat sijoittelunäkymään.

5 UML OHJELMISTOPROSESSIN TUKENA

Kuten Jacobson & al., 1999 asian kuvaavat, ei täydellistä prosessia ohjelmiston tekemiseen ole. Ehkä suunta parempaan päin on kuitenkin otettu. RUP ja OCE voidaan rinnastaa toisiinsa oliopohjaisina ohjelmistoprosessimalleina. Kumpikin niistä on kehitetty tuotteeksi, mikä on harvinaista prosessimallien keskuudessa. Niistä löytyy paljon yhteisiä piirteitä, mutta myös eroja. UML on kummassakin prosessissa keskeisessä asemassa.

Tässä luvussa käsitellään prosessimallien eroja ja yhtäläisyyksiä sekä eri prosessimallien vahvuuksia. UML:n roolia prosesseissa on kuvattu omana osuutenaan. UML käyttöön liittyviä kysymyksiä, sen puutteita ja mallinnuksessa huomioon otettavia asioita on käsitelty osuuden loppupuolella.

5.1 Verkkoversiot

Ohjelmistoprosessien esittämiseen ei ole aiemmin juuri kiinnitetty huomiota. Ne ovat olleet luettavissa yleensä kirjoista. Tähän nämä kaksi tässä esiteltyä mallia tuovat huomattavan parannuksen. Molemmat ovat käytettävinä verkkoversioina HTML-sivustoina. RUP on tarjolla myös paikalliselle koneelle asennettava versiona. Näin jaetut mallit tarjoavat monia parannuksia prosessin käytössä.

Verkossa oleva prosessimalli parantaa sen saatavuutta. Kaikki mallia tarvitsevat löytävät sen samasta paikasta. Sen jakelu on myös helpompaa. Kun prosessimalliin tulee päivityksiä (molemmat malleista on versioitu ohjelmien tavoin), saadaan uusi versio nopeasti jakeluun. Syytä ei ole väheksyä sitäkään seikkaa, että paperin käyttö on huomattavasti vähäisempää. Myös paikallisen tason prosessien varastoinnin tarve on huomattavasti vähäisempi.

Verkkoversio tarjoaa prosessin käytön kannalta huomattavia etuja. Tarvittava tieto löytyy nopeasti. Esimerkit ovat sähköisessä muodossa, ja niitä voi käyttää muunneltuina omien projektien pohjina. Molempiin malleihin kuuluu myös hakutoiminto, joka helpottaa tiedon etsimistä. RUP:n yhteydessä puhutaankin enemmän tietämyskannasta kuin prosessimallista (Rational, 2001).

5.2 Dokumentaatio

Molemmat prosessimalleista tarjoavat runsaan määrän erilaisia mallidokumenttipohjia projektin tekoon. Nämä ovat OCE-mallissa pääasiassa Microsoft Word –dokumentteja. Myös joitain Excel-, Access- ja MS Project-pohjia kuuluu näihin dokumentteihin. OCE:ssa pohjat

eivät yleensä sellaisenaan käy dokumentointiin, vaan erilaisista malleista voidaan yhdistää halutunlainen dokumentti. Myös RUP tarjoaa valikoiman erilaisia pohjia. Valikoima sisältää hieman enemmän formaatteja kuin OCE. Pohjaformaatit ovat Word, Html, Microsoft Project, Frame Maker ja Rational SoDA. RUP:n pohjat on tarkoitettu käytettäviksi sellaisenaan, mutta halutessaan tarpeettomia asioita voi pudottaa pois dokumenteista.

OCE-mallissa dokumentit on jaoteltu *työtuoterakenteisiin* (Work Product Structure). Projektiryhmä valitsee rakenteeseen kuuluvista dokumenttipohjista tarvitsemansa. Tätä prosessia kutsutaan prosessin *räätälöinniksi* (tailoring). Jokainen projekti sisältää prosessimallin määrittämän hakemistorakenteen, johon toteutettava dokumentit sijoittuvat. Tämä päätason hakemistorakenne sisältää myös alihakemistoja, jossa päätasoon kuuluvat dokumentit on jaoteltu tarkemmin.

Päätasolla OCE-malliin kuuluu seitsemän eri työtuotetta, jotka on sijoitettu työtuoterakenteeseen. *Organisaation ympäristö* (organizational environment) sisältää tietoa asiakasorganisaatiosta liiketoiminnan näkökulmasta. *Järjestelmän ympäristö* (systems environment) sisältää tietoa työkaluista, apuohjelmistoista, standardeista ja menettelytavoista. *Liiketoimintatapaus/Liiketoimintayhteydet* (business case/relationship) sisältää tietoa asiakkaan liiketoiminnan tarpeista ja sen liiketoiminnan yhteyksistä sekä tehdyt tarjouspyynnöt. *Projektisuunnitelma* (project plan) sisältää projektisuunnitelman. *Seurantatyötuotteet* (tracking work products) sisältää seurantaraportteja projektin tuloksista. *Organisaation arkkitehtuuristandardit ja menettelytavat* (organizational architecture standards and procedures) sisältää tietoa käytettävistä koodaus-, käyttöliittymä- ja työkalustandardeista. *Arkkitehtuuri* (architecture), joka on tärkein tuotetuista työtuotteista, sisältää ohjelmiston suunnittelussa syntyneen materiaalin ja mallit jaettuna ”4+1”-mallin mukaisesti.

Työtuote voi jakautua vielä pienempiin kokonaisuuksiin. Niinpä esimerkiksi suuremmassa projektissa projektisuunnitelma voi sisältää erilliset hakemistot *rajauksen hallinta-suunnitelmalle* (scope management plan), *riskien hallintasuunnitelmalle* (risk management plan), *laadun ja kontrollin hallintasuunnitelmalle* (quality/control management plan), *työryhmän hallintasuunnitelmalle* (team management plan), *kustannusten hallintasuunnitelmalle*, (cost management plan), *hankintojen hallintasuunnitelmalle* (procurement management plan), *kommunikoinnin hallintasuunnitelmalle* (communication management plan) ja *aikataulun hallintasuunnitelmalle* (schedule management plan).

RUP:ssa dokumentit on jaoteltu hakemistoihin työvirran vaiheiden mukaisesti. Jokainen työnkulku sisältää *artefaktijoukon* (artifact set), joka on kuvattu esimerkiksi dokumentissa. Liiketoiminnan mallintaminen –työnkulku sisältää artefaktit, jotka kuvaavat järjestelmän liiketoiminnan taustat. Vaatimukset–työnkulku määrittelee järjestelmältä vaadittavat vaatimukset. Analyysi & suunnittelu –työnkulku kertoo tiedon, miten vaatimuksissa esitetyt vaatimukset toteutetaan. Toteutus–työnkulku kuvaa, miten suunniteltu järjestelmä on toteutettu. Testaus-työnkulku sisältää kaiken testausinformaation. Sijoittelu–työnkulku määrittelee tiedon siitä, miten Toteutus-työnkulussa kuvattu tuote siirretään tuotantoympäristöön. Konfigurointi ja muutosten hallinta –työnkulku kertoo työvaiheeseen kuuluvat tiedot. Projektin hallinta –työnkulku kuvaa projektin ja prosessin suunnittelussa tarvittavat tiedot. Viimeiseksi ympäristö-työnkulku sisältää artefaktit, joita käytetään ohjeina koko projektin ajan.

OCE:ssa dokumentointi on melko vapaasti muodostettavissa pohjista. Projekti joutuu aina itse tekemään valinnat tarjolla olevasta valikoimasta. Malli kuitenkin myös määrää joitain dokumentteja, joita on käytettävä sellaisenaan ja jotka eivät ole räätälöitävissä. OCE-mallissa ei oteta kantaa siihen, miten dokumentit jaellaan. Käytettävät pohjat ovat vapaasti muunneltavissa HTML:ksi tai muuksi formaatiksi. OCE:n peruslähtökohtana on ollut työvälineriippumattomuus. Jokainen projekti saa itse valita käytettävät CASE- yms. välineet omien tarpeittensa mukaan.

Vaikka RUP tarjoaa dokumentteja prosessin mallintamiseen avuksi, ei niiden rooli ole kovinkaan tärkeä. Kruchten (2000) rinnastaa artefaktit muiden prosessimallien työtyyppeihin, mutta hän täsmentää, etteivät ne yleensä ole dokumentteja. Hänen mukaansa RUP yrittää välttää järjestelmällistä paperidokumenttien tuottamista. Tuottavin ja käytännöllisin tapa ylläpitää järjestelmän artefakteja on pitää ne työkalujen tuottamissa malleissa. Kun tarvitaan jotain dokumentteja, ne generoidaan näiden työkalujen avulla. Näin saadaan aina viimeisintä tietoa oleva dokumentointi. RUP:ssa CASE välineiden käyttö on keskeisessä osassa. Niiden avulla voidaan saavuttaa huomattavaa etua esimerkiksi tuottavuudessa (King ja Galliers, 1994).

Työkalut ovat toisaalta RUP:n heikkous, koska se on niin tiukasti sidottu juuri Rationalin omiin tuotteisiin. Lähtökohtana prosessia tehdessä on ollut se, että RUP:a käyttävä organisaatio käyttää myös Rationalin tuotteita projektin ylläpitoon. Tämä, ei kovinkaan yllättävä seikka tulee esille prosessimallissa. Ovathan prosessin kehittäjät Rationalin

palveluksessa. Kaikkiin työvaiheisiin tarjotaan yksityiskohtaisia ohjeita, miten vaihe tehdään siihen tarkoitetulla Rationalin tuotteella. Toisaalta, jos ei tarvitse käytännön apua mallien tuottamisessa, ei mikään estä tekemästä RUP:n artefakteja muillakin kuin Rationalin välineillä. Tällöin menetetään kuitenkin monia työvälineiden tarjoamia etuja, kuten automaattinen dokumentointi Rosella tuotetuista malleista (SoDA), jos käytetyt välineet eivät tukea tähän tarjoa.

5.3 UML:n rooli prosesseissa

UML:llä on keskeinen rooli molemmissa prosesseissa, ja sitä käytetään molemmissa melko samalla tavalla, mutta erojakin löytyy. Nämä erot tulevat esille lähinnä käyttötapausten ja luokkakaavioiden käytön yhteydessä.

OCE:ssa ja RUP:ssa käyttötapausten käyttäminen eroaa joiltain osin. OCE:ssa käyttötapauksien rooli jää hieman epäselväksi. Niitä käytetään pääasiassa liiketoiminnan ja vaatimusten kuvaukseen. Eroa näiden välillä ei kuitenkaan ole tehty aivan yhtä selkeästi kuin RUP:ssa. RUP jakaa käyttötapaukset liiketoimintakäyttötapauksiin ja käyttötapauksiin. Näin on tehty selvä ero näiden kahden ryhmän, järjestelmää hieman eri näkökulmista kuvaavien käyttötapausten kesken.

Luokkakaavioiden rooli on molempien mallien mukaan keskeinen, ja niitä on kuvattu molemmissa yksityiskohtaisesti. Molemmat mallit jakavat luokkakaaviot eri ryhmiin samalla tavalla. Kumpikin sisältää liiketoiminta, analyysi- ja suunnitteluvaiheen luokkakaavioesitykset. Erot ovatkin pääasiassa esitystavassa. Kun OCE käyttää UML:n standardinotaatiota suorakulmioesityksineen, käyttää RUP omaa notaatiotaan erilaisten luokkien kuvaukseen. Tämä voi aiheuttaa aluksi vaikeuksia, koska se vaatii lisänotaation opettelua. RUP:n kuvaustapojen käyttö ei ole välttämätöntä, koska normaalien luokkakaavioiden käyttö stereotyyppien kanssa kuvaa saman asian. Toisaalta tämä on pyrkimys poispäin UML:n filosofiasta ”sama notaatio kaikkeen”, josta monet UML:n ongelmat johtuvat (Simons ja Graham, 1998).

Oliokaaviot ovat molemmissa malleissa jätetty kokonaan käyttämättä ja tästä voi päätellä, ettei niiden rooli ole kovin keskeinen mallinnuksessa. Järjestelmän dynaamisen toiminnan kuvaus on toteutettu muiden dynaamisen toiminnan kuvaamiseen tarkoitettujen kaavioiden avulla. Oliot esiintyvät tila-, viestiyhteys-, yhteistyö- ja toimintokaavioissa, joten niillä on siten rooli mallinnettaessa järjestelmää. Tila-, vuorovaikutus- (viestiyhteys- ja

yhteistyökaavio), ja toimintokaavioita käytetään prosessimallien mukaan mallinnuksessa melko samalla lailla ja samoissa yhteyksissä. Ne on sijoitettu molemmissa prosessimalleissa myös oliomalliin.

Komponenttikaavioiden käytöstä OCE antaa hieman paremmin informaatiota kuin mitä RUP tekee. Tämä johtuu osaltaan siitä, että OCE:a suunniteltaessa komponenttiajattelu on ollut keskeisesti mukana. Sijoittelukaavioiden käyttö ei kummassakaan prosessimallissa ole kovin tärkeässä roolissa. OCE:ssa niistä annettava kuvaus on kuitenkin vielä RUP:a niukempi.

RUP pystyy sitomaan eri kaaviot paremmin kokonaisuudeksi. Muun muassa käyttötapaus- ja luokkakaavioiden keskinäinen suhde tulee selvästi esiin. Tämä käy selville esimerkiksi kuvasta 2.6. Tämä voi johtua siitä, että prosessimalli on kehitetty rinnan UML:n kanssa. Kaavioiden käyttöä ja prosessimallia on näin voitu verrata koko ajan toisiinsa. Toisaalta tällainen mallinnustekniikan käyttö lisää ylläpidettävien mallien määrää ja aiheuttaa näin lisätyötä. Täytyy myös muistaa se seikka, ettei käytetty kuviointinotaatio ole UML:n notaation mukaista.

Käytettävän mallinnuksen määrään ts. siihen, mistä kaikesta malleja täytyy tuottaa, kumpikaan prosessimalli ei anna selvää vastausta. Molemmista löytyy viittauksia siihen, että kaiken mallintaminen on mahdotonta. Arkkitehtuurin kannalta keskeisten asioiden mallintamista korostetaan kummassakin prosessimallissa. RUP tarjoaa koko järjestelmää kuvaavia esimerkkejä ja kirjoja aiheesta, joista voi olla hyötyä mallien määrää suunniteltaessa Quatrani (2000). OCE:sta löytyy pelkästään yksinäisiä, irrallaan kokonaisuudesta olevia esimerkkejä, joten tämä kysymys jää suunnittelijoiden ratkaistavaksi.

5.4 Prosessimallien hyviä ja huonoja puolia

OCE ei ota kovin paljon kantaa siihen, missä järjestyksessä prosessin eri vaiheet on suoritettava määrittelyä ja projektin lopettamista lukuun ottamatta. ”4+1”-mallin osat voidaan OCE:n mukaan muodostaa missä järjestyksessä tahansa, oli kyseessä arkkitehtuurin perustaminen- tai arkkitehtuurin kehittäminen -vaihe. ”4+1”-malli on koko prosessin perusta.

Tästä seikasta voi olla sekä hyötyä että haittaa. Toisaalta se tarjoaa mahdollisuuden käyttää prosessimallia vapaammin, mistä on hyötyä kokeneille suunnittelijoille. Joissakin tapauksissa projektin alkuvaiheessa ei pystytä tarkkaan määrittelemään sitä, mitä oikeastaan pitäisi tehdä. Usein asiakkaallakaan ei ole kuvaa siitä, mitä hän sovellukselta todella haluaa. Tällöin voi olla hyötyä vapaammat kädet suunnittelulle antavasta prosessista.

Toisaalta vapaus vaatii kokeneita suunnittelijoita. Kokemattomalle suunnittelijalle voi tuottaa ongelmia se, ettei jokaista prosessiin kuuluvaa vaihetta suoriteta tietyssä etukäteen annetussa järjestyksessä. Näin ei tarjota mitään selkeää polkua, jota pitkin etenemällä vaihe saadaan tehtyä. Jako ”4+1”–mallin mukaan voi myös aiheuttaa ongelmia. Suunnittelijoille voi tuottaa ongelmia sijoittaa asiat niille kuuluviin kokonaisuuksiin. OCE-mallissa painotetaankin sitä seikkaa, että suunnittelijoiden tulee olla kokeneita, heillä täytyy olla kokemusta suunnittelusta vastaavissa projekteissa ja vahva olio-ohjelmoinnin menetelmien tuntemus.

RUP:ssa tarjotaan tiukemmat rajat projektin vaiheille. Projekti etenee eri työvirran läpi iteraatioissa, ja tehtävät suoritetaan aikajärjestyksessä. Prosessimallin käytössä voi OCE:a helpommin joutua tilanteeseen, ettei malli sovellukaan kyseisen ohjelmiston suunnitteluun. Tällaiset seikat pitäisi pystyä huomaamaan etukäteen, mutta käytännössä tämän tekeminen voi olla vaikeaa. Jos se kuitenkin huomataan, voidaan RUP:n prosessia räätälöidä projektiin sopivaksi. OCE:ssa prosessimallin katsotaan olevan sen verran väljä, ettei tällaista mekanismia tarvita. Lisäksi GSMS:ssä on tarjolla myös perinteisempi ja ohjaavampi työtyyppi (Dev+), joka soveltuu OCE:a paremmin myös vähän kokemattomamman projektiryhmän käytettäväksi.

RUP:n räätälöinti on toteutettu siten, että verkossa olevasta prosessimallista voidaan muokata halutun kaltainen siihen tarkoitukseen tarkoitetun välineen avulla. Tämä väline on nimeltään Development Kit. Developement Kitin avulla saadaan tehtyä uusi verkkosivusto, johon voidaan laittaa linkkejä niihin RUP:n osiin, joita tarvitaan räätälöidyssä projektissa. Kruchten, (2001b) täsmentää, että räätälöidyn prosessinmallin tekeminen kuitenkin vaikeuttaa RUP:n uusien versioiden käyttöönottoa. Mitä enemmän prosessia on räätälöity, sitä vaikeampaa on uuden RUP:n uuden version käyttöönotto.

Arkkitehtuurilla on keskeinen rooli molemmissa prosesseissa. Molempien prosessimallien takana on ajatus ”4+1”–mallista. OCE:ssa tätä on korostettu erityisesti vielä sillä, että prosessien vaiheet on prosessin kuvauksessa jaettu ”4+1”-mallin mukaan. Tämä aiheuttaa sen, että OCE on aluksi hieman sekava. Ei ole aivan selkeää se, minne lokeroon mikäkin asiakokonaisuus kuuluu. Sen jälkeen, kun tämä on selvinnyt, käyttö helpottuu.

5.5 UML:n puutteita

UML:stä löytyy vielä ominaisuuksia, jotka asettavat sen käyttökelpoisuuden hieman kyseenalaiseen valoon. Tämän huomaa esimerkiksi silloin, kun alkaa tutkia UML:ä

käsittelevää kirjallisuutta. Kun puhutaan kielestä ja sen notaatiosta odottaisi sen olevan formaali. Näin ei näytä UML:n kohdalla vielä täysin olevan. Sama asia on esitetty hieman poikkeavalla tavalla eri kirjoissa. UML on kokenut pieniä muutoksia eri versioiden välillä, mutta tämäkään ei selitä pieniä kirjallisuudessa esiintyviä eroja. Oman osansa eroihin tuovat mallinnusvälineet, joiden avulla ei pysty piirtämään oikean notaation mukaisia kaavioita.

Tästä muodostuu väkisin kuva, ettei UML ole vielä kehittynyt valmiiksi välineeksi ohjelmistojen mallintamiseen ja ei eikä ollen ole täysin valmis väline ohjelmistojen mallintamiseen. Tämän seikan tuovat esiin myös Engels ja Groenewegen (2000) puhuessaan UML:n puutteista. He mainitsevat kolme keskeistä puutetta, jotka löytyvät UML:stä. Ensimmäiseksi UML ei ole tarpeeksi sovellusaluespesifistä. Tämä johtuu siitä, että UML on kehitetty yleiskäyttöiseksi (general purpose) mallinnuskieleksi. Toiseksi osaa UML:n syntaksista ja semantiikasta ei ole edelleenkään määritelty tarpeeksi formaalisti, mikä mahdollistaa sen useanlaiset tulkinnat. Kolmanneksi mahdollisuus muuttaa automaattisesti UML suoraan toteutukseksi on edelleen avoin asia.

UML:n yleiskäyttöisyyden aiheuttamiin ongelmiin on tarjolla eräs ratkaisu. Tällainen on UML:n laajentamismekanismien käyttö. Conallen (1999) tarjoaa tästä esimerkin verkkosovellusten teon yhteydessä. Hänen mukaansa pelkkien luokkakaavioiden avulla ei pystytä kuvaamaan tyhjentävästi verkkosovelluksen toimintaa. Hän tarjoaa tähän ongelmaan ratkaisuna luokan stereotyyppien käytön. Tällä keinolla pystytään spesifioimaan esimerkiksi, kuuluvatko luokat asiakkaan (stereotyyppi <<client page>>) vai palvelimen (stereotyyppi <<server page>>) päähän asiakas/palvelin-sovelluksessa. Myös assosiaatioissa luokkien välillä voidaan käyttää tämänkaltaisia stereotyyppejä. Tällöin stereotyyppi voi olla esimerkiksi <<build>>. Tässä tapauksessa palvelinsivu muodostaa asiakassivun. Vastaavalla tavalla laajentamismekanismeja voidaan käyttää esimerkiksi reaaliaikaisten järjestelmien mallinnuksessa (Selic, 1999).

Laajentamismekanismien avulla UML:ää voidaan käyttää enemmän sovellusalue-spesifisen tiedon mallintamiseen. Laajennokset voivat toisaalta aiheuttaa ongelmia mallin ymmärtämisessä projektiryhmän eri jäsenten kesken. Tämä ongelma voi esiintyä nimenomaan silloin, kun on määritelty omia stereotyyppejä. Niiden käyttö vaatii yhteisistä käytännöistä ja merkintätavoista sopimisen projektiryhmän sisällä, jotta kaikilla on käsitys siitä, mitä käytetyt stereotyypit tarkoittavat. Ne on myös dokumentoitava.

Simons ja Graham (1998) tuovat esille lisää UML käyttöön liittyviä ongelmia. Heidän mukaansa monet ongelmista eivät johdu pelkästään UML:stä vaan myös sitä käyttävistä ihmisistä. Heidän mukaansa suurin osa ongelmista johtuu UML:n notaation monikäsitteisyydestä, kognitiivisesti väärän suuntaisesta suunnittelusta, systeemin keskeisten ominaisuuksien puutteellisesta kuvauksesta, välineiden riittämättömästä tuesta ja suunnittelijoiden kokemattomuudesta.

Notaation monikäsitteisyys tarkoittaa heidän mukaansa sitä, että esimerkiksi käyttötapaukset ovat usein liian monitulkintaisia. Suunnittelijat eivät myöskään ymmärrä, mitä käytetyillä merkintätavoilla tarkoitetaan. Heidän mukaansa käytännön kokemuksien kautta on havaittu, että suunnittelijat käyttävät liian paljon aikaa alustavien semanttisten kuvausten tekemiseen analysointi vaiheessa luullen siitä olevan paljon hyötyä myöhemmin. Todellisuudessa monet näistä suunnittelijoiden huomioista ovat merkityksettömiä myöhemmin.

Simons ja Graham (1998) mukaan esimerkiksi assosiaatioiden semanttisesti oikea nimeäminen voi olla ajanhukkaa. Näiden suhteiden oikea löytäminen on erittäin vaikeaa, mikä kasvattaa niiden löytämiseen tarvittavaa aikaa. Järjestelmän keskeisten ominaisuuksien kuvaaminen ei onnistu kunnolla esimerkiksi siksi, että UML soveltuu hyvin kuvaamaan analysoinnissa käytettäviä abstrakteja suhteita käsitteiden välillä, mutta on huono erottamaan tärkeitä asiakas/palvelin-vaatimuksia.

Oikeiden kaavioiden käyttö oikeassa tilanteessa voi aiheuttaa myös päänvaivaa. Sopivien kaavioiden löytäminen varsinkaan dynaamisten asioiden mallintamiseen ei välttämättä ole helppoa. Tätä ei helpota se seikka, että UML:ssa on puutteita ajonaikaisessa mallintamisessa (Tanuan, 1998). Prosessimallit voivat helpottaa tätä ongelmaa, mutta myös niiden soveltumisessa juuri tehtävän kaltaisen ohjelmiston prosessiksi voi olla ongelmia. Kaavioiden käyttöönottoa mietittäessä tulee lähtökohtana pitää, ettei kaikkia yhdeksää kaaviota ole mahdollista käyttää missään projektissa. Päätös käytettävistä kaavioista on tehtävä jokaisen sovellusalueen kohdalla erikseen (Jäger & al., 1999).

Myös mallinnusvälineet voivat aiheuttaa ongelmia. Mallinnusvälineet sisältävät ominaisuuksia, jotka voivat estää tekemästä halutun kaltaisia malleja, vaikka UML notaation mukaan sen pitäisi olla täysin mahdollista. Tällainen on esimerkiksi Microsoft Vision Enterprisen ominaisuus, joka estää piirtämästä luokkakaaviossa paketin sisään luokkia. Väline yksinkertaisesti estää tämän teon. Ihmetystä herättävät myös toisen alan ”huippuvälineen”,

Rational Rose piirto-ominaisuudet. Mallinnus sisältää paljon piirtotyötä ja luulisi, että sen onnistumiseen kiinnitettäisiin erityistä huomiota. Rosen kohdalla näin ei ainakaan näytä olevan.

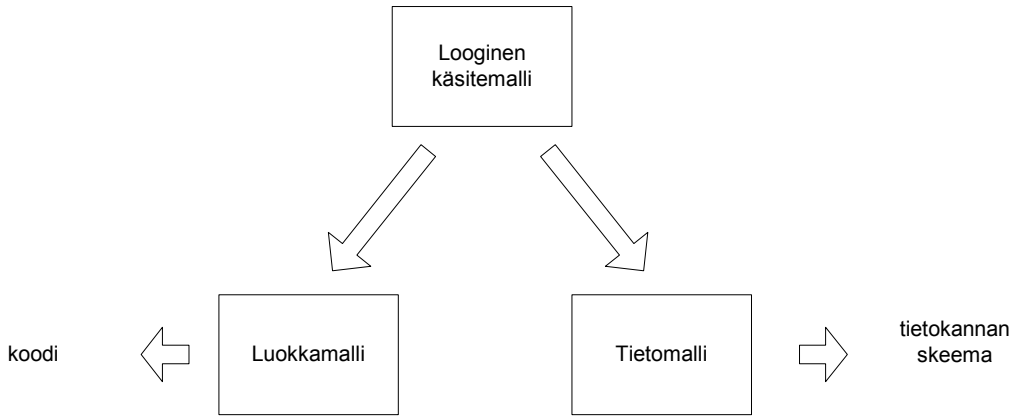
5.6 UML mallintamisessa

Vaikka UML sisältää puutteita, on se kuitenkin merkittävä apuväline ohjelmistokehitystyöhön. Löytyy suuri määrä esimerkkitapauksia, joissa sen käyttö on merkittävästi auttanut ohjelmistotyön tekemistä (Rational, 2001b). UML:n avulla on saatu parannettua uudelleenkäytettävyyttä, lyhennettyä ohjelmistojen kehitysaikaa ja nostettua tuottavuutta (Rational, 2001a).

Tärkeimpiä syitä siihen, miksi UML:ää käytetään, perustellaan EDS:n (2001) mukaan sillä, että se on avoin standardi ja eikä siihen ole tulossa omistajan tekemiä muutoksia, se tukee koko ohjelmiston elinkaarta, se soveltuu erilaisten ohjelmistojen tekoon ja sen pohjana on kokemus ja käyttäjien todellinen tarve. Lisäksi monet työkalut tukevat sen käyttöä. Todellisuudessa tilanne alkaa olla se, ettei oliopohjaisen ohjelmiston mallintamiseen ole tällä hetkellä tarjolla muuta varteenotettavaa kilpailijaa.

Mallintamisen suunnittelussa joutuu tekemään kompromisseja, koska kaiken mallintaminen on mahdotonta. Kaavioita voi helposti kertyä niin suuri määrä, että niiden ylläpito ei enää onnistu. Mallintamisessa tulisikin pyrkiä keskittymään olennaisiin asioihin. Arkkitehtuuri on nostettu molemmissa prosessimalleissakin erittäin keskeiseen rooliin. Lähtökohtana voidaan pitää, että se tulisi mallintaa melko täydellisesti.

Mallintamiseen voidaan valita tietty aspekti tehtävän ohjelmiston tai sovellusalueen mukaan. Ohjelmistosta voidaan analyysin aikana muodostaa looginen käsitemalli, joka kuvaa tehtävän järjestelmän liiketoiminnan. Tästä loogisesta käsitemallista voidaan kehittää toisaalta luokkamallia, toisaalta tietomallia (kuva 5.1). Usein voi olla järkevää ylläpitää ainoastaan toista näistä malleista, koska ylläpitoon ja muutosten hallintaan ei tarvitse sitoa niin paljon resursseja. Tällöin on kysymys lähtökohdasta, joka suunnitteluun otetaan. Tällaisten päätösten tekeminen vaatii aina sovellusaluespesifistä kokemusta.



Kuva 5.1. Luokkamalli ja tietomalli.

RUP:ssa lähtökohtana se, että järjestelmästä ylläpidetään kumpaakin mallia. Painotuksen voi katsoa olevan enemmän luokkamallin kuin tietomallin puolella, koska prosessikuvauksessa tietomalli saa huomattavasti vähemmän huomiota. OCE mahdollistaa selkeämmin sen, että prosessi voi keskittyä pelkästään tietomalliin ja järjestelmä kuvataan sen näkökulmasta.

UML:ään, niin kuin moneen muuhunkin uuteen asiaan ohjelmistotuotannossa, liittyy paljon ”hypeä”. Se kuulostaa monien artikkelien ja kirjoitusten mukaan olevan väline, joka ratkaisee kaikki ohjelmistokehityksen ongelmat. Tätä se ei tietenkään tee. Tulos voi olla myös päinvastainen, jos ei olla täysin selvillä sen käyttöönotossa vaadittavista toimenpiteistä.

UML:n käyttöönotto ja käyttö ei ole aivan yksinkertainen asia. Se vaatii yritykseltä panostusta henkilökunnan kouluttamiseen. On ensiarvoisen tärkeää, että kaikki ymmärtävät, mitä tehdyillä malleilla kuvataan. Perusteiden oppimisen jälkeen on edessä ongelma, miten malleja tehdään ja mitä kaavioita käytetään asioiden mallintamiseen missäkin projektin vaiheessa. Tässä UML:ää käyttävät prosessimallit tarjoavat apua. Toisaalta tutkielmassa esitellyt prosessimallit tuovat myös lisävaatimuksia esimerkiksi projektin hallintaan (Kruchten, 2001a). Käyttäjillä olisi hyvä olla myös kokemusta olioperustaisesta suunnittelusta ennen UML:n käyttöönottoa.

Kun menetelmät ovat hallussa, voi keskittyä olennaiseen eli mallintamiseen. Aluksi on syytä pohtia tarkkaan, millaisia malleja projektissa aiotaan tuottaa ja mitä tarkoitusta varten mikäkin näistä kaavioista tehdään. On esimerkiksi helppo sortua toisaalta liian tarkkojen yksityiskohtien kuvaamiseen tulevasta järjestelmästä vaatimusmäärittelyä tehtäessä. Asiakas ei välttämättä saa mitään lisäarvoa järjestelmän sisäisen toiminnan tuntemisesta. Analyysiä suoritettaessa voidaan alkaa puolestaan kuvaamaan järjestelmän teknistä toimintaa, vaikka

sitä nimenomaan tulisi välttää. Toisaalta järjestelmän teknisessä suunnittelussa liian korkealla abstraktiotasolla olevista malleista, joista ei saa tarpeeksi yksityiskohtaista ja yksikäsitteistä kuvaa asiasta, ei ole juuri mitään hyötyä.

5.7 Mallinnusvälineiden käyttö

Mallinnusvälineitä on tarjolla suuri joukko ja niiden valinta voi aiheuttaa ongelmia projektin alkuvaiheessa. Yleisin kysymys on, kannattaako sijoittaa usean kymmenen tuhannen markan lisensseihin ja valita laajaa tukea tarjoava mallinnusväline, johon kuuluu esimerkiksi dokumentointia ja projektin hallintaa tukevia työkaluja. Toinen ääripää voi olla halpa piirrostyökalu, jolla pystytään tuottamaan tarvittavat kaaviot UML:n notaation mukaisesti ilman syntaksin tarkastus yms. ominaisuuksia. Kärjistetysti voisi sanoa, että tarjolla on hyviä mallinnusvälineitä huonoilla piirto-ominaisuuksilla ja hyviä piirrosvälineitä huonoilla mallinnusominaisuuksilla.

Yleisesti näyttää siltä, että projektin kokoa voidaan pitää ratkaisevana tekijänä mallinnusvälineen valinnassa. Pienissä ohjelmistoissa malleja pystytään hallitsemaan, ylläpitämään ja dokumentoimaan ilman tehokkaita mallinnustyökaluja. Monia niiden tarjoamista ominaisuuksista ei tarvita. Näitä ovat esimerkiksi automaattinen koodin generointi luokkakaavioista tai tietokantakaavion generointi tietokannasta.

Toisaalta ohjelmiston koon kasvaessa voi mallien ylläpidon osuus kasvaa liian suureksi ilman kehittyneitä mallinnustyökaluja, jotka sisältävät monesti myös dokumentoinnin automatisointia. Aikaisemmat, huonosti hoidetut projektit voivat olla dokumentoimatta ja mallintamatta kokonaan. Jos tällaisten projektien pohjalta aletaan kehittää uutta, voi mallinnusvälineiden generointi ja mallin generointiin (forward and reverse engeneering) käytettävistä ominaisuuksista olla merkittävää hyötyä.

6 YHTEENVETO

Käsiteltyjen ohjelmistoprosessimallien ja UML:n käytön avulla saadaan parannettua ohjelmistojen uudelleenkäytettävyyttä, lyhennettyä ohjelmistojen kehitysaikaa ja nostettua tuottavuutta. Tällaisten menetelmien käyttöönotto alkaa olla välttämätöntä, koska ohjelmistot ja järjestelmät kasvavat ja monimutkaistuvat koko ajan.

UML:n ja prosessimallien käyttöönotto ja käyttö kuitenkin vaativat, että suunnittelijoiden tulee olla kokeneita, heillä täytyy olla kokemusta suunnittelusta vastaavanlaisissa projekteissa ja vahva olio-ohjelmoinnin menetelmien tuntemus. Ilman näitä ominaisuuksia voi UML:n ja prosessimallien käyttö aiheuttaa enemmän ongelmia kuin hyötyä.

Esitellyt mallit tarjoavat verkkoversioidensa kautta uudenlaisen lähestymistavan itse prosessien kuvaamiseen. Ne myös tuovat huomattavia parannuksia esimerkiksi tiedon saatavuuteen, jakeluun ja prosessien räätälöintiin. Dokumentointiin tarkoitettujen mallien ja esimerkkien avulla on mahdollista saada nopeasti kuva siitä, mitä artefakteja tai tuotteita kuuluu eri projektin vaiheisiin.

Vaikka UML sisältääkin vielä ”lastentauteja”, on se tehokas apuväline mallinnukseen. Sen perusrakenne on kehitetty vuosien mallinnuskokemusten pohjalta, ja sen jatkokehityksestä huolehtii riippumaton organisaatio. Näin sen puutteet hioutuvat varmasti kokemusten kautta pois.

UML ei kuitenkaan ole mikään ongelmien automaattinen ratkaisu. Sitä käytettäessä täytyy muistaa, ettei kaikkea pystytä mallintamaan sen avulla. Koko ajan on pidettävä selkeänä mielessä kysymykset siitä, mitä varten tuotettua mallia ollaan tuottamassa ja kenelle malli on tarkoitettu. Kun tehtyyn malliin saadaan oikea abstraktiotaso, tarjoaa UML tehokkaan keinon tehdä ohjelmiston rakennuspiirrustukset.

Tässä tutkielmassa UML:ää ja prosessimalleja lähestyttiin kirjallisuuden ja käyttökokemuksien pohjalta. Käyttökokemuksia eri mallien ja UML:n avulla tehtävästä ohjelmistosuunnittelusta ei ole kertynyt kovin paljon ja tämä näkyy varmaan tutkielmassa. Aihe olisi mahdollistanut lähestymisen myös eri näkökannoista. Tutkimuksessa olisi voinut esitellä esimerkkiprojektin, jonka avulla olisi saattanut havainnollistaa prosessien ja mallien käyttöä. UML:n semantiikka itsessään olisi ollut riittävä tutkimuksen aiheeksi. Mahdollista olisi tehdä myös vertailua tässä esiteltyjen prosessimallien ja perinteisempien mallien, kuten esimerkiksi vesiputousmallin kesken. Kiinnostava lähestymistapa aiheeseen olisi myös

UML:n laajentamismekanismien käytettävyyden tutkiminen tai prosessimallien räätälöinnin tutkiminen.

VIITELUETTELO

Boehm, B.:Anchoring the Software Process, *IEEE Software,* **13** (4):73--82, 1996.

Booch, G., Rumbaugh, J. and Jacobson, I.: *The Unified Modeling Language User Guide,* Addison-Westley, 1999.

Conallen, J.: Modeling Web Application Architectures with UML. *Communications of the ACM,* **42**(10):63--70, 1999.

Curtis, B. and Paulk, M.: Creating a software process improvement program, *Information and Software Technology*, **35** (6/7):381--386, 1993.

EDS Corporation (2001). *GSMS home page*. Internet WWW-sivu, https://www.gsms-am.eds.com/ (9.5.2001).

Engels, G. and Groenewegen, L.: Object-Oriented Modeling: A Roadmap, *International Conference on Software Engineering, June 4-11, 2000, Limerick, Ireland,* 105--116.

Fuggetta, A.: Software Process: A Roadmap, *International Conference on Software Engineering, June 4-11,, Limerick, Ireland, 2000,* 27--34.

Graham, D.: Incremental development: review of nonmonolithic life-cycle development models, *Information and Software Technology,* **31** (1):7--20, 1989.

Jacobson, I., Booch, G. and Rumbaugh, J.,: *The Unified Software Development Process,* Addison-Westley, 1999.

Jäger, D., Schleicher a., Westfechtel B.: Using UML for Software Process Modeling*, Proceedings of the 7th European Engineering Conference held jointly with the 7th ACM SIGSOFT symposuim on Foundations of software engineering September 6 - 10, Toulouse France*, *1999,* 91--108.

King, S. and Galliers, R.: Modelling the CASE process: empirical issues and future directions, *Information and Software Technology,* **36** (10):587--596, 1994.

Kruchten, P.: *From Waterfall to Iterative Lifecycle - A tough transition for project managers,* Internet WWW-sivu, URL:http://www.rational.com/products/whitepapers/102016.jsp (9.5.2001).

Kruchten, P.: The “4+1” View Model of Sofware Architecture. *IEEE Software,* **12** (6):42--50, 1995.

Kruchten, P.: *The Rational Unified Process - An Introduction Second Edition,* Addison-Westley, 2000.

Kruchten, P.: Rational *Unified Process - Best Practices for Software Development Teams*, Internet WWW-sivu, URL:http://www.rational.com/products/whitepapers/100420.jsp (9.5.2001).

Marshall, C.: *Enterprise Modeling with UML,* Addison-Westley, 1999.

Project Management Institute. *A Guide to the Project Management Body of Knowledge*, Internet WWW-sivu, URL:http://www.pmi.org/publictn/pmboktoc.htm, (11.5.2001)

Quatrani, T.: *Visual Modeling with Rational Rose 2000 and UML,* Addison-Westley, 2000.

Rational. *RUP evaluation*. Internet WWW-sivu, URL:http://www.rational.com/products/rup/tryit/eval/gen_eval.jsp, (9.5.2001).

Rational. *Success stories*. Internet WWW-sivu, URL:http://programs.rational.com/success/, (9.5.2001).

Royce, W.:, Managing the development of large software systems: Concepts and techniques, *Proceedings of IEEE WESTCON, Los Angeles, CA,* 1--9., 1970.

Rumbaugh, J., Jacobson, I. and Booch, G.: *The Unified Modeling Language Reference Manual,* Addison-Westley, 1999.

Selic, B.: Turning Clockwise: Using UML in the Real-Time Domain. *Communications of the ACM,* **42**(10):46--54, 1999.

Simons, A and Graham, I.:37 Things that Don’t Work in Object-Oriented Modelling with UML, *Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications), Friday, July 24th, Brussels, Belgium, 1998,* 209--232.

Tanuan, M.:, Software Architecture in the Business Software Domain: The Descartes Experience, *Proceedings of the third international workshop on Software architecture November 1 - 5, Orlando, FL USA, 1998*, 145--148.