Mikko Taittonen

UML:n dynaamisten mallien integrointitarkastelu metamallien avulla

Tietojärjestelmätieteen
Pro gradu -tutkielma
20.6.2007

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Taittonen, Mikko Antti Jaakko
UML:n dynaamisten mallien integrointitarkastelu metamallien avulla / Mikko Taittonen
Jyväskylä: Jyväskylän yliopisto, 2007
63 s.
Pro gradu -tutkielma

Tämä tutkielma käsittelee UML:n dynaamisia malleja. UML on kuvauskieli ohjelmistojen ja tietojärjestelmien suunitteluun. Dynaamiset mallit ovat malleja jotka kuvaavat järjestelmän toimintaa ja käyttäytymistä. UML:n dynaamisia malleja ovat toimintamalli, tilamalli, yhteistoimintamalli ja sekvenssimalli. Tämän tutkielman tavoitteena on selvittää näiden mallien välisiä johdonmukaisuusongelmia ja päällekkäisyyksiä sekä dynaamisten mallien kehittymistä UML:n eri versioiden välillä.

Edellä mainituista malleista on tehty metamallit ja näistä metamalleista on muodostettu globaali metamalli käyttäen Batinin, Cerin ja Navathen näkymien integrointitekniikkaa. Globaali metamalli sisältää jäsennyksen UML:n dynaamisen mallintamisen keskeisistä käsitteistä. Metamallien integroinnin yhteydessä saatiin tietoja dynaamisten mallien välisistä ristiriidoista ja päällekkäisyyksistä.

UML:n kehittymistä tutkittiin vertailemalla UML:n versioiden 1.3 ja 2.0 spesifikaatioita keskenään. Vertailun tuloksena saatiin lista dynaamisten mallien uudistuksista.

AVAINSANAT: UML, dynaaminen malli, metamalli, näkymien integrointi

SISÄLLYS

1 JOHDANTO

Unified Modeling Language (UML) on kuvauskieli ohjelmistojen ja tietojärjestelmien suunitteluun. UML on kehitetty kolmen suositun menetelmän synteesinä. Sen pohjana ovat olleet Object Modeling Technique (OMT) (Rumbaugh, Blaha, Premerlani ym. 1991), Boochin menetelmä (Booch 1994) sekä Object-Oriented Software Engineering (OOSE) (Jacobson, Christerson, Jonsson ym. 1992). Tässä tutkimuksessa keskitytään pääasiassa UML:n versioon 1.3, mutta tutkimusta laajennetaan myös uusimman version (2.0) puolelle.

UML:ssä on sekä rakenteellisia että dynaamisia malleja. Rakenteellisilla malleilla kuvataan järjestelmäkokonaisuuksia ja niiden koostumista osista. *Dynaamisilla malleilla* taas tarkoitetaan malleja, joilla kuvataan järjestelmän toimintaa ja käyttäytymistä. Dynaamisia malleja UML 1.3:ssa ovat toimintamalli, tilamalli, yhteistoimintamalli ja sekvenssimalli. UML 2.0:ssa on muutamia uusia mallityyppejä. Näitä ovat kokoava vuorovaikutusmalli ja ajoitusmalli. (Koskimies, Koskinen, Maunumaa ym. 2004) Niitä ei kuitenkaan käsitellä tässä tutkimuksessa.

Toimintamalli on pohjimmiltaan vuokaavio, joka esittää miten kontrollivirta etenee toiminnalta toiselle. Tilamalli taas kuvaa tiloja, joita olio käy läpi elinaikanaan reagoidessaa erilaisiin tapahtumiin. Yhteistoimintamalli ja sekvenssimalli ovat vuorovaikutusta kuvaavia malleja. Ne kuvaavat oliota, niiden välisiä suhteita sekä viestejä joita oliot lähettävät toisilleen. Yhteistoimintamalli korostaa keskenään kommunikoivien olioiden välisiä suhteita. Sekvenssimalli taas korostaa olioden välisten viestien ajallista järjestystä.

Syntyhistoriastaan johtuen UML:ssä on todettu vakavia ongelmia. Selicin, Ramackersin ja Kobrynin (2002) mukaan UML luotiin suuren ja monitaustaisen ryhmän toimesta, jolloin on ollut väistämätöntä, että mukaan tulee monimerkityksellisyyttä ja

käsitteellistä päällekkäisyyttä. Paige, Ostroff ja Brooke (2000) esittävät yhdeksän periaatetta mallinnuskielen suunnitteluun. Näitä periaatteita ovat yksinkertaisuus (simplicity), yksikäsitteisyys (uniqueness), ristiriidattomuus (consistency), saumattomuus (seamlessness), peruutettavuus (reversibility), skaalautuvuus (scalability), tuettavuus (supportability) ja luotettavuus (reliability). Tämän tutkielman kannalta oleellisia näistä ovat yksikäsitteisyys ja ristiriidattomuus. Yksikäsitteisyys tarkoittaa sitä, että yhden mallinnuskohteen voi ilmaista vain yhdellä tavalla. Ristiriidattomuus taas tarkoittaa sitä, että mallinnuskieli saavuttaa tavoitteet, jotka sille on asetettu. Mikä tahansa ominaisuus kielessä, joka ei tue sille asetettuja tavoitteita pitäisi hylätä. Koska UML:n suunnittelussa ei ole ollut muuta tavoitetta kuin mallinnuskielten standardointi, tätä periaatetta on ollut vaikea saavuttaa. Paige ym. ehdottavat että UML suunniteltaisiin uudelleen suunnittelutavoitteen ollessa rationalisointi, jolloin kieli olisi yksinkertaisempi ja siinä olisi vain yksi tapa mallintaa tietty asia.

Simons & Graham (1999) kuvaavat ongelmia joita suunnittelijat ovat kohdanneet käyttäesään UML 1.3:a työssään. Ongelmat on luokiteltu neljään kategoriaan, joita ovat epäjohdonmukaisuudet (inconsistency), monimerkityksellisyydet (ambiquity), riittämättömyydet (adequacy) ja harhaanjohtavuudet (cognitive misdirection). Tämän tutkielman kannalta näistä kolme ensimmäistä ovat relevantteja. Epäjohdonmukaisuudet tarkoittavat sitä että jotkin UML mallien käsitteet ovat ristiriidassa keskenään tai yleisesti hyväksyttyjen käsitteiden kanssa. Monimerkityksellisyydet tarkoittavat sitä, että jotkin UML:n käsitteet ovat niin huonosti määritelty, että suunnittelijat voivat tulkita niitä monella tavalla. Riittämättömyydet taas tarkoittavat sitä, että joitain tärkeitä suunnittelu- ja analyysikäsitteitä ei pystytä ilmaisemaan UML:llä ollenkaan. Epäjohdonmukaisuus ongelmia Simons & Graham (1999) olivat havainneet UML:n dynaamisissa malleissa kaksi kappaletta, monimerkityksellisyyksiä kaksi kappaletta sekä riittämättömyys ongelmia neljä kappaletta.

Tässä tutkimuksessa hyödynnetään metamalleja sekä näkymien integrointitekniikkaa. *Metamalli* on käsitteellinen malli tietojärjestelmien kehittämismenetelmästä tai sen osasta. Metamalleja on useamman tyyppisiä riippuen siitä, mitä tietoa menetelmistä

halutaan mallintaa. Metatietomalli on metamalli, joka kuvaa menetelmän staattisia ominaisuuksia eli menetelmän käsitteellistä rakennetta ja notaatiota (Tolvanen 1998). Tässä työssä metamalli ymmärretään metatietomalliksi. Näkymien integrointi on Batinin ym. (1992) esittämä tekniikka tietokantoja kuvaavien näkymien integrointiin. Siinä pyritään yhdistämään johdonmukaisesti eri suunnittelijoiden tekemät mallit samasta kohdealueesta. Tässä tutkielmassa näkymien integrointia sovelletaan metamallien integrointiin.

Tämän tutkimuksen tavoitteena on tuottaa integroitu metamalli UML:n dynaamisista malleista ja sen avulla tutkia, missä määrin UML:n dynaamiset mallit ovat käsitteistöltään keskenään johdonmukaisia ja missä määrin dynaamiset mallit ovat käsitteellisesti päällekkäisiä. Lisäksi tutkitaan mitä uusia käsitteitä UML:ään on sisällytetty sen uusissa versioissa. Tutkimuksella pyritään vastaamaan seuraaviin kysymyksiin:

1. Ovatko UML:n dynaamiset mallit käsitteistöltään keskenään johdonmukaisia?
2. Missä määrin UML:n dynaamiset mallit ovat käsitteellisesti päällekkäisiä?
3. Mitä uusia käsitteitä UML:ään on tuotu version 1.3 jälkeen?

Ensimmäinen kysymys tarkoittaa sitä, että tutkitaan onko eri mallien samannimisillä käsitteillä myös tarkoitettu samoja asioita. Toisessa taas kysytään sitä, kuinka paljon samoja käsitteitä eri malleissa on. Kolmas kysymys tarkoittaa, sitä miten dynamisen mallintamisen kenttä on muuttunut ajan mittaan. Samalla kartoitetaan käyttäytymisen käsitteitä laajemmalla alueella.

Tutkimus suoritettiin kahdessa jaksossa. Tutkimus oli aloitettu kun UML 1.3 on ollut ajankohtainen, ja se on lopetettu kun 2.0 on ollut UML:n uusin versio. Näin ollen kahteen ensimmäiseen tutkimuskysymykseen on vastattu 1.3:n perusteella. Koska UML oli kehittynyt tutkimuksen aikana, haluttiin tutkimusta laajentaa myös uusimpaan UML:n versioon.

Tutkielma rakentuu seuraavasti. Luvussa 2 esitellään metamallinnusta ja metamallien integrointia yleisesti. Luvussa 3 kuvataan metamallinnuskielen ja -välineen valintaa sekä UML:n dynaamisia malleja yleisesti. Lisäksi esitetään metamallit UML:n dynaamisista malleista. Luvussa 4 esitellään tarkemmin integrointimenetelmä, ja kuvataan miten yksittäiset metamallit on integroitu toisiinsa. Integroinnin yhteydessä selvitetään mallien johdonmukaisuus- ja päällekkäisyysongelmat. Luvun lopussa esitetään globaali metamalli UML:n dynaamisista malleista. Luvussa 5 UML:n kehittymistä on tutkittu vertailemalla UML:n eri versioiden kirjallisia kuvauksia toisiinsa. Tutkielma päättyy luvun 6 yhteenvetoon.

2 MALLIEN INTEGROINTI METAMALLIEN AVULLA

Tämän luvun tarkoituksen on antaa yleiskuva metamalllinnuksesta ja metamallien integroinnista. Lisäksi kerrotaan näiden tekniikoiden sovelluksista.

2.1 Metamallinnus

Metamallinnus on monipuolinen teknologia, jota voidaan soveltaa monella tietojenkäsittelyn alueella. Yksinkertaisimmillaan metamalli voidaan määritellä malliksi mallista. Metamallien luontia kutsutaan metamallintamiseksi. Metamallinnuksen sovelluskohteita ovat mm. tietojärjestelmien kehittämismenetelmät, tietokannat, ohjelmointikielet ja monet muut tietotekniikan alat. Tässä luvussa kerrotaan metamallinnuksesta pääasiassa tietojärjestelmäsovellusalueella.

Metamallien luonnin mahdollistavat metamallinnuskielet. Metamallinnuskielillä kuvataan suunnittelumenetelmiä saman tapaan kuin suunnittelumenetelmillä kuvataan tietojärjestelmiä. Tolvanen (1998) luettelee muutamia metamallinnuskieliä: ASDM (Heym & Österle 1992), CoCoA (Venable 1993), ER (Chen 1976), GOPRR (Smolander 1993), MEL/MDM (Harmsen 1997), NIAM (Nijssen & Halpin 1989) ja OPRR (Smolander 1991). Tolvasen mukaan suurin osa näistä kielistä on ER-pohjaisia. Poikkeuksena on NIAM, joka on olio-pohjainen, sekä MEL joka perustuu ensimmäisen asteen predikaattilogiikkaan. Tällä hetkellä yksi merkittävimmistä metamallinnuskielistä on Meta Object Facility (MOF) (OMG 2000a). Object Management Group (OMG) käyttää sitä esimerkiksi UML:n metamalleissa. Käytännössä MOF on yksinkertaistettu versio UML:n luokkakaaviosta.

Metamalleja voidaan selittää erilaisten tasojärjestelmien kautta. Kleppe, Warner ja Bact (2003) selvittävät mitä mallit ja tasojärjestelmät tarkoittavat OMG:n arkkitehtuurissa. Heidän mukaan malli on kuvaus järjestelmästä jollain hyvin määritellyllä kielellä. Metamallintaminen on mekanismi, jolla voidaan kuvata em. hyvin määriteltyjä kieliä. Metamalleissa käytettävät käsiteet määrittelee metametamalli.

Taulukossa 1 on kuvattu OMG arkkitehtuurin tasot Kleppen, Warnerin ja Bactin (2003) mukaan. Tasoja ovat instanssitaso (M0), mallitaso (M1), mallin malli –taso (M2) sekä M2 -mallin taso (M3). Tasojen käsitteiden välillä on instance of –suhde. Instanssitasolle kuuluvat esim. ohjelman luokkien ajonaikaiset instanssit ja tietokantojen varsinainen sisältö. Kaikki mahdolliset instanssit on määritelty M1 -tasolla esim. UML:n luokka-käsitteen avulla. M2 -tasolla taas määritellään minkälainen voi olla UML:n luokkakäsite. Siis että luokkaan kuuluvat nimi, attribuutit, operaatiot ja niin edelleen. M2 -tason käsitteet määritellään M3 -tasolla metametakielellä eli tässä tapauksessa MOF:lla.

TAULUKKO 1. OMG arkkitehtuurin tasot Kleppen, Warnerin ja Bactin (2003) mukaan

OMG arkkitehtuurin tasot	UML -käsite
M0	Asiakas-instanssi: Mark Everyman
M1	Asiakas-luokka
M2	UML-luokka
M3	MOF-luokka

Jarke, Klamma & Lyytinen (2003) esittelevät ISO:n Information Resource Dictionary Standard (IRDS) -standardin (ISO 1990). Kyseinen standardi jakaa tiedon ja sitä kuvaavat mallit neljään päällekkäiseen tasoon. IRDS:n alin taso on sovellustaso (Application Level). Sovellustaso sisältää tarkasteltavana olevan järjestelmän varsinaiset tiedot. Tämä voi tarkoittaa esim. ohjelman ajonaikaisia luokkien instansseja, tietokannan sisältöä tai jollain suunnittelumenetelmällä kuvattujen mallien instansseja. Sovellustason yläpuolella on metatietotaso (IRD Level). Metatietotasolla ovat esim.

tietokoneohjelma, tietokannan skeema tai suunnittelumenetelmällä tuotettu malli. Tietokoneohjelma määrittelee minkälaisia olioita voi ohjelman ajonaikana syntyä. Tietokannan skeema taas määrää tietokannan sisältämät tiedot, niiden tietotyypit ja suhteet. Suunnittelumenetelmällä kuvatttu malli taas määrittelee minkälaisia käsite-instansseja tietojärjestelmän osa-alueella voi olla. Metatietotason yläpuolella on metamallitaso (IRD Definition Level). Metamallitaso sisältää esim. ohjelmointikielen määrityksen, tietokannan tietomallin ja suunnittelumenetelmän kuvauksen. Ylimpänä tasona IRDS:ssä on metametamallitaso (IRD Definition Schema Level). Tälle tasolle kuuluvat menetelmät, joilla kuvataan esim. ohjelmointikieliä ja suunnittelumenetelmiä. Taulukossa 2 on annettu esimerkkejä IRDS:n eri tasojen sisällöistä.

TAULUKKO 2. IRDS:n tasot esimerkkeineen

IRDS taso	Ohjelmointikielet	Tietokannat	Suunnittelumenetelmät
Application Level	Unit #00001	Mikko	Rahtikirja #234234
IRD Level	Unit-luokka	Käyttäjätaulu	Rahtikirja-luokka
IRD Definition Level	Java-kielen määritys	Relaatiomalli	UML-standardi
IRD Definition Schema Level	BNF-notaatio	Joukko-oppi	MOF

Metamallien avulla voidaan kuvata suunnittelumenetelmiä monesta eri näkökulmasta. Jarke, Klamma ja Lyytinen (2003) esittelevät Jarken ym. (1998) timanttimalliin, joka on kaksiosainen viitekehys metamallien luokitteluun. Ensimmäinen osa koostuu eri näkökulmista eli ontologioista, notaatioista ja prosesseista ja toinen osa järjestelmän kehityksen tavoitteista. Järjestelmän kehityksen tavoitteet ohjaavat metamallinnuksessa tehtyjä valintoja. Ontologioilla tarkoitetaan tietojärjestelmän sovellusalueen peruskäsitteitä, notaatiolla järjestelmien kuvaustapaa ja prosesseilla käytäntöjä joilla tietojärjestelmämallit saadaan aikaiseksi.

Tolvasen (1998) (s. 81) mukaan metamallit voidaan jakaa eri tyyppeihin sen mukaan minkälaista menetelmätietoa ne mallintavat. Esimerkiksi metatietomalli on metamalli, joka mallintaa menetelmän staattisia piirteitä. Metamalleja voidaan käyttää myös prosessien kuvaukseen. Koskinen ja Marttiin (1997) kuvaavat prosessien tukea MetaCASE työkalussa metamallitasojen avulla. Jokaisella menetelmäkuvauksen tasolla on staattisia piirteitä kuvaavan mallin lisäksi prosessia kuvaava malli. Esimerkiksi kehittämistasolla tietojärjestelmäkuvauksien prosessivastine on tietojärjestelmän kehittämisen prosessi. Esimerkiksi UML kuvauksia vastaa Rational Unified Process (RUP) (Kruchten 2003) prosessi. Menetelmän kuvaustasolla UML:n kuvausta vastaa RUP menetelmän kuvaus ja niin edelleen. Prosessien metamalleilla kuvataan suunnittelumenetelmään liittyvä prosessi ja integroidaan prosessin käsitteet menetelmän staattisiin käsitteisiin.

Leppäsen (1994) mukaan malli voi olla rakenteellinen tai dynaaminen. Mallin tyypin mukaan määräytyy edelleen metamallin tyyppi. Metarakennemalli kuvaa rakenteellisia malleja ja metaprosessimalli kuvaa dynaamisia malleja.

2.2 Metamallien integrointi

Metamallien integrointi on yksinkertaisesti sitä, että useampi metamalli yhdistetään yhteisten käsitteiden avulla yhdeksi tai useammaksi yhdistetyksi metamalliksi. Metamallien integrointi perustuu yleensä johonkin tekniikkaan, kuten Batinin ym. (1992) esittämään näkymien integrointitekniikkaan. Batinin lähestymistapaa tai sen kaltaista lähestymistapaa ovat käyttäneet mm. van Hillegersberg & Kumar (1999), Pastor & Price (1997) ja Henderson-Sellers & Bulthuis (1997). Batinista selvästi eroavaa integrointitekniikkaa on käyttänyt Saeki (1995). Saekin tekniikassa menetelmä tai sen osa määritellään luokkana, jossa menetelmän rakenne kuvataan luokan attribuutteina ja menetelmän prosessi kuvataan luokan metodeina. Itse integrointi tapahtuu menetelmäluokkien välisten assosiaatioiden ja perinnän avulla.

Metamallien integroinnille tavallaan käänteinen menetelmä on ontologioiden muodostus. Ontologioiden muodostuksessa pyritään etsimään jonkin järjestelmäkehityksen tai liiketoiminta-alueen keskeiset käsitteet, ja muodostamaan näistä käsitteistä ydinmalli. Esimerkiksi Halttunen, Lehtinen ja Nykänen (2006) ovat muodostaneet käsitteellisen kehyksen yritysarkkitehtuurien kuvaukseen. Kehys on saatu aikaan analysoimalla yritysarkkitehtuurialuetta kokonaisuutena, ja etsimällä alueen keskeisiä käsitteitä olemassa olevista standardeista, sekä keskustelemalla käytännön mallintajien kanssa. Käsiteanalyysejä ja keskusteluja iteroimalla on lopuksi muodostettu käsitteellinen malli yritysarkkitehtuurien olennaisimmista käsitteistä. Käänteinen menetelmä yritysarkkitehtuureita kuvaavan mallin muodostukselle olisi ollut metamallintaa yritysarkkitehtuureita eri näkökulmista kuvaavat mallit, ja integroida ne yhdeksi metamalliksi. Tämän menetelmän tuloksena syntynyt malli olisi kuitenkin keskittynyt liikaa olemassa oleviin käsitteisiin ja sitä olisi ollut vaikea laajentaa.

2.3 Metamallinnuksen ja metamallien integroinnin sovelluksia

Metamallinnuksen sovellukset voidaan jakaa esimerkiksi seuraaviin osa-alueisiin: menetelmien kuvaus, menetelmien analysointi ja menetelmien kehittäminen. Perinteisin metamallinnuksen sovellusalue on ollut menetelmien kuvaus. Menetelmien kuvauksella voidaan tarkoittaa esimerkiksi olemassa olevan tekstikirjamenetelmän kuvausta case-välineeseen tai uuden menetelmän kuvausta käyttämällä metamalleja. Esimerkkinä ensimmäisestä on Metaedit+ (Kelly, Lyytinen & Rossi 1996) ja jälkimmäisestä UML:n määritykset (OMG 2000b).

Toinen metamallinnuksen sovellusalue on menetelmien analysointi. Metamallien avulla saadaan eri menetelmät kuvattua samoilla yhteismitallisilla käsitteillä, jolloin eri menetelmiä on helppo vertailla ja analysoida. Esimerkiksi Tolvanen & Rossi (1996) ovat metamallintaneet 12 eri menetelmää, ja analysoineet metamalleja tilastollisin menetelmin.

Menetelmien analysointia voidaan tehdä myös metamalleja integroimalla. COMMA projektissa (Henderson-Sellers & Bulthuis 1997) tutkittiin kuinka neljässätoista eri oliomenetelmässä oli kuvattu staattisia piirteitä, luokkien välistä yhteistoimintaa sekä yksittäisen luokan tilasiirtymiä. Tutkimuksessa koottiin ydinmallit menetelmien staattisista käsitteistä sekä tilasiirtymiä koskevista käsitteistä. Pasto & Price (1997) ovat tutkineet uudelleenkäytön tukea neljässä eri menetelmässä. Tutkimuksessa on aluksi metamallinnettu yksittäisen menetelmän kaikki eri mallit, ja lopuksi integroitu ne koko menetelmää kuvaavaksi ydinmalliksi. Tämän jälkeen on metamallinnettu yleisimmät uudelleenkäyttöön liittyvät käsitteet, ja verrattu tätä metamallia tutkittavien menetelmien ydinmalleihin samankaltaisuuksien löytämiseksi.

Myös van Hillegerberg ja Kumar (1999) ovat käyttäneet metamalleja ja niiden integrointia menetelmien analysointiin. Heidän tehtävänään on ollut vetää yhteen kokonaiskuvaa olio-keskeisen järjestelmäkehityksen käsitteistä. Tämä on tapahtunut tekemällä metamallit kolmen eri olio-ohjelmointikielen käsitteistä ja integroimalla nämä käsitteet yhdeksi ydinmalliksi. Ohjelmointikieliä koskeva yhdistetty metamalli on integroitu COMMA-projektissa tuotettuun metamalliin. Näin on saatu aikaiseksi metamalli, joka kattaa olio-käsitteitä sekä suunnittelumenetelmien että ohjelmointikielien alueilta. Tämän yhdistetyn mallin avulla on voitu esimerkiksi analysoida olio-menetelmien ja –kielien välisiä eroja ja yhtäläisyyksiä.

Metamallit ovat mahdollistaneet myös vaivattoman ja tilannekohtaisen menetelmien kehittämisen ja eri menetelmien yhdistämisen. Tolvanen (1998,191) kuvaa menetelmien kehittämismenetelmän, jossa suunnittelumenetelmiä muokataan projekteista saadun palautteen mukaan. Olennainen osa Tolvasen kuvaamaa menetelmää on tietojärjestelmämenetelmien kuvaaminen ja muokkaaminen metamallien avulla.

Toinen tapa hyödyntää metamalleja menetelmien muokkauksessa on yhdistää olemassa olevia menetelmiä uusiksi suunnittelumenetelmiksi. Leppänen (2000) kuvaa menetelmien kehittämistekniikan, jolla eri menetelmien osia voidaan koota yhteen samalla huomioiden syntyvän menetelmän johdonmukaisuus. Leppäsen menetelmän ytimenä on eri menetelmien metamallien integrointi Batinin ym. (1992) tekniikalla.

Myös Saeki (1995) kuvaa uusien menetelmien luontia menetelmäkappaleita yhdistelemällä.

2.4 Yhteenveto

Tässä luvussa tutustuttiin metamallinnukseen, metamallien integrointiin ja näiden teknologioiden sovelluksiin. Metamallinnuksen keskeisistä käsitteistä esiteltiin metamallinnuskielet sekä metamallinnustasot. Näin pyrittiin luomaan pohjaa luvussa 3 kuvatulle dynaamisten mallien metamallinnukselle.

Metamallien integrointi kuvattiin yleisesti ja samalla mainittiin kaksi toisistaan poikkeavaa integrointitekniikkaa. Käsitystä metamallien integroinnista pyrittiin laajentamaan kuvamalla sille käänteinen menetelmä eli ontologioiden muodostus. Metamallien integroinnin esittely luo pohjaa luvussa 4 tapahtuvalle UML:n dynaamisten mallien metamallien integroinnille.

Tässä luvussa kuvattiin myös erilaisia metamallinnuksen ja metamallien integroinnin sovelluksia. Sovelluksia esittelemällä haluttiin osoittaa, että tässä tutkielmassa käytettyä menetelmää on sovellettu muuallakin.

3 UML:N DYNAAMISTEN MALLIEN METAMALLINNUS

Tässä luvussa kuvataan metamallinnustyön toteutusta siihen liittyvien tavoitteiden ja valintojen kautta. Metamallinnustyöhön liittyviä valintoja ovat mallinnettavien käsitteiden valinta sekä mallinnustyökalujen valinta. Toteutukseen liittyvien seikkojen jälkeen jokaisesta UML:n dynaamisesta mallista annetaan yleiskuvaus esimerkkien kera. Yleiskuvauksen jälkeen kuvataan mallien keskeisimmät käsitteet ja niiden väliset suhteet tarkemmalla tasolla. Mallien keskeisimmät käsitteet ja niiden väliset suhteet esitetään lopuksi tiivistettynä metamallikaavioiden avulla.

3.1 Metamallinnuksen tavoitteet ja toteutus

Tässä kohdassa kuvataan metamallinnuksen tavoitteita ja toteutusta. Metamallien toteutuksen yhtenä ongelmana oli mallinnettavien käsitteiden valinta. Toisena ongelmana oli mahdollisimman käyttökelpoisen mallinnuskielen ja sitä tukevan kielen valinta.

3.1.1 Tavoitteet

Metamallinnuksen tavoitteena oli tuottaa jäsennykset yksittäisten dynaamisten mallien käsitteistä. Näitä jäsennyksiä käytetään syötteenä metamallien integroinnissa, jonka avulla voidaan tutkia mallien välisiä päällekkäisyyksiä ja epäjohdonmukaisuuksia. Metamallit pyrittiin luomaan mahdollisimman tarkasti ja laajasti, jotta integrointitarkastelussa voitaisiin tehdä mahdollisimman paljon havaintoja.

3.1.2 Mallinnettavien käsitteiden valinta

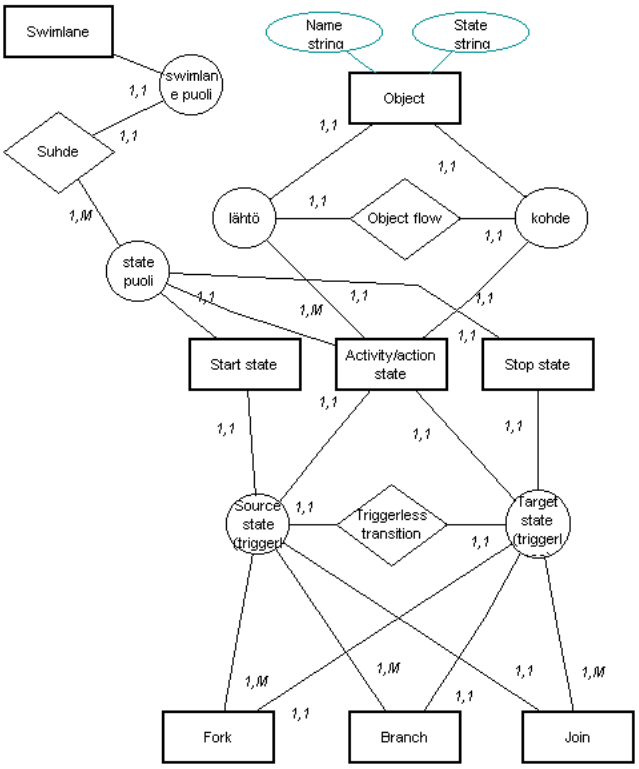
Mallinnettavien käsitteiden valinnalle ei ollut selkeää perustetta. Mukaan otettiin ne käsitteet, jotka esiintyivät UML:n käyttäjäoppaan (Booch, Rumbaugh & Jacobson 1999) ja UML:n manuaalien (Rumbaugh, Jacobson & Booch 1999) esimerkeissä. Oletettiin että nämä ovat yleisimmät UML:n dynaamisten mallien käsitteet. Vastaavanlaisen valintatilanteen edessä ovat ollet myös van Hillegersberg ja Kumar (1999, 119). Heidän tehtävänään oli mallintaa ja integroida erilaisia olimallinnustekniikoita ja –kieliä. Tässä työssään he joutuivat tekemään valintoja mallinnettavista käsitteistä. Heidän nyrkkisääntönä oli että kaikki käsitteet, jotka saivat merkittävää huomiota menetelmän kuvauksessa, päätyivät metamalliin. Van Hillegersbergin ja Kumarin mallinnustyötä ohjasivat myös Welken (1992) esittelemät yhdenmukaisuuden (consonance) ja häviämättömyyden (no loss) periaatteet. Yhdenmukaisuus tarkoittaa lyhyesti sitä, että mallintajan havaintojen ja mallinnuskäsitteiden välillä pitäisi olla 1:1 suhde. Yksi mallinnuskäsite kuvaa siis yhtä mallinuskohdetta. Häviämättömyys taas tarkoittaa sitä, että metamallin tulisi olla sellainen, että mallinnettavasta ilmiöstä ei kadoteta tietoa.

3.1.3 Metamallinnuskielen ja – välineen valinta

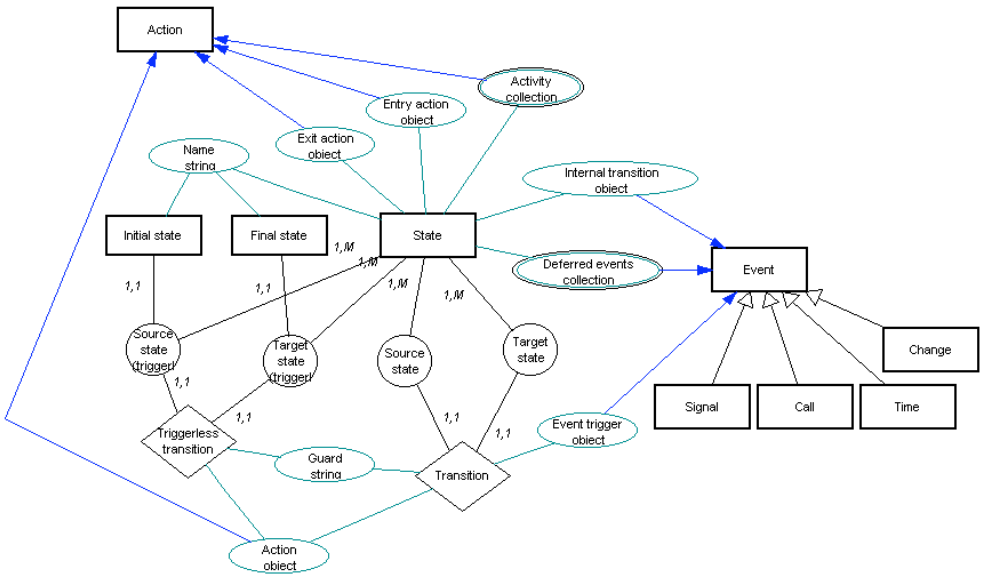
Metamallinnustyö aloitettiin valitsemalla metamallinnuskieli. Vaihtoehtoja oli kolme: GOPRR (Smolander 1993), ER-mallinnuskieli (Chen 1976) ja UML (OMG 2000b). Valintakriteereinä olivat kuvauskieleen liittyvä osaaminen, kuvauskielen välinetuki, kuvauskielen sopivuus UML:n mallintamiseen ja yhteensopivuus Batinin integrointitekniikan kanssa.

Ensimmäisenä tutkittavaksi otettiin GOPRR, jota on käytetty menestyksekkäästi monien menetelmien mallintamiseen. GOPRR:ää kokeiltiin muutamien UML:n dynaamisten

mallien mallintamiseen. Kuvioissa 1 ja 2 on kuvattu toiminta- ja tilamallien metamallit GOPRR:llä.



KUVIO 1. Toimintamallin metamalli GOPRR kielellä kuvattuna



KUVIO 2. Tilamallin metamalli GOPRR kielellä kuvattuna

GOPRR:n käyttö kuitenkin hylättiin, koska kuvioista tuli liian monimutkaisia kuvauskielen vaatimien eksplisiittisten roolikäsitteiden vuoksi. Toiseksi Batinin ym. (1992) integrointitekniikka ei tue roolien käsitettä. Kolmanneksi GOPRR:ää tukevasta MetaEdit+ -välineestä (Metacase 2006) ei saatu tutkielman tekoon soveltuvaa lisenssiä.

Myös ER-mallinustekniikkaa harkittiin. Sen etuna olisi ollut sen suora soveltuvuus Batinin integrointitekniikkaan. ER tekniikkaa ei kuitenkaan valittu, koska sille ei löytynyt sopivaa ilmaista mallinnusvälinettä. ER-tekniikan haittapuolena oli myös se, että se ei ollut mallintajalle tarpeeksi tuttu.

Loppujen lopuksi metamallinuskieleksi valittiin UML:n luokkakaaviotekniikka. Se on mallintajalle tuttu, sitä tukevia työkaluja löytyi runsaasti ja sitä pystyi soveltaen käyttämään Batinin integrointitekniikan kanssa. UML:n luokkakaaviosta löytyy myös MOF (OMG 2000a) niminen muunnos, jota on käytetty erityisesti metamallinnukseen. Sitä on käytetty pääasiassa OMG:n omisssa spesifikaatioissa. MOF:ia ei kuitenkaan käytetty tässä tutkielmassa, koska se ei ollut mallintajalle tarpeeksi tuttu, eikä kyseiselle menelmälle löytynyt ilmaista välinetukea.

UML:n välinepariksi kokeiltiin aluksi AgroUML (Tigris.org 2007) –nimistä ilmaista java-pohjaista työkalua. AgroUML:n ongelmina olivat mallinnusjankohtana välineen hitaus ja kömpelö käyttöliittymä. Mallinnusvälineeksi valittiin loppujen lopuksi Objecteering/UML väline (Objecteering Software 2006), joka oli ilmainen ja tarpeeksi tehokas väline mallinnusurakkaan.

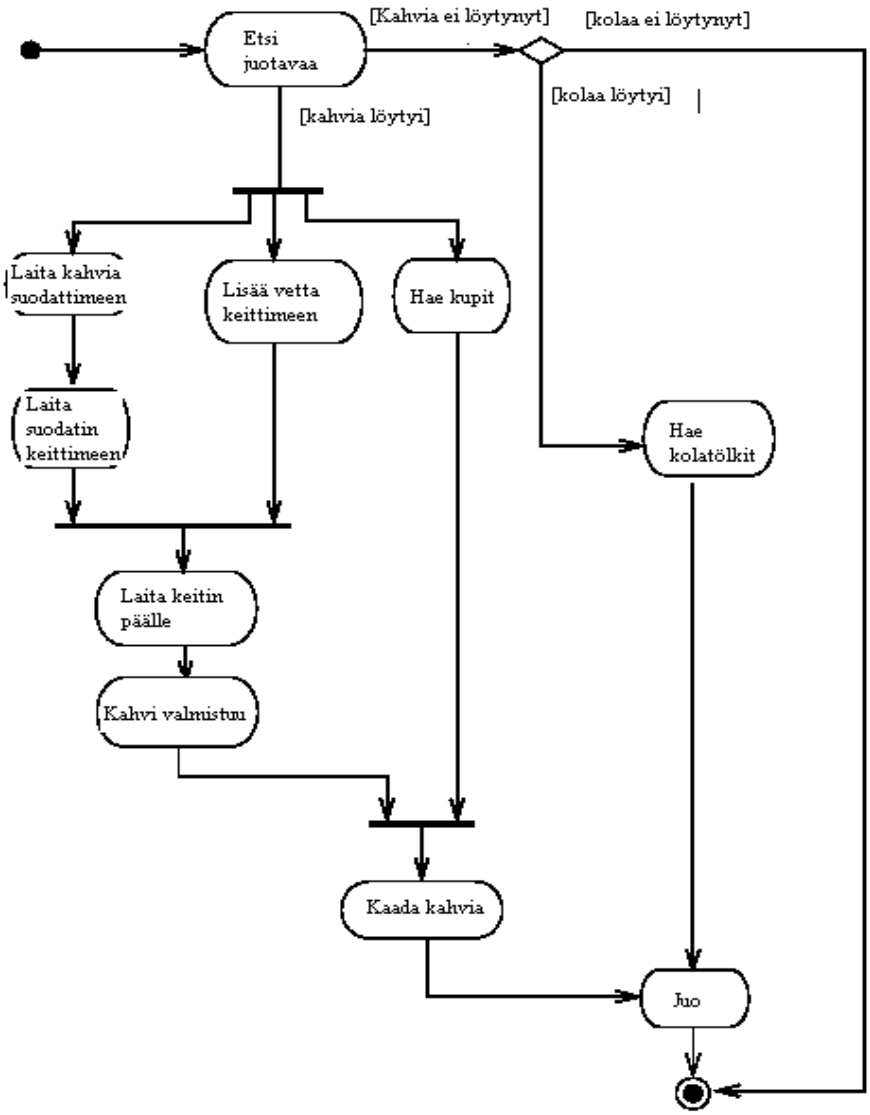
3.2 Toimintamalli

Tässä kohdassa kuvataan UML:n toimintamalli yleisellä tasolla esimerkkejä avuksi käyttäen. Yleiskuvauksen jälkeen kuvataan toimintamallin keskeisimmät käsitteet tarkemmin ja esitetään toimintamallin metamalli.

3.2.1 Yleiskuvaus

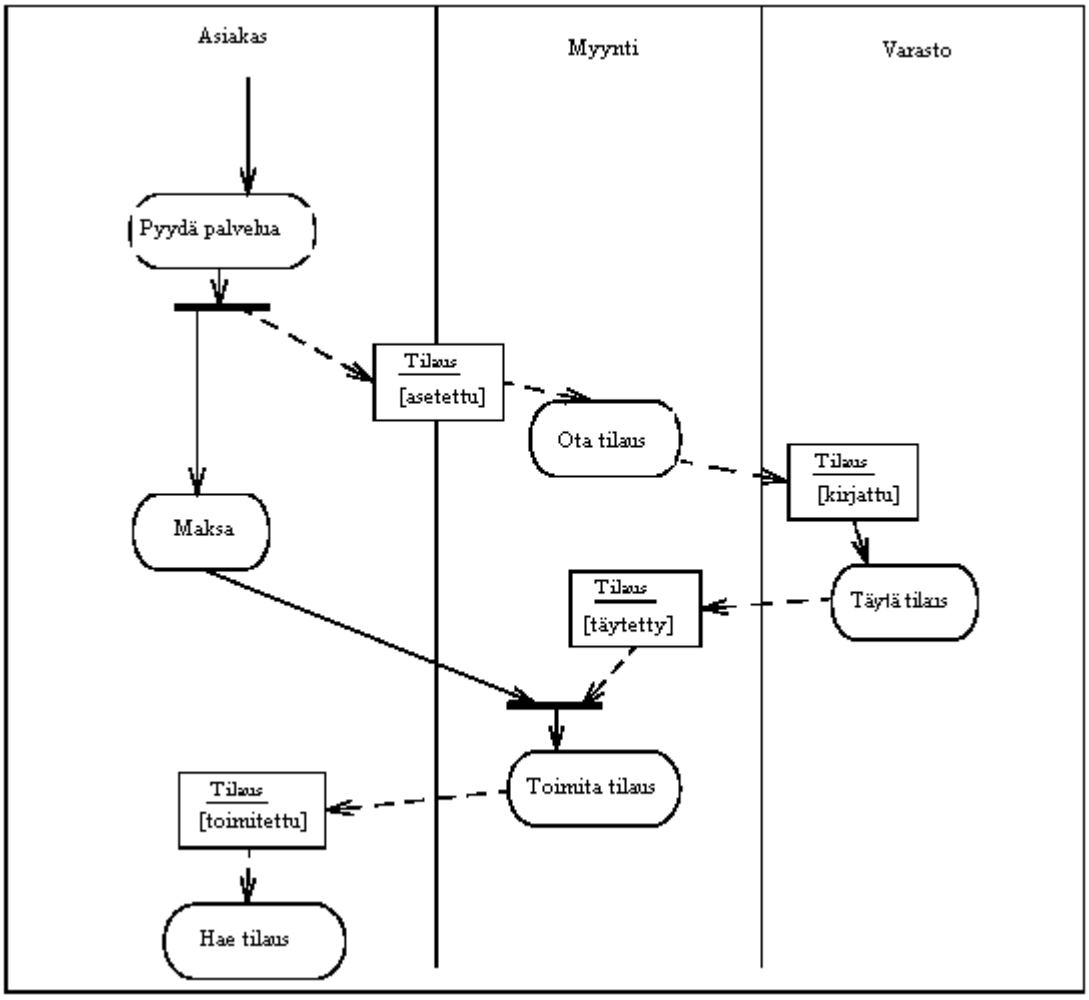
Toimintamalli (activity diagram) on pohjimmiltaan vuokaavio, joka kertoo miten kontrollivirta etenee toiminnalta toiselle. Toimintamallia käytetään kuvaamaan järjestelmän toiminnallisia piirteitä. Yleensä tämä tarkoittaa laskennallisen prosessin peräkkäisten ja mahdollisesti rinnakkaisten askelien mallintamista. Toimintamallilla voidaan kuvata myös olion siirtymistä tilasta toiseen kontrollivirran eri kohdissa. Toimintamalleja voidaan käyttää mallintamaan olioyhteisön dynamiikkaa tai niitä voidaan käyttää mallintamaan jonkin operaation kontrollivirtaa. Kun vuorovaikutusta kuvaavat mallit (interaction diagrams) eli sekvenssi- ja yhteistoimintamallit korostavat kontrollivirran etenemistä oliolta oliolle, toimintamalli painottaa kontrollivirran etenemistä toiminnalta toiselle. (Booch, Rumbaugh & Jacobson 1999, 257)

Kuviossa 3 on esimerkki toimintamallin käytöstä. Kyseinen esimerkki kuvaa juoman hankkimiseen liittyviä askelia. Esimerkin kuvaama toiminta lähtee liikkeelle aloitustilasta, joka on merkitty mustalla pisteellä. Aloitustilasta on siirtymä ’Etsi juotavaa’ -tilaan. Siirtymää merkitään nuolella ja tilaa kulmista pyöristetyllä suorakulmiolla. ’Etsi juotavaa’ -tilasta voidaan edetä kahta vaihtoehtoista siirtymää pitkin. Vaihtoehtoisiin siirtymiin liittyvä siirtymäehto on merkitty hakasulkujen sisälle. Jos ’kahvia löytyi’ -ehto on tosi, siirrytään suorittamaan kahvinkeittoon liittyviä useampia rinnakkaisesti suoritettavia toimintoja. Kohtaa, jossa rinnakkaiset toiminnot alkavat, kuvataan vaakasuoralla palkilla. Samalla tavalla kuvataan myös kohtaa, jossa rinnakkaiset toiminnot loppuvat. Kahvin keittoa kuvaavien rinnakkaisten tilojen sekä ’Kaada kahvia’ ja ’Juo’ –tilojen kautta siirrytään lopetustilaan. Lopetustilaa merkitään mustalla ympyröidyllä pisteellä. Jos ’Etsi juotavaa’ –tilan toisen siirtymän ehto on tosi, siirrytään haaraumaan. Vinoneliöllä merkityn haarauman kohdalla tehdään valinta ’kolaa löytyi’ ja ’kolaa ei löytynyt’ -siirtymien välillä. Ensimmäisen vaihtoehdon toteutuessa siirrytään ’Hae kolatölkit’ ja ’Juo’ –tilojen kautta lopetustilaan, jälkimmäisestä taas siirrytään suoraan lopetustilaan.



KUVIO 3. Juoman hankintaan liittyvä toimintamallin esimerkki (OMG 2000b).

Kuviossa 4 on toinen esimerkki toimintamallin käytöstä. Esimerkki kuvaa tuotteen tilaukseen, toimittamiseen ja maksamiseen liittyvää työnkulkua. Lisäkäsitteinä edelliseen esimerkkiin nähden ovat radat, oliot ja oliovirrat. Radat merkitään kaaviossa kahdella pystysuoralla viivalla, oliot suorakulmiolla ja oliovirrat katkoviivoitetulla nuolella. Radat kuvaavat tässä esimerkissä asiakasta ja henkilökuntaa, oliot käsiteltävää tilausta ja oliovirrat sitä miten henkilökunta tai asiakas käsittelee tilausta.



KUVIO 4. Toimintakaavio tilaustenkäsittelystä (OMG 2000b).

3.2.2 Metamalli

Alla kuvataan edellisen kohdan esimerkeissä esiintyneet käsitteet tarkemmin. Käsitteiden kuvaukset antavat pohjaa metamallin ymmärtämiselle.

Toimintamallin *tilat* (state) kuvaavat toimintaa, eivät olion tiloja (Rumbaugh, Jacobson & Booch 1999, 81). Tiloja voi olla kahdenlaisia: *toimintotiloja* (action state) ja *toimintatiloja* (activity state). Toimintotilat kuvaavat keskeytymätöntä, jakamatonta toimintaa, jolla on merkityksetön kesto. Toimintatilat taas kuvaavat pitempikestoista

toimintaa, joka voidaan jakaa osiin ja myös keskeyttää. Toimintatila voi koostua muista toiminto- tai toimintatiloista. Toimintatiloihin voi liittyä myös aloitus- (entry action) ja lopetustoiminto (exit action) (Booch, Rumbaugh & Jacobson, 261). Toimintamallissa on myös *aloitustiloja* (start state) ja *lopetustiloja* (stop state). Aloitustila on tila, josta toimintamallin kuvaama toiminta alkaa. Lopetustila on tila, jossa toiminta loppuu.

Siirtymä (transition) kuvaa kontrollivirran siirtymistä tilasta toiseen. Toimintakaavioiden yhteydessä siirtymiin ei liity tapahtumia, vaan tilasta toiseen siirrytään, kun tilaan liittyvä toiminto tai toiminta loppuu (Booch, Rumbaugh & Jacobson 1999, 262). Siirtymissä tilasta toiseen saattaa olla välivaiheena kontrollivirran valinta haarauman avulla tai vaihtoehtoisten kontrollivirtojen yhdistäminen yhdistimen avulla. Myös kontrollivirran jakautuminen useammalle säikeelle ja kontrollivirran palautuminen yhdelle säikeelle on mahdollista jakajan ja liittäjän avulla.

Haarauman (branch) avulla valitaan jokin vaihtoehtoisista kontrollivirroista. Haaraumaan liittyy yksi sisääntuleva siirtymä ja kaksi tai useampia ulospäin lähtevää siirtymää. Ulospäin lähtevät siirtymät on varustettu siirtymäehdoin (guard condition). Siirtymäehto on jokin lauseke, jonka totuusarvoa voidaan testata. Kontrollivirta jatkuu sen siirtymän mukaan, jonka siirtymäehto on tosi. *Yhdistin* (merge) esittää pistettä, jossa toiminta jatkuu taas yhden kontrollivirran mukaan. Yhdistimeen liittyy useampia sisääntulevia siirtymiä ja yksi ulospäin lähtevä siirtymä.

Jakajassa (fork) toimintamallin kuvaama toiminta jakautuu useamman säikeen suoritettavaksi. Jakajaan tulee sisään yksi siirtymä, josta toiminta hajaantuu kahteen tai useampaan siirtymään tai oliovirtaan. *Liittäjässä* (join) siirrytään rinnakkaisten säikeiden suorittamasta toiminnnasta yhden säikeen suorittamaan toimintaan. Liittäjässä yhdistyy kaksi tai useampia siirtymiä tai oliovirtoja, ulospäin lähtee yksi siirtymä.

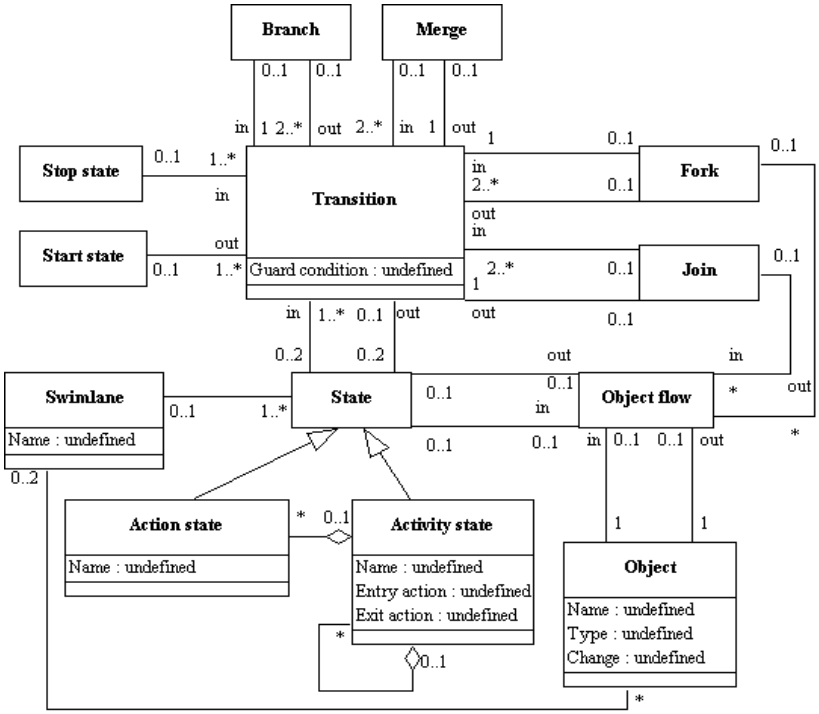
Toimintamallissa voidaan kuvata myös *olioita* (object), joita eri tilat luovat, muuttavat tai tuhoavat. Olio on erillinen kokonaisuus, jolla on hyvin määritellyt rajat ja identiteetti, ja joka käsittää tilan ja käyttäytymisen (Rumbaugh, Jacobson & Booch

1999). Olioista voidaan kertoa nimi, tyyppi sekä se miten olion tila, attribuuttien arvot tai rooli muuttuu.

Oliovirta (object flow) on suhde olion ja tilan välillä, joka kertoo, onko jokin tila on luonut olion, muuttanut oliota tai tuhonnut olion. Voidaan ajatella, että oliot ovat tilojen syötteitä tai tulosteita riippuen virran suunnasta.

Radan (swimlane) avulla voidaan osittaa toimintamallin kuvaamaa toimintaa vastuualueiden mukaan. Radoilla voidaan kertoa esimerkiksi, mitkä liiketoimintayksiköt ovat vastuussa tietyistä toiminnoista tai toiminnasta. Tilasta on vastuussa aina yksi rata, kun taas oliosta voi olla vastuussa yhdestä kahteen rataa. Ratoja ei aina mallinneta toimintakaavioissa.

Yllä esitellyistä käsitteistä ja niiden välisistä suhteista on muodostettu metamalli. Metamalli on esitetty kuviossa 5.



KUVIO 5. Toimintamallin metamalli.

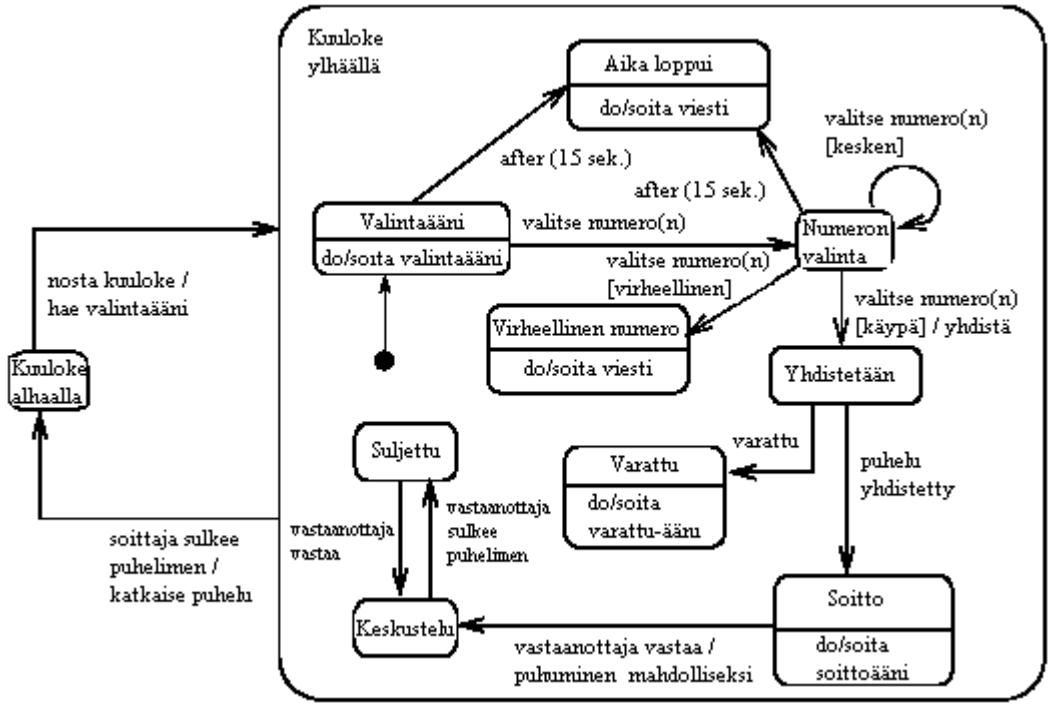
3.3 Tilamalli

Tässä kohdassa kuvataan UML:n tilamalli yleisellä tasolla esimerkkejä avuksi käyttäen. Yleiskuvauksen jälkeen kerrotaan toimintamallin keskeisimmät käsitteet tarkemmin ja esitetään toimintamallin metamalli.

3.3.1 Yleiskuvaus

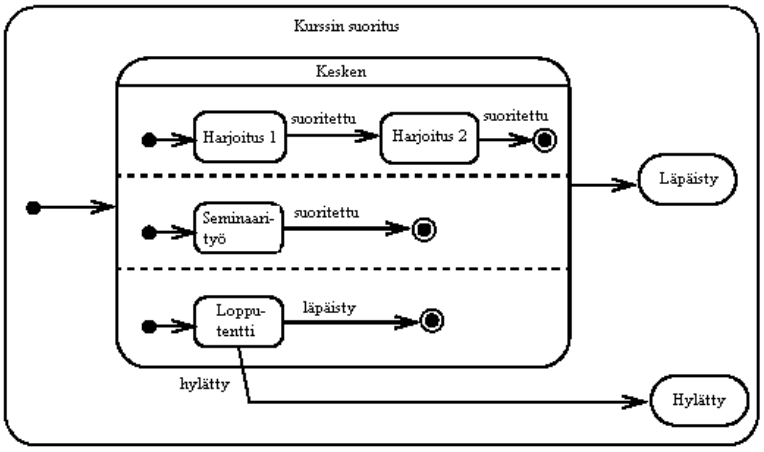
Tilamallia (statechart diagram) käytetään mallintamaan yksittäisen olion käyttäytymistä. Tilamalli kuvaa sarjan tiloja, joita olio käy läpi elinaikanaan reagoidessaan erilaisiin tapahtumiin. (Booch, Rumbaugh & Jacobson 1999, 287)

Kuviossa 6 on esimerkki tilamallin käytöstä. Kyseinen esimerkki kuvaa puhelinsoittoon liittyviä tiloja. Esimerkissä käytetty notaatio on melkein sama kuin toimintamallin yhteydessä. Erona ovat koostetilat ja siirtymiiin liittyvät tapahtumat ja toiminnot. Esimerkin kuvaama toiminta lähtee liikkeelle ’nosta kuuloke’ -tapahtumasta. Tämän jälkeen siirrytään ’Kuuloke ylhäällä’ -koostetilaan. Koostetilassa ensimmäinen tila on ’Valintaääni’, josta numeroja näppäiltäessä siirrytään ’Numeron valinta’ -tilaan tai tietyn ajan kuluttua ’Aika loppui’ -tilaan. Jos ’Numeron valinta’ –tilassa ollessa syötetty numero on oikein, siirrytään ’Yhdistetään’ –tilaan. Väärän numeron saatuessa siirrytään ’Virheellinen numero’-tilaan. ’Yhdistetään’ -tilasta on mahdollisuus siirtyä ’Varattu’ - tai ’Soitto’ -tilaan. Jos ’Soitto’ -tilassa ollessa tapahtuu ’vastaa puhelimeen’ –tapahtuma, siirrytään ’Keskustelu’ –tilaan. Vastaanottajan sulkiessa puhelimen siirrytään ’Suljettu’ –tilaan.



KUVIO 6. Puhelinsoitto tilakaaviona (OMG 2000b).

Kuviossa 7 on koostetila, joka kuvaa jonkin kurssin suorittamista. Koostetilassa on kolme alitilaa: ’Kesken’, ’Läpäisty’ ja ’Hylätty’. Näistä ensimmäisessä on kolme rinnakkaista tilasiirtymäketjua. Kun kaikissa näissä ketjuissa päästään lopetustilaan, siirrytään ’Läpäisty’ -tilaan. Jos lopputentissä epäonnistutaan, siirrytään ’Hylätty’ -tilaan.



KUVIO 7. Tilakaavio kurssin suorittamisesta (OMG 2000b).

3.3.2 Metamalli

Edellisen kohdan esimerkeissä esiintyneet käsitteet määritellään alla tarkemmin. Samalla luodaan pohjaa tilamallia kuvaavan metamallin ymmärtämiselle.

Tila (state) on hetki olion elinaikana, jolloin se tyydyttää jonkin ehdon, suorittaa jotain toimintaa tai odottaa jotain tapahtumaa (Booch, Rumbaugh & Jacobson 1999, 290). Tilamallin yhteydessä tilalla voi olla nimi, *aloitus*- (entry action) ja *lopetustoiminnot* (exit action) sekä siihen voi liittyä sisäisiä siirtymiä ja lykättyjä tapahtumia. Tila liittyy aina yhteen *olioon*, oliolla taas voi olla useampia tiloja. Tilamallissa on myös kaksi erikoistilaa: *aloitustila* (initial state) ja *lopetustila* (final state). Nimensä mukaisesti aloitustila on tila, josta tilamallin kuvaama toiminta alkaa. Lopetustila taas on tila, johon tilamallin kuvaama toiminta päättyy.

Yleisesti *tapahtuma* (event) on jokin ilmiö, jolla on tapahtuma-aika sekä -paikka. Tilamallin yhteydessä tapahtuma on jokin heräte, joka laukaisee siirtymän (Booch, Rumbaugh & Jacobson 1999, 290). Tapahtumille voidaan antaa nimi, parametrit ja tyyppi. Tapahtumia on neljänlaisia: kutsutapahtumassa (call event) kutsutaan operaatiota, muutostapahtumassa (change event) tietty ehto muuttuu todeksi, signaalitapahtumassa (signal event) otetaan vastaan signaali ja aikatapahtumassa (time event) kuvataan ajan kulumista tai absoluuttisen ajankohdan ohittamista (Rumbaugh, Jacobson & Booch 1999, 268). *Lykätyllä tapahtumalla* (deferred event) tarkoitetaan sellaista tapahtumaa, johon tietty tila ei voi suoraan reagoida tilasiirtymiensä kautta, mutta johon halutaan silti ottaa kantaa.

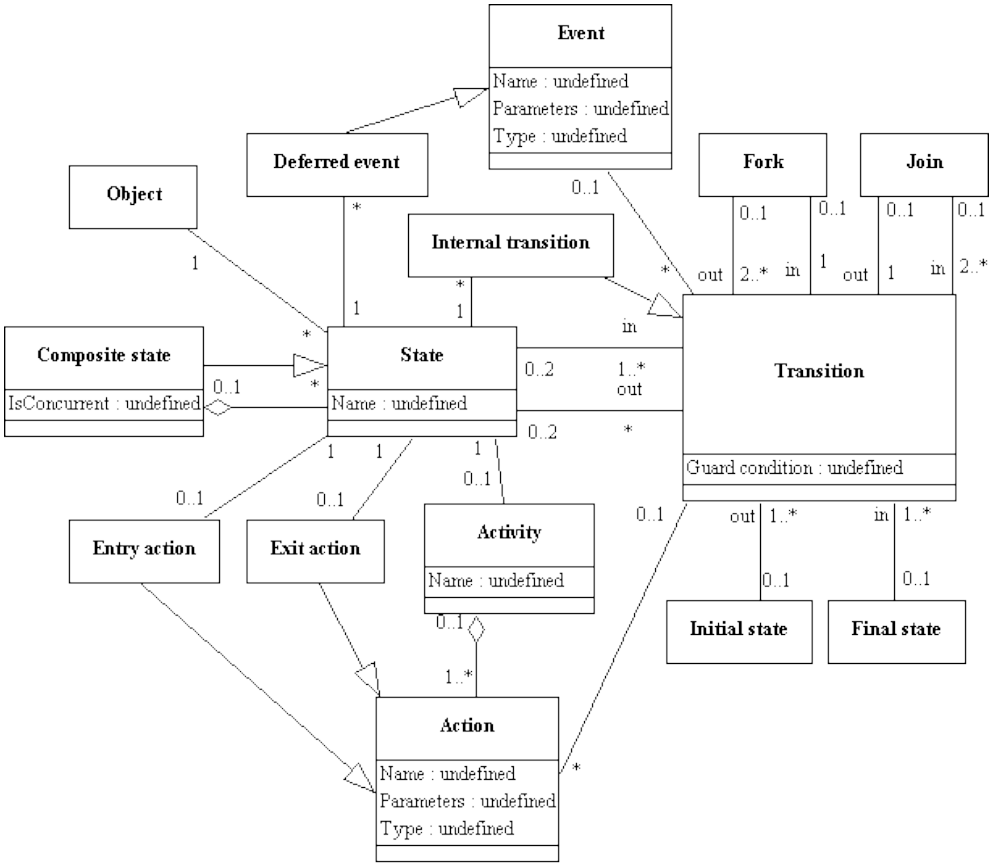
Siirtymä (transition) on suhde kahden tilan välillä, jossa ensimmäisestä tilasta siirrytään toiseen tietyn tapahtuman sattuessa ja tiettyjen ehtojen ollessa voimassa (Booch, Rumbaugh & Jacobson 1999, 290). Siirtymään liittyvää ehtoa kutsutaan siirtymäehdoksi (guard conditon). Sama tapahtuma voi liittyä useaan siirtymään. Tällöin siirtymien eroina ovat siirtymäehdot. Siirtymän aikana voidaan suorittaa toimintoja.

Sisäinen siirtymä (internal transition) on siirtymä, jossa siirtymän lähtö- ja kohdetilat ovat samoja. Sisäisen siirtymän yhteydessä ei suoriteta tilan aloitus- ja lopetustoimintoja.

Koostetila (composite state) on tila, joka koostuu alitiloista ja niiden välisistä siirtymistä. Koostetila voi kuvata peräkkäisiä tiloja tai useampaa rinnakkaisesti tapahtuvaa tilasiirtymäketjua. Koostetilalla on kaikki tavallisen tilan ominaisuudet eli nimi sekä aloitus- ja lopetustoiminnot. Tilasiirtymät voivat kohdistua suoraan koostetilan alitiloihin tai itse koostetilaan, jolloin tilasiirtymät alkavat koostetilan aloitustilasta. Koostetilasta poistuminen voi tapahtua suoraan jostain tilasta tai lopetustilaan siirtymisen jälkeen. Jos koostetila kuvaa rinnakkaisia tilasiirtymiä, voidaan siirtymä tällaiseen koostetilaan tehdä *jakajan* (fork) kautta ja vastaavasti poistuminen voidaan tehdä *liittäjän* (join) kautta. Jakajan ja liittäjän käyttö ei kuitenkaan ole pakollista.

Toiminto (action) on suoritettava lauseke, joka muodostaa laskennallisen prosessin abstraktion (Booch, Rumbaugh & Jacobson, 211). Toiminnolla on nimi ja parametreja. Toimintoja on useamman tyyppisiä: kutsutoiminto (call) on olion operaation kutsu, palautustoiminto (return) palauttaa arvon viestin kutsujalle, lähetystoiminto (send) lähettää signaalin jollekin oliolle, luontitoiminto (create) luo olioita ja tuhoamistoiminto (destroy) tuhoaa olioita (Booch, Rumbaugh & Jacobson, 211). Tilamallissa toiminnot voivat olla tilan aloitus- ja lopetustoimintoja tai osana tilan suorittamaan toimintaa. Myös siirtymiin voi liittyä toimintoja.

Tila suorittaa *toimintaa* (activity), kunnes jokin tapahtuma keskeyttää toiminnan tai toiminta päättyy itsestään. Tapahtuman yhteydessä laukaistaan tilasiirtymä, muuten jäädään odottamaan jotain tapahtumaa. Toiminta koostuu useammasta toiminnosta. Kuviossa 8 on yllä esitellyistä käsitteistä ja niiden välisistä suhteista koottu metamalli.



KUVIO 8. Tilamallin metamalli

3.4 Yhteistoimintamalli

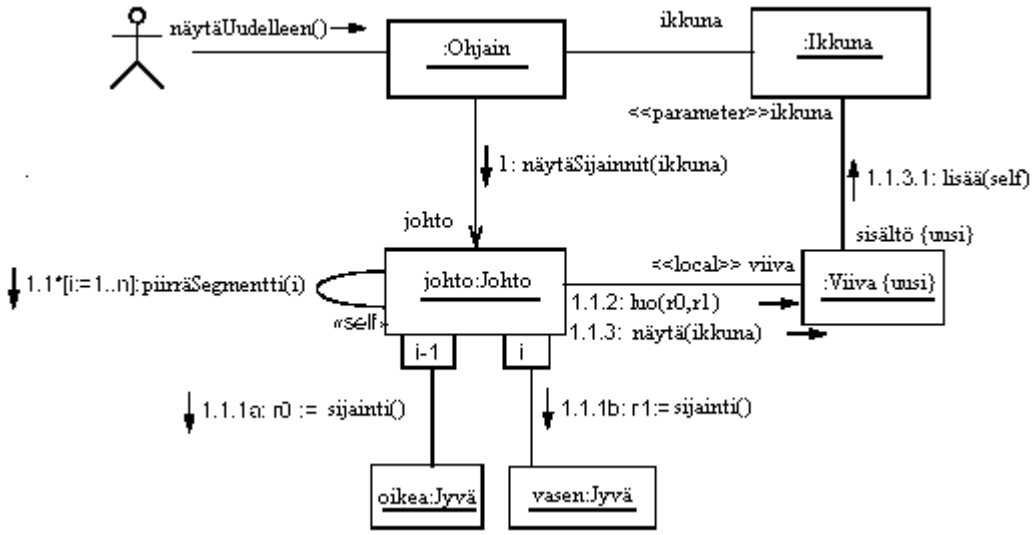
Tässä kohdassa kuvataan UML:n yhteistoimintamali yleisellä tasolla esimerkkejä avuksi käyttäen. Yleiskuvauksen lisäksi kuvataan yhteistoimintamallin keskeisimmät käsitteet tarkemmin ja esitetään yhteistoimintamallin metamalli.

3.4.1 Yleiskuvaus

Yhteistoimintamalli (collaboration diagram) ja sekvenssimalli ovat vuorovaikutusta kuvaavia malleja (interaction diagram). Vuorovaikutusta kuvaavat mallit kuvaavat olioita ja niiden välisiä suhteita sekä viesteistä joita oliot lähettävät toisilleen.

Yhteistoimintamalli korostaa viestejä lähettävien ja vastaanottavien olioiden välisiä rakenteellisia suhteita. (Booch, Rumbaugh & Jacobson, 243)

Kuviossa 9 on esimerkki yhteistoimintamallin käytöstä. Kyseinen esimerkki kuvaa ’johto’ –olion kuvan päivittämistä. Yhteistoiminta alkaa ’näytäUudelleen’ -viestin lähetyksellä. Viestin lähettäjänä on tikku-ukolla kuvattava aktori ja vastaanottajana suorakulmiolla kuvattava ’Ohjain’ –luokan olio. Aktorin ja olion yhdistävä viiva on linkki. ’Ohjain’ –luokan oliosta on edelleen linkit ’Ikkuna’ –luokan olioon ja ’Johto’ –luokan olioon. ’näytäUudellen’ –viestin jälkeen lähetetään ’näytäSijainnit’ –viesti. ’näytäSijainnit’ –viestin jälkeen vuorossa on n kertaa toistettava ’piirräSegmentti’ –viesti, jolla on ’sijainti’, ’luo’ ja ’näytä’ –aliviestit. ’piirräSegmentti’ –viestin parametrin arvoilla i-1 lähetään ’sijainti’ –viesti ’oikea’ –oliolle ja arvoilla i ’vasen’ –oliolle. ’sijainti’ –viestin paluuarvot tallentuvat muuttujiin r0 ja r1. Kun sijaintitiedot on saatu, luodaan uusi ’Viiva’ –luokan olio viestillä ’luo’. ’Viiva’ –luokan oliolle lähetetään ’näytä’ –viesti. ’näytä’ –viestin aliviesti ’lisää’ lähetetään ’Ikkuna’ –luokan oliolle. Viestin parametrina on viite ’Viiva’ –luokan olioon.



KUVIO 9. Esimerkki vuoravaikutuskaaviosta (OMG 2000b)

3.4.2 Metamalli

Alla kuvataan edellisen kohdan esimerkissä esiintyneet käsitteet tarkemmin. Samalla luodaan pohjaa yhteistoimintamallia kuvaavan metamallin ymmärtämiselle.

Olioita (object) on sekä passiivisia (passive) että aktiivisia (active). Passiivinen olio on olio, jolla ei ole omaa kontrollisäiettä (Rumbaugh, Jacobson & Booch 1999, 384). Aktiivinen olio (active object) on taas olio, joka omistaa prosessin tai säikeen, ja pystyy aloittamaan kontrollitoimintaa (Booch, Rumbaugh & Jacobson 1999, 311). Passiivisen olion toiminnan käynnistää siis jokin aktiivinen olio. Yhteistoimintamallissa olioille voidaan määrittää nimi sekä tyyppi. Tyypillä tarkoitetaan olion luokkaa.

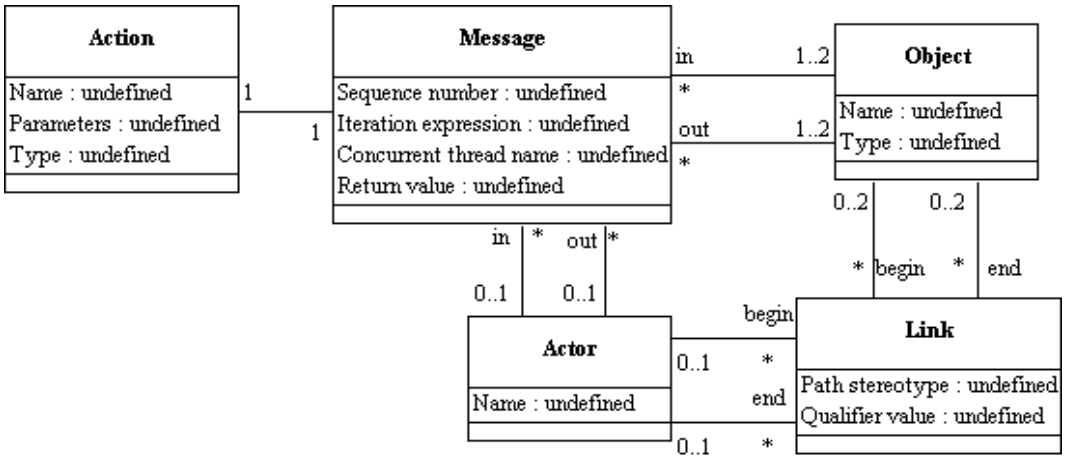
Aktori (actor) on abstraktio järjestelmän ulkopuolisista kokonaisuuksista, joiden kanssa järjestelmä on vuorovaikutuksessa (Rumbaugh, Jacobson & Booch 1999, 144). Aktoreita ovat esimerkiksi järjestelmää käyttävät ihmiset tai jokin toinen atk-järjestelmä, joka käyttää hyväkseen kuvatun järjestelmän toimintoja. Aktori aloittaa usein yhteistoimintamallin kuvaaman toiminnan.

Viestit (message) ovat olioiden välistä tiedonvälitystä, josta odotetaan seuraavan toimintaa (Booch, Rumbaugh, Jacobson, 210). Viestejä lähettävät toisilleen oliot ja aktorit. Viestien järjestysnumeroilla (sequence number) kerrotaan missä järjestyksessä oliot ja aktorit lähettävät viestejä toisilleen. Järjestysnumeroiden avulla viestejä voidaan järjestää myös hierarkiaan.Viestejä voidaan toistaa toistolauseen (iteration expression) ilmoittama määrä. Jos useampia viestejä halutaan lähettää samanaikaisesti, lisätään viestiin rinnakkaisten säikeiden nimet (concurrent thread name). Palautusarvot (return value) ovat arvoja, joita viesteihin liittyvät toiminnot palauttavat.

Toiminto (action) on toimintaa, joka tapahtuu viestin vastaanottavassa oliossa. Toiminnolla on nimi sekä parametrit. Erilaisia toimintoja ovat mm. kutsutoiminto (call), palautustoiminto (return), lähetystoiminto (send) ja luontitoiminto (create) (Booch, Rumbaugh & Jacobson, 211).

Yhteistoimintamallissa oliot ja aktorit viittaavat toisiinsa *linkkien* (link) avulla. Kun oliosta tai aktorista on linkki toiseen olioon tai aktoriin, on viestien lähetys mahdollista. Linkkejä on viittä eri tyyppiä: ’association’ –linkki määrittää, että tietystä oliosta on viite toiseen olioon, ’self’ –linkki määrittää, että olio lähettää viestejä itselleen, ’global’ –linkki määrittää, että viitattava olio on näkyvyydeltään globaali, ’local’ –linkki määrittää, että viitattava olio on näkyvyydeltään paikallinen ja ’parameter’ –linkki määrittää, että viitattava olio välittyy parametrina (Booch, Rumbaugh & Jacobson 1999, 210). Linkin tyyppi kerrotaan ’path stereotype’ –määreessä. Jos tietyn viestin vastaanottaja voi vaihdella, ilmaistaan tämä vaihtoehtoisuus linkkiin liittyvän ’qualifier value’ –määreen avulla.

Yllä esitellyistä käsitteistä ja niiden välisistä suhteista voidaan esittää metamalli. Metamalli on esitetty kuviossa 10.



KUVIO 10. Yhteistoimintamallin metamalli.

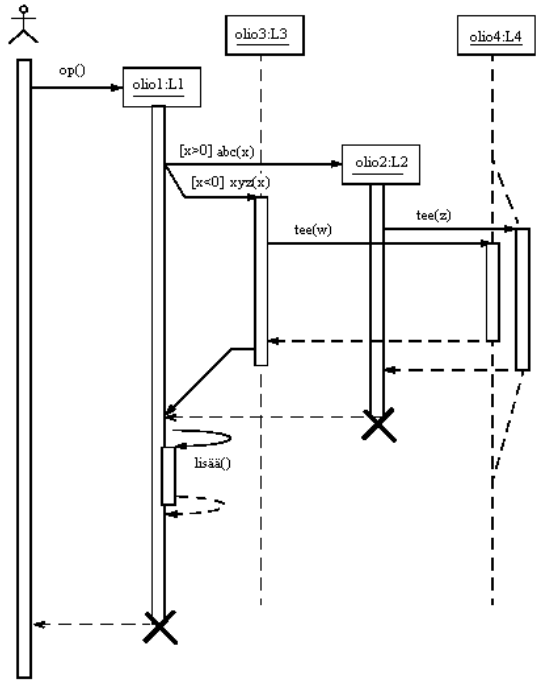
3.5 Sekvenssimalli

Tässä kohdassa kuvataan UML:n sekvenssimalli yleisellä tasolla esimerkkejä avuksi käyttäen. Yleiskuvauksen jälkeen kerrotaan sekvenssimallin keskeisimmät käsitteet tarkemmin ja esitetään sekvenssimallin metamalli.

3.5.1 Yleiskuvaus

Sekvenssimalli (sequence diagram) on yhteistoimintamallin tapaan vuorovaikutusta kuvaava malli. Erona yhteistoimintamalliin on se, että sekvenssimalli korostaa olioiden välisten viestien ajallista järjestystä (Booch, Rumbaugh & Jacobson, 243).

Kuviossa 11 on esimerkki sekvenssimallin käytöstä. Kyseinen esimerkki kuvaa abstraktia toimintaa. Esimerkin notaatio on osittain sama kuin yhteistoimintamallin yhteydessä. Eroina ovat olioiden välisten linkkien puuttuminen, olioihin liittyvä elinkaarta kuvaava katkoviiva, sekä elinkaareen liittyvä kontrollikeskittymää kuvaava pystysuora palkki. Lisäksi viesteihin ei liitetä järjestysnumeroita. Sekvenssimallissa viestien lähetysjärjestys etenee ylhäältä alas. Esimerkin toiminta lähtee liikkeelle aktorin lähettämästä ’op’ –viestistä, joka luo ’olio1’ –olion. ’olio1’ lähettää joko olion ’olio2’ luovan ’abc’ –viestin tai oliolle ’olio3’ ’xyz’-viestin riippuen muuttujan x arvosta. Vastaavasti lähetetään joko ’tee(z)’ tai ’tee(w)’ –viestit ’olio4’ –oliolle. Molempien vaihtoehtojen jälkeen kontrolli palautuu ’olio1’ –oliolle, joka lähettää ’lisää’ –viestin itselleen. Lopuksi kontrolli palaa aktorille ’olio1’ –olion tuhoutuessa.



KUVIO 11. Esimerkki sekvenssimallista (OMG 2000b).

3.5.2 Metamalli

Alla kuvataan edellisessä kohdassa esiintyneet käsitteet tarkemmin. Samalla luodaan pohjaa sekvenssimallia kuvaavan metamallin ymmärtämiseen.

Aktorit (actor) ja *oliot* (object) ovat sekvenssimallin yhteydessä melkein samoja käsitteitä kuin yhteistoimintamallin yhteydessä. Eroina ovat olioihin liittyvät elinkaaret ja olioihin sekä aktoreihin liittyvät kontrollikeskittymät.

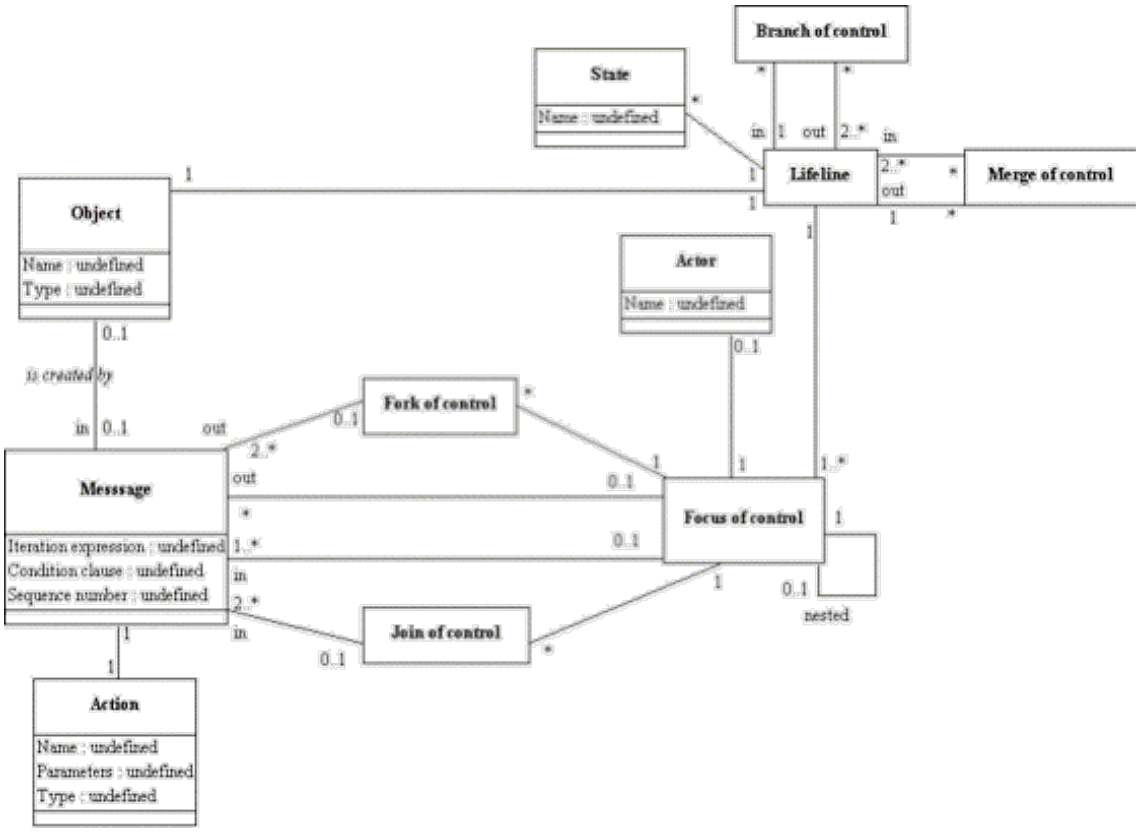
Olioon liittyvä *elinkaari* (lifeline) kuvaa aikaa, jolloin olio on olemassa. Elinkaaren päässä oleva rasti kertoo olion tuhoutumisesta. Elinkaareen voi liittyä yksi tai useampi kontrollikeskittymiä. Elinkaari voidaan jakaa vaihtoehtoisiin osiin *haarauman* (branch of control) avulla. Elinkaaren jakaminen on tarpeellista, koska olio voi ottaa vastaan vaihtoehtoisia viestejä. Kohtaa elinkaarella, jossa vaihtoehtoiset viestit yhdistyvät, kutsutaan *yhdistimeksi* (merge of control). Elinkaaren yhteydessä voidaan myös kertoa miten olion *tila* (state) muuttuu, kun olio vastaanottaa viestejä.

Kontrollikeskittymä (focus of control) kuvaa aikaa, jolloin olio tai aktori suorittaa toimintaa. Olioihin voi liittyä useampia kontrollikeskittymiä. Kontrollikeskittymä alkaa, kun olio vastaanottaa viestin, ja loppuu, kun viestin lähettäjälle lähetetään palautusviesti. Kontrollikeskittymään voi liittyä myös sisäkkäisiä (nested) kontrollikeskittymiä, joita käytetään kun olio lähettää viestin itselleen.

Viestit (message) ovat olioiden välistä tiedonvälitystä, josta odotetaan seuraavan toimintaa (Booch, Rumbaugh, Jacobson, 210). Sekvenssimallissa viestejä lähettävät toisilleen oliot ja aktorit. Suoraan olioihin kohdistuvat viestit ovat oliota luovia viestejä, muut viestit näkyvät kontrollikeskittymien välisinä viesteinä. Viestejä voidaan toistaa toistolauseen (iteration expression) ilmoittama määrä, ja niihin voidaan liittää ehtolause (condition clause). Ehtolauseen totuusarvon perusteella valitaan jokin vaihtoehtoisesti lähetettävistä viesteistä. Pistettä, josta vaihtoehtoiset viestit lähtevät, kutsutaan *jakajaksi*

(fork of control). Vaihtoehtoisten viestien aloittamat vaihtoehtoiset kontrollivirrat yhdistyvät *liittäjä* (join of control) –nimisessä pisteessä. Viesteihin liittyy aina *toiminto* (action). Toiminto on sekvenssimallin yhteydessä täsmälleen sama käsite kuin yhteistoimintamallin yhteyssä.

Yllä kuvattuja käsitteitä ja niiden välisiä suhteita kuvaava metamalli on esitetty kuviossa 12.



KUVIO 12. Sekvenssimallin metamalli.

3.6 Yhteenveto

Tässä luvussa on kuvattu metamallintamisen tavoitteita ja toteutukseen liittyviä valintoja. Tavoitteita kuvaamalla pyrittiin selkeyttämään metamallien osuutta tutkimuksessa. Toteutukseen liittyviä valintoja kuvaamalla on pohjustettu metamallien relevanttiutta tutkimuksen osana.

Lisäksi tässä luvussa on kuvattu yleisesti ja yksityiskohtaisesti kaikki UML:n dynaamiset mallit. Yksityiskohtaiset käsitteiden kuvaukset on tiivistetty metamallien avulla. Metamallit ovat toimineet syötteenä luvussa 4 kuvatulle metamallien integroinnille.

4 DYNAAMISTEN MALLIEN INTEGROINTI

Tässä luvussa kuvataan UML:n dynaamisten mallien metamallien integrointia. Ensiksi kuvataan integrointitekniikka ja –periaatteet. Sen jälkeen kuvataan varsinainen integroinnin kulku kahdessa vaiheessa. Lopuksi esitetään ydinmalli UML:n dynaamisten mallien käsitteistä ja esitellään integraation avulla saatuja tuloksia.

4.1 Integrointitekniikka ja -periaatteet

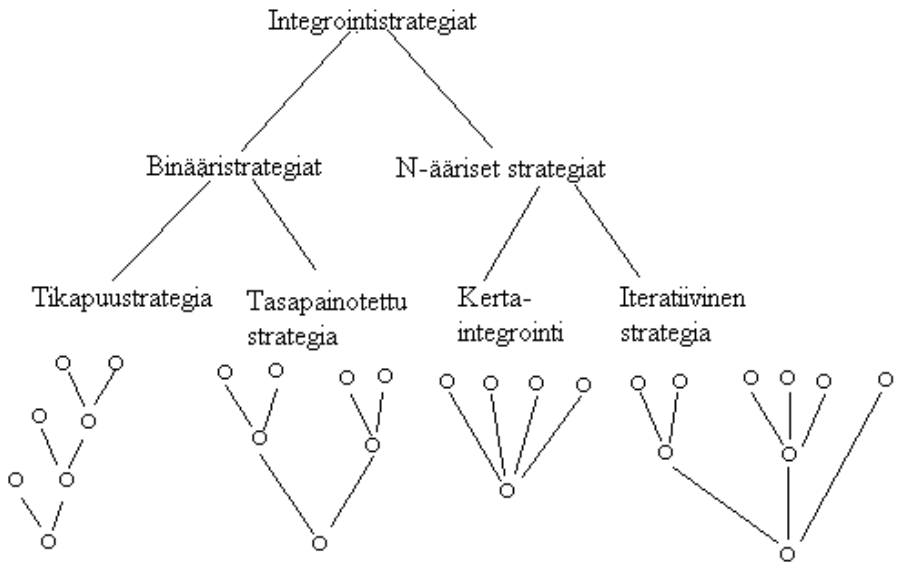
Batinin ym. (1992) mukaan näkymien integrointi on prosessi, jossa yhdistetään käsitteellisiä malleja globaaliksi malliksi siten, että kaikki tietyn sovelluksen vaatimukset täyttyvät. Käsitteellisiä malleja on useita, koska monimutkaista sovellusta on tarkoituksenmukaista suunnitella pienissä osissa ja monen suunnittelijan toimesta. Näkymien integroinnin tavoitteena on löytää kaikki ne mallien osat, jotka viittaavat samoihin todellisuuden osiin, ja yhtenäistää niiden esitykset. Näkymien integrointi on tarkoitettu tietokantojen suunnitteluun, mutta Leppäsen (2000) mukaan sitä voidaan käyttää myös mallien integrointiin, koska malli voidaan ymmärtää näkymänä. Mallit ovat tässä työssä edellä kuvattuja UML:n dynaamisten mallien metamalleja.

Suurin ongelma näkymien integroinnissa on Batinin ym. (1992) mukaan yhdistettävien näkymien erojen löytäminen. Erot näkymissä johtuvat suunnittelijoiden erilaisista näkökulmista, mallien rakenteiden vastaavuuksista ja epäyhteensopivista määrityksistä. Suunnittelijoiden näkökulmien erilaisuudet ovat esimerkiksi sitä, että käsitteet nähdään eri abstraktiotasoilla tai sitä että käsitteisiin liitetään erilaisia attribuutteja. Rakenteiden vastaavuuksilla tarkoitetaan esimerkiksi sitä, että tietty asia voidaan ilmaista yhdessä mallissa attribuuttina ja toisessa mallissa käsitteenä. Epäyhteensopivat määritykset taas ovat suunnittelun yhteydessä tehtyjä virheitä. Edellä mainitut syyt mallien eroihin johtavat ristiriitoihin eli saman käsitteen erilaisiin esityksiin.

Näkymien integrointiin liittyvät tehtävät suoritetaan isossa (integration in the large) ja pienessä mittakaavassa (integration in the small). Ison mittakaavan integrointitehtävissä valitaan mitkä näkymät integroidaan keskenään ja missä järjestyksessä. Pienen mittakaavan integroinnissa taas on kyse kahden tai useamman näkymän välillä olevien ristiriitojen analysoinnista ja ratkaisusta.

Ison mittakaavan integrointiin Batini ym. (1992) esittelevät kaksi lähestymistapaa. Ensimmäinen lähestymistapa on valita useita näkymiä integroitavaksi kerrallaan, jolloin syntyy useita osittain integroituja näkymiä. Tämä lähestymistapa ei kuitenkaan ole Batinin ym. mukaan käytännöllinen, koska ristiriitojen havaitseminen on hankalaa. Toisena vaihtoehtona Batini ym. ehdottavat näkymien integrointia pareittain siten, että integroinnin tuloksena olisi aina yksi näkymä, joka kasvaisi asteittain globaaliksi malliksi.

Myös Kaivola (1991) kuvaa näkymien valintaan liittyviä tehtäviä. Tässä yhteydessä näkymien valintaa kutsutaan integrointistrategiaksi. Kaivola on jakanut integrointistrategiat binäärisiin ja n-äärisiin strategioihin. Binäärisissä strategioissa integroidaan aina kaksi näkemystä kerrallaan. Joko kaksi paria kerrallan tai niin että uusi malli integroidaan edellisen integroinnin välituloksen kanssa. Edellisessä tapauksessa on kyseessä tasapainotettu strategia ja jälkimmäisessä tikapuustrategia. N-äärisissä strategioissa taas integroidaan kerrallaan enemmän kuin kaksi näkymää. Vaihtoehtoisia n-äärisiä strategioita ovat kertastrategia ja iteratiivinen strategia. Kertastrategiassa integroidaan kaikki näkymät kerrallaan. Iteratiivisessa strategiassa taas samanarvoiset näkymät ryhmitellään ja integroidaan keskenään, ja näin saadut väliaikaiset mallit integoidaan keskenään. Integrointistrategiat on kuvattu kuviossa 13.



KUVIO 13. Integrointistrategiat (Kaivola 1991, 30).

Tässä työssä integrointistrategiaksi valitaan tasapainotettu binääristrategia. Parit muodostetaan toisaan mahdollisimman lähellä olevista metamalleista. Lähtöpareiksi muodostuvat tällöin tila- ja toimintamallien sekä sekvenssi- ja yhteistoimintamallien metamallit. Parit integroidaan keskenään ja näin muodostuneet väliaikaiset mallit integroidaan edelleen toisiinsa. Näin saadaan lopulta muodostettua globaali metamalli.

Pienen mittakaavan integrointitehtäviä ovat ristiriitojen analysointi ja ratkaisu. Batinin ym. (1992) mukaan ristiriitojen analysoinnin tavoitteena on etsiä erilaiset tavat esittää samaa todellisuuden osaa kahdessa eri näkymässä. Ristiriitojen analyysissä voidaan erottaa kaksi osatehtävää: nimeämis- ja rakenneristiriitojen analysoinnit. Nimeämisristiriitojen syynä ovat synonyymit ja homonyymit. Synonyymit tarkoittavat sitä, että samaan asiaan viitataan usealla nimellä, ja homonyymeilla sitä että eri asioihin

viitataan samoilla nimillä. Nimeämiskonfliktit ratkaistaan nimeämällä uudelleen ristiriidassa olevat käsitteet.

Kun nimeämisristiriidat on ratkaistu, siirrytään rakenneristiriitojen analyysiin. Rakenneristiriitojen analyysissa on tarkoitus selvitttää voidaanko samannimiset käsitteet yhdistää keskenään. Käsitteiden yhdistämisessä käytetään apuna seuraavanlaista luokittelua: identtiset käsitteet, yhteensopivat käsitteet ja yhteensopimattomat käsitteet. Identtisillä käsitteillä tarkoitetaan käsitteitä, joilla on täsmälleen sama esitysrakenne, ominaisuudet ja suhteet. Yhteensopivilla käsitteillä on erilaiset esitysrakenteet tai ominaisuudet ja suhteet, mutta ne eivät ole ristiriidassa keskenään. Yhteensopimattomilla käsitteillä on ominaisuuksia ja suhteita, jotka ovat ristiriidassa keskenään. Ristiriidat yhteensopimattomien käsitteiden välillä voivat johtua esimerkiksi erilaisista kardinaalisuuksista tai perintäsuhteista käsitteiden välillä. Ratkaisuna ristiriitoihin on jomman kumman näkymän rakenteen valinta, tai sellaisen uuden esityksen muodostaminen, joka tukee molempien käsitteiden kaikkia rajoitteita.

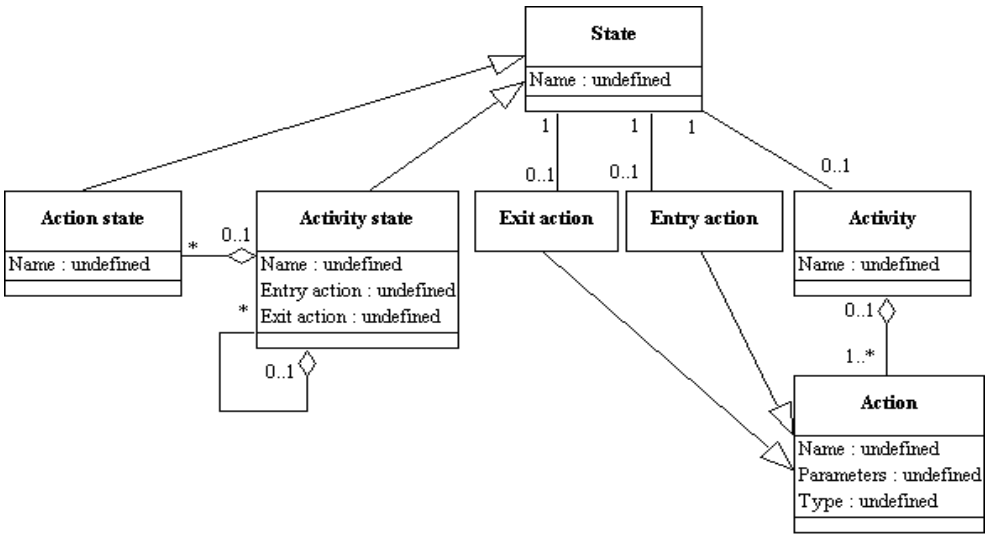
Kun nimeämis- ja rakenneristiriidat on saatu ratkaistua, suoritetaan näkymien yhdentäminen. Batinin ym. mukaan näkymien yhdentämisessä on syötteenä kaksi näkymää ja tulosteena on näkymä, jossa on kaikki mahdolliset kahden syötenäkymän käsitteet ja suhteet. Täysin toisiaan vastaavat käsitteet voidaan yhdistää suoraan, kun taas käsitteet joissa on erilaiset attribuutit, yhdistetään ottamalla molempien käsitteiden attribuuteista yhdiste. Yhdentämisen jälkeen syntynyttä integroitua mallia voidaan muokata uudelleen sen laadun parantamiseksi. Kaivolan (1991) mukaan Batini, Lenzerin ja Navathe (1986) ovat esittäneet laatukriteereitä, joilla integroitua näkymää voidaan testata. Näitä kriteerejä ovat täydellisyys ja oikeellisuus, minimaalisuus ja ymmärrettävyys.

Seuraavissa kohdissa kuvataan miten UML:n dynaamisten mallien metamallit on integroitu. Tasapainotetun binääristrategian mukaan on edetty tila- ja toimintamallien ja sekvenssi- ja yhteistoimintamallien metamallien integroinnin kautta globaalin metamallin tuottamiseen.

4.2 Tila- ja toimintamallien metamallien integrointi

Tilamallin ja toimintamallin käsitteiden välillä oli kaksi nimeämisristiriitaa. Aloitus- ja lopetustilan käsitteille oli malleissa annettu eri nimet. Toimintamallissa ne olivat nimeltään start state ja stop state, kun taas tilamallissa ne olivat nimeltään initial state ja final state. Nämä ristiriidat ratkaistiin käyttämällä tilamallin nimeämiskäytäntöjä.

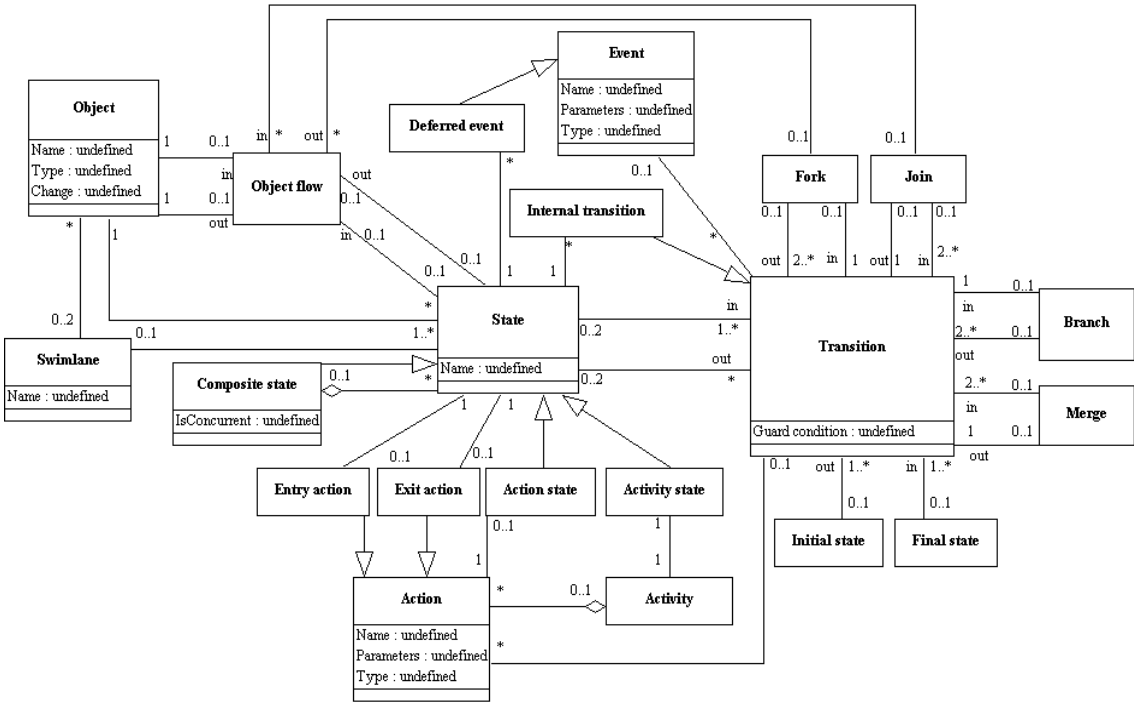
Tilakäsitteen kohdalla huomattiin rakenteellinen ristiriita, joka johtui yhteensopimattomista käsitteistä. Ongelmana oli se, että toimintamallin toiminto- ja toimintakäsitteet periytyivät tilakäsitteesta, kun taas tilamallin kohdalla toimintakäsite liittyi tilakäsitteeseen assosiaation kautta. Jos molempien mallien tilakäsitteeseen liittyvät käsitteet integroitaisiin suoraan, saataisiin kuviossa 14 kuvattu metamallin osa.



KUVIO 14. Yhdistetyn tilakäsitteen rakenne.

Kuviossa 14 esitettyä mallia muokattiin uudelleen seuraavalla tavalla. Toiminto- ja toimintakäsitteet liitettiin assosiaatioilla toiminto- ja toimintatilakäsitteisiin. Toiminto- ja toimintatiloista siirrettiin nimiattribuutti tilakäsitteeseen ja toimintatilan yhteydestä

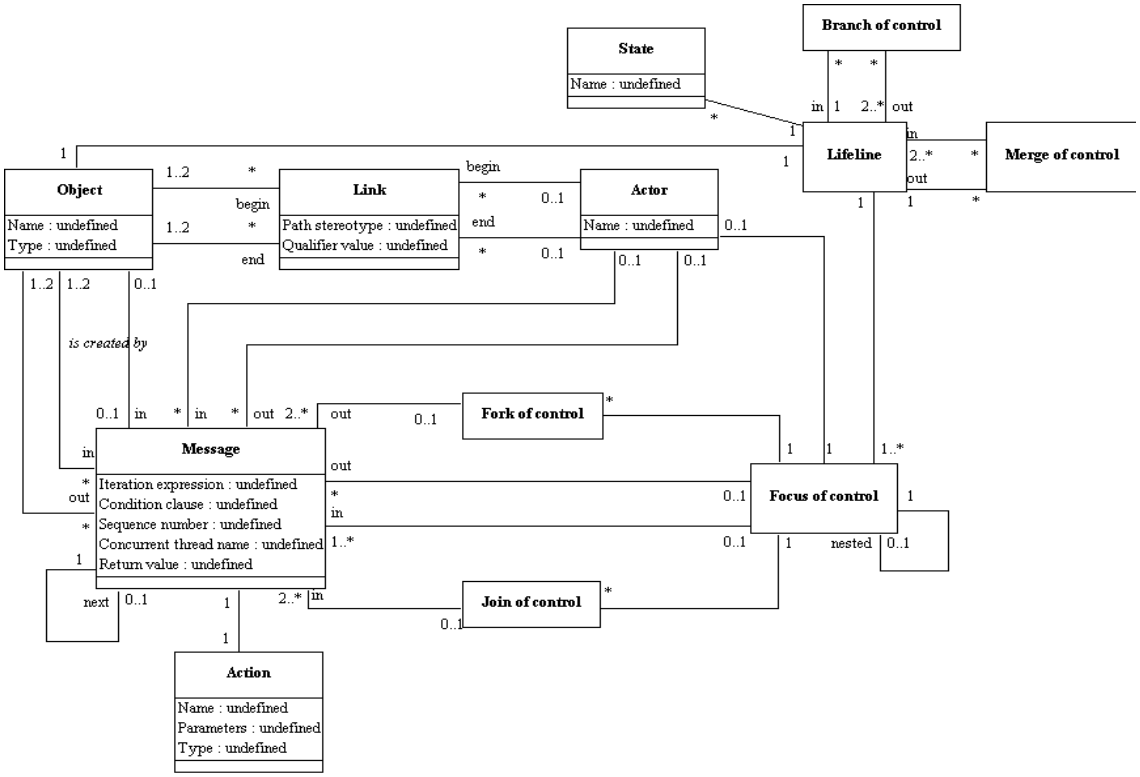
poistettiin aloitus- ja lopetustoimintoattribuutit. Lopullinen integroitu metamalli (Kuvio 15) saatiin aikaiseksi tilakäsitteiden uudelleenorganisoinnin jälkeen, yhdistämällä toiminta- ja tilamallien yhteiset käsitteet.



KUVIO 15. Toiminta- ja tilamallien metamallien integroitu metamalli.

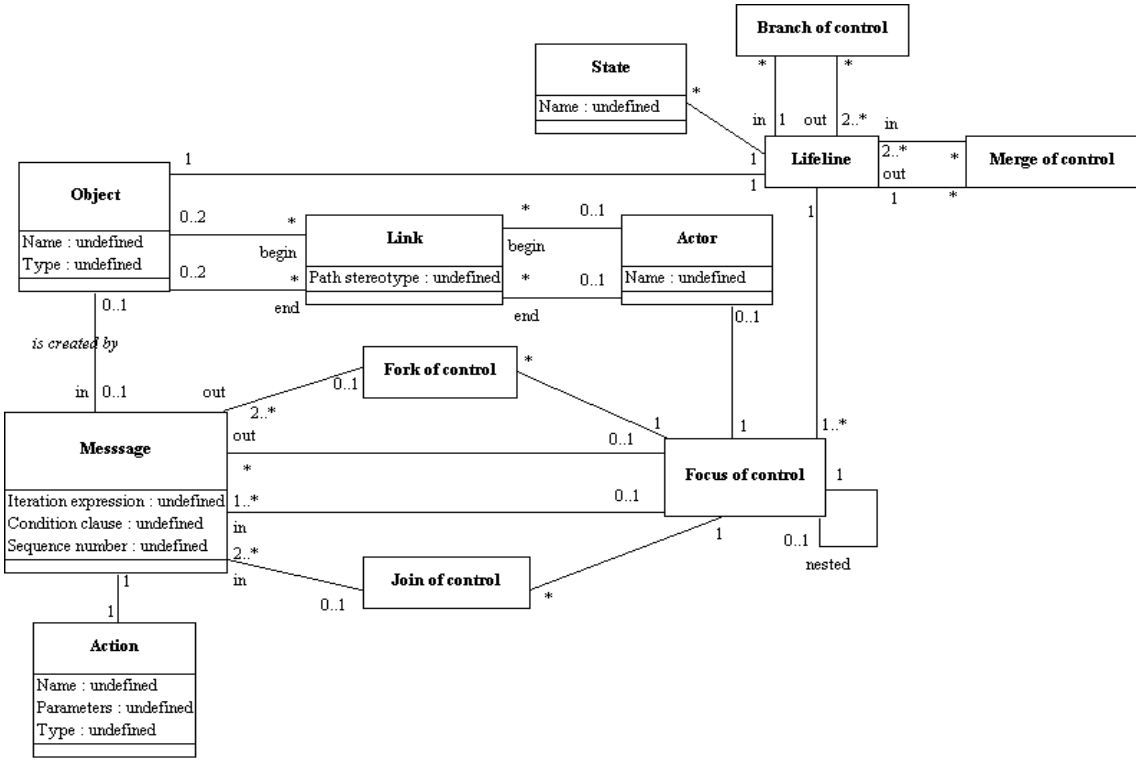
4.3 Sekvenssi- ja yhteistoimintamallien metamallien integrointi

Sekvenssi- ja yhteistoimintamallien metamallien välillä ei ollut nimeämis- tai rakenneristiriitoja, joten metamallit voitiin integroida suoraan toisiinsa. Integrointi tapahtui toiminto-, viesti-, olio- ja aktorikäsitteiden kautta. Kyseisiin käsitteisiin liitettiin molemmista metamalleista niihin liittyvät suhteet ja käsitteet. Viestikäsitteeseen yhdistettiin molempien metamallien viestikäsitteiden attribuutit. Integroinnin tulos on kuviossa 16.



KUVIO 16. Sekvenssi- ja yhteistoimintamallien metamallien integroitu metamalli ennen uudelleenjärjestelyä.

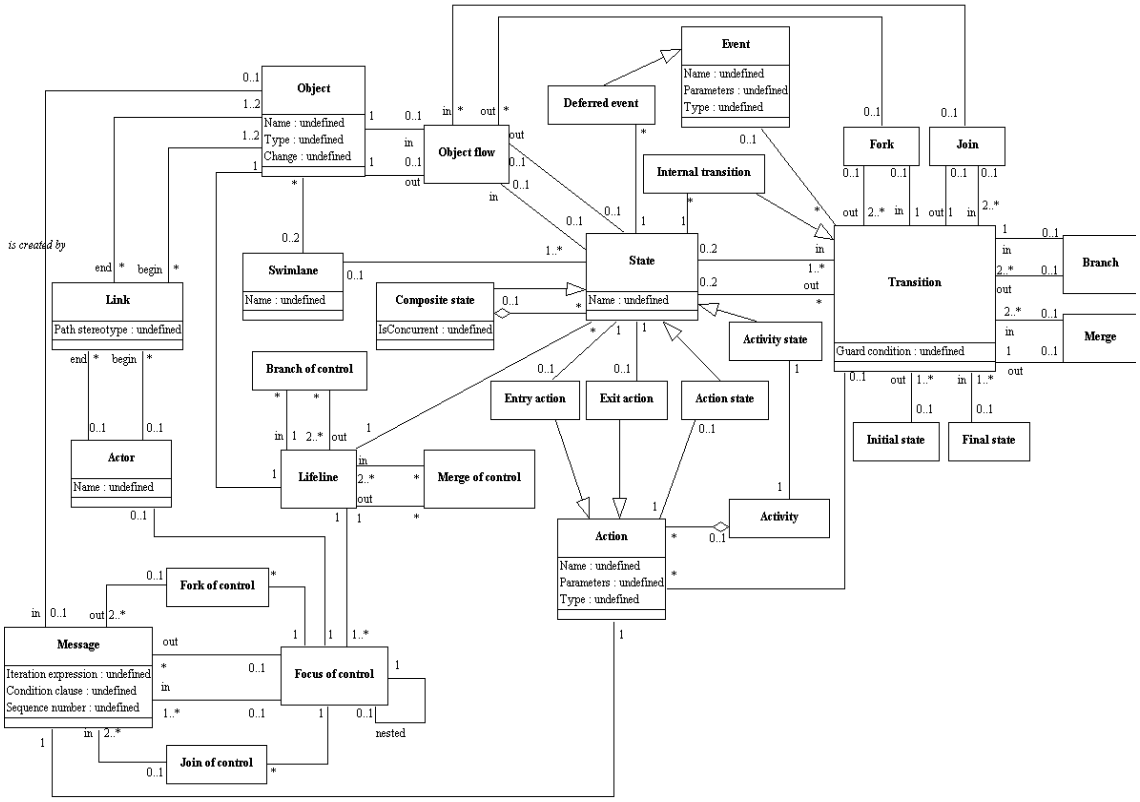
Integroitu metamalli kaipasi kuitenkin vielä jonkin verran uudelleenjärjestelyä. Ensinnäkin olion ja aktorin viestiin liittyvät assosiaatiot eivät olleet tarpeellisia, sillä sama asia oli ilmaistu viestin ja kontrollikeskittymän välisillä assosiaatioilla. Toiseksi viestikäsitteestä voitiin poistaa assosiaatio itseensä, koska saman asian ajoi järjestysnumero (Sequence number) –attribuutti. Kolmanneksi myös viestin rinnakkainen säie (Concurrent thread) –attribuutti oli ylimääräinen. Sama asia voitiin ilmaista sillä, että lähetetään yhtäaikaa viestejä useammalle kontrollikeskittymälle. Neljänneksi linkkikäsitteen tarkennearvo (Qualifier value) –attribuuttia ei myöskään tarvittu sillä, viestien vaihtoehtoisuutta voitiin ilmaista jakajan ja vaihtoehtoisten viestien avulla. Lopullinen integroitu malli on esitetty kuviossa 17.



KUVIO 17. Sekvenssi- ja yhteistoimintamallien integroitu metamalli.

4.4 Globaali metamalli ja integroinnin tulokset

Edellä tuotettujen integroitujen metamallien välillä ei ollut nimeämis- eikä rakenneristiriitoja. Metamallit voitiin integroida toisiinsa suoraviivaisesti käyttäen lähtökohtina olio-, tila- ja toimintokäsitteitä. Lopullinen tulos on esitetty kuviossa 18.



KUVIO 18. Globaali metamalli.

Integraatioprosessin aikana saatiin selville muutamia mallien välisiä ristiriitoja. Toiminta- ja tilamallien välillä löytyi kaksi nimeämisristiriitaa. Nämä koskivat aloitus- ja lopetustilojen nimeämisiä. Toimintamallissa aloitustilaa kutsuttiin start stateksi ja tilamallissa initial stateksi. Lopetustilaa kutsuttiin toimintamallissa stop stateksi ja tilamallissa final stateksi. Nimeämisristiriita ratkaistiin käyttämällä tilamallin nimeämiskäytäntöjä.

Integroinnin aikana löydettiin myös yksi rakenneristiriita. Toimintakäsitteiden suhdetyyppi tilakäsitteeseen oli erilainen toiminta- ja tilamallien välillä. Toimintamallissa tämä suhde oli perintä ja tilamallin kohdalla assosiaatio. Ristiriita ratkaistiin integraatiomallin uudelleen muokkauksella.

Taulukossa 3 kuvataan mitä käsitteitä UML:n dynaamiset mallit sisältävät. Taulukon avulla voidaan nähdä, että dynaamisissa malleissa erottuu kaksi ryhmää, toiminta- ja

tilamallit sekä sekvenssi- ja yhteistoimintamallit. Toiminta- ja tilamallien välillä on seitsemän yhteistä käsitettä ja sekvenssi- ja yhteistoimintamallien välillä on neljä yhteistä käsitettä. Kaikkien tai melkein kaikkien mallien yhteisiksi käsitteiksi havaittiin olio-, tila-, ja toimintokäsitteet. Sekvenssi- ja yhteistoimintamallit voidaan tulkita toistensa vaihtoehdoiksi. Eroina on kuitenkin se, että sekvenssimalli painottaa viestien ajallista järjestystä ja vaihtoehtoisuutta, kun taas yhteistoimintamalli painottaa viestejä lähettävien olioiden välisiä suhteita. Vaikka toiminta- ja tilamalleillakin on paljon yhteisiä käsitteitä, ei niitä kuitenkaan voida käyttää toistensa vaihtoehtoina. Tähän perusteena on, että toimintamallilla tarkastellaan toimintaa yleensä, ja tilamallilla tarkastellaan yhden olion tilan muuttumista.

TAULUKKO 3. Dynaamisten mallien käsitteet.

	Toimintamalli	Tilamalli	Yhteistoimintamalli	Sekvenssimalli
Tila	X	X		X
Toimintotila	X			
Toimintatila	X			
Siirtymä	X	X		
Haarauma (Branch)	X			
Yhdistin (Merge)	X			
Jakaja (Fork)	X	X		
Liittäjä (Join)	X	X		
Olio	X	X	X	X
Aloitustoiminto		X		
Lopetustoiminto		X		
Aloitustila (Initial/Start state)	X	X		
Lopetustila (Final/stop state)	X	X		
Oliovirta	X			
Rata	X			
Tapahtuma		X		
Koostetila		X		
Lykätty tapahtuma		X		
Sisäinen siirtymä		X		
Toiminto		X	X	X
Toiminta		X		
Linkki			X	
Aktori			X	X

(jatkuu)

TAULUKKO 3. (jatkuu)

Elinkaari				X
Viesti			X	X
Kontrollikeskittymä				X
Haarauma (Branch of control)				X
Yhdistin (Merge of control)				X
Jakaja (Fork of control)				X
Liittäjä (Join of control)				X

4.5 Yhteenveto

Tässä luvussa kuvattiin tutkimuksessa käytetty integrointitekniikka, integroinnin eri vaiheet sekä esitettiin globaali metamalli UML:n dynaamisten mallien käsitteistä. Lisäksi esitettiin integroinnin ja globaalin metamallin perusteella tehtyjä havaintoja.

Integrointitekniikasta kerrottiin sen taustasta, integroinnin ison ja pienen mittakaavan tehtävistä sekä ristiriitojen ratkaisumenetelmistä. Taustasta mainittiin, että Batin integrointitekniikkaa on alunperin käytetty tietokantanäkymien integrointiin. Ison mittakaavan integroinnista kuvattiin erilaiset integrointistrategiat ja pienin mittakaavan integroinnista kerrottiin erilaiset ristiriitatyypit ja niiden ratkaisut.

Varsinainen integrointi kuvattiin kahdessa vaiheessa. Ensiksi kuvattiin tila- ja toimintamallien integrointi, sitten sekvenssi – ja yhteistoimintamallin integrointi. Tila- ja toimintomallien integroinnin yhteydessä kuvattiin kahden nimeämisristiriidan ratkaisu sekä yhden rakenteellisen ristiriidan ratkaisu. Sekvenssi- ja yhteistoimintamallien integroinnissa ei havaittu ristiriitoja.

Globaali metamalli ja muut integroinnin tulokset esiteltiin tämän luvun viimeisessä kohdassa. Integroinnin tuloksia olivat mallien väliset ristiriidat sekä taulukko eri mallien sisältämistä käsitteistä.

Askel kohti globaalin metamallin laajentamista tehdään luvussa viisi, jossa tutkitaan UML 2.0:n uusia dynaamisen mallintamisen ominaisuuksia. Globaalia metamallia

voitaisiin laajentaa lisäämällä siihen UML 2.0:n uudet ominaisuudet metamallintamisen ja integroinnin avulla.

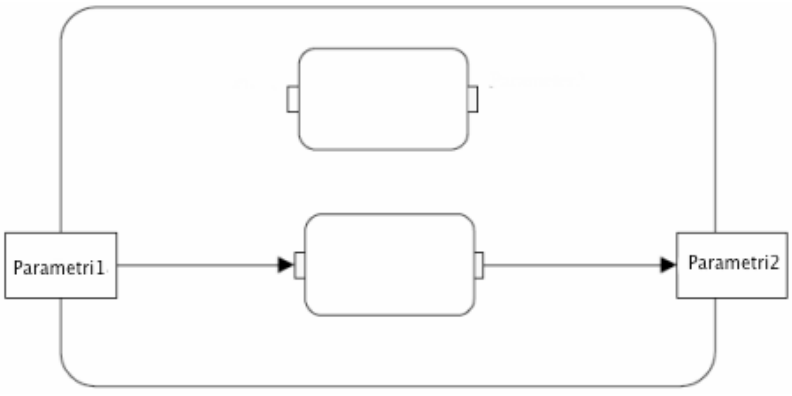
5 DYNAAMISTEN MALLIEN MUUTOKSET UML 2.0:SSA

Tässä luvussa esitellään dynaamisten mallien muutoksia UML 2.0:ssa verrattuna UML 1.3:een. Eroja on etsitty pääasiassa tutkimalla The Unified Modeling Language Reference Manual teoksen ensimmäistä (Rumbaugh, Jacobson & Booch 1999) ja toista painosta (Rumbaugh, Jacobson & Booch 2005). Ensimmäinen painos käsittelee versiota 1.3 ja toinen versiota 2.0. Näiden väliin mahtuu vielä UML:n versiot 1.4 (OMG 2001) ja 1.5 (OMG 2003). Sitä missä versiossa muutokset 1.3 ja 2.0 välillä ovat tulleet ei lähdetä erittelemään.

5.1 Toimintamalli

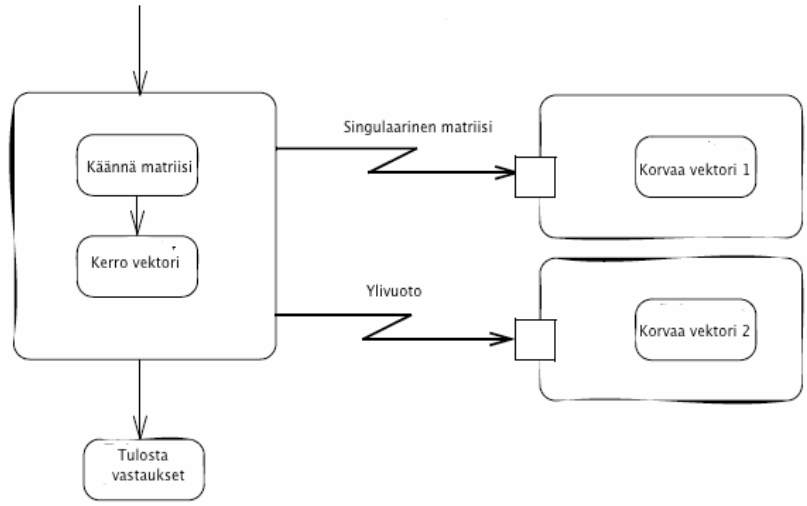
Rumbaughin ym. (2005) mukaan UML 1.3:ssä toimintamalli ja tilamalli perustuivat samaan metamalliin. UML 2:ssa tilamalli ja toimintamalli on erotettu toisistaan. Toimintamalli perustuu nykyään löyhästi Petri-verkkoihin (Petri 1962). Uusia käsitteitä versiossa 2.0 ovat mm. tapit (pin), poikkeukset (exception), keskeytykset (interrupt), laajennusalue (expansion region), tietovirrat (data flow), tietovarastot (data store) ja kontrollivirrat (control flow).

Tapit (pin) liittyvät toimintoihin. Ne esittävät toiminnon syötteitä ja tulosteita. Jokaiselle syöttö- ja tulostusparametrille on oma tappi. Tapeista löytyy erilaisia erikoistapauksia, kuten poikkeuksiin ja virtoihin (stream) liittyvät tapit (Rumbaugh ym. 2005, 520). Kuviossa 19 on esimerkki tappi-notaatiosta. Parametri1 -toiminnosta lähtevä nuoli osoittaa syötetappiin ja Parametri2 –toimintoon viittaava nuoli lähtee tulostetapista.



KUVIO 19. Tapit (OMG 2005)

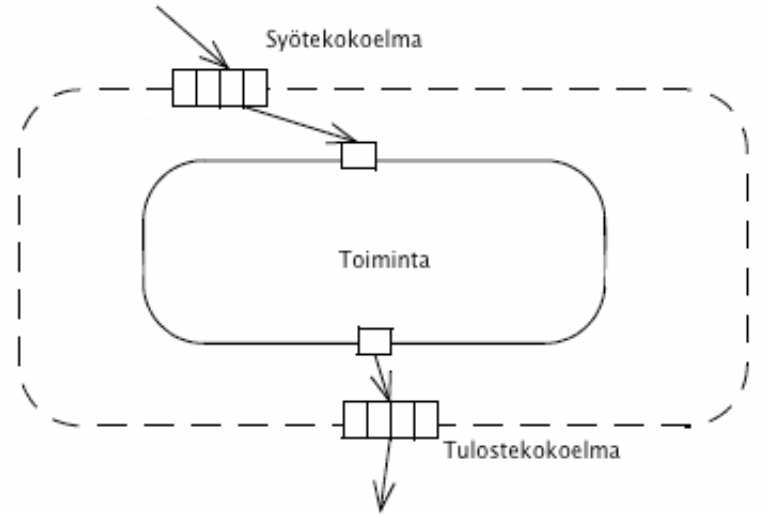
Rumbaughin ym. (2005) mukaan poikkeusten käsittely on muutettu UML 2.0:ssa lähemmäksi ohjelmointikielten poikkeuskäsitettä. Poikkeukset ovat heidän mukaan epätavallisia tilanteita, jotka johtuvat suoritusalustan toiminnallisista virheistä tai jotka on eksplisiittisesti huomioitu mallinnuksessa. Poikkeuksen ilmeneminen keskeyttää normaalin toimintavirran ja aloittaa poikkeuksen käsittelijän suorituksen. Poikkeusten käsittelyyn liittyviä käsitteitä ovat poikkeuksen käsittelytappi (expection pin) ja poikkeusten käsittelijä (exception handler). Kuviossa 20 on kuvattu poikkeusten käsittelyyn liittyvää notaatiota. Salamaa muistuttavat nuolet ovat poikkeusten käsittelijöitä ja ’Korvaa vektori’ -toimintojen kyljessä olevat laatikot ovat poikkeuksen käsittely tappeja.



KUVIO 20. Poikkeusten käsittely (OMG 2005)

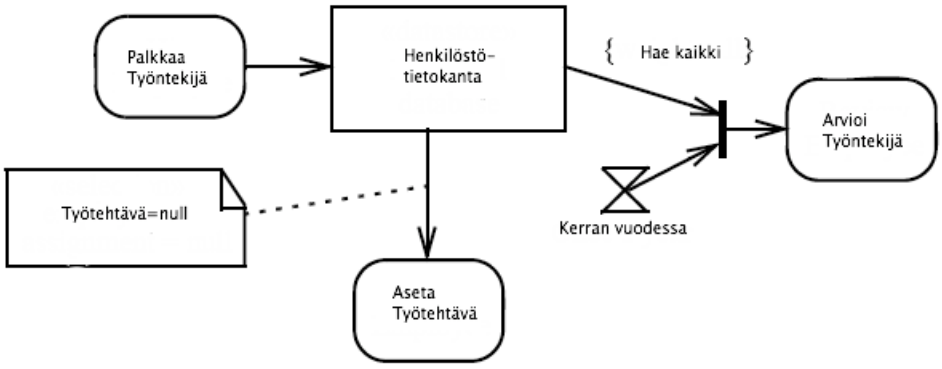
Rumbaughin ym. (2005) mukaan keskeytykset ovat tilanteita, joissa toimintoa ei voida suorittaa loppuun, koska jokin tapahtuma tekee toiminnosta epäoleellisen. Keskeytyksien käsittely on analoginen poikkeusten käsittelyn kanssa.

Laajennusalue (expasion region) on for-all tyyppinen toistorakenne, jossa toimintoja suoritetaan niin monta kertaa kuin arvokokoelmissa (collection values) on alkiota. Arvokokoelmia on kahta tyyppiä, syötekokoelmia ja tulostekokoelmia. Syöte- ja tulostekokoelmien tyyppien ja kokojen täytyy vastata toisiaan. (Rumbaugh ym. 2005, 347) Esimerkki laajennusalueesta löytyy kuviosta 21.



KUVIO 21. Laajennusalue (OMG 2005)

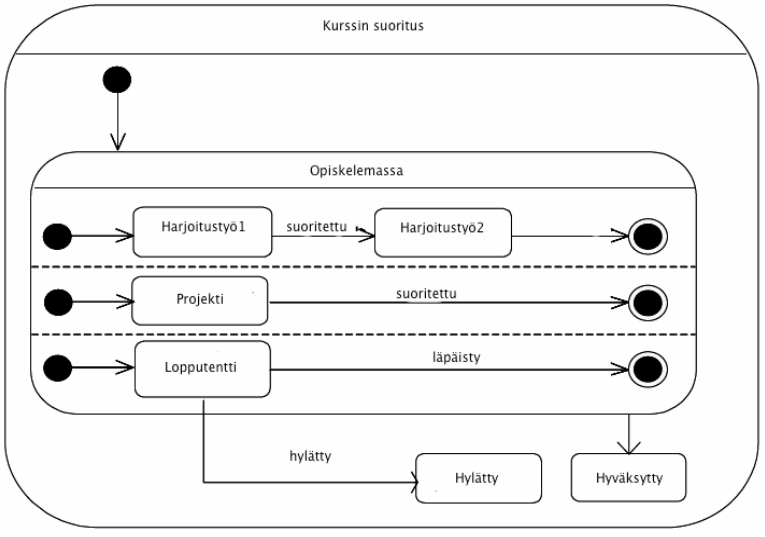
UML 2.0:ssa toimintojen väliset siirtymät (transition) on korvattu kontrolli- ja tietovirroilla (controlflow, dataflow). Rumbaughin ym. (2005) mukaan kontrollivirta on suhde kahden toiminnon välillä, kun taas tietovirta on suhde toiminnon ja tietovaraston välillä (datastore) tai kahden tapin (pin) välillä. Tietovarastokäsite kuvaa pysyvän tiedon varastoa. Kuviossa 22 on esimerkki tietovaraston päivityksestä



KUVIO 22. Tietovarasto (OMG 2005)

5.2 Tilamalli

Tilamalli ei ole muuttunut paljoakaan versioon 2.0 tultaessa, lukuunottamatta oman metamallin käyttöönottoa. Isoimmat muutokset koskevat aluekäsitteen (region) käyttöä komposiittitilojen kuvauksessa. Rinnakkainen komposiittitila (concurrent composite state) on korvattu ortogonaalitilan (orthogonal state) käsitteellä. Ortogonaalitila on jaettu ortogonaalialueisiin (orthogonal region), jotka kaikki toimivat rinnakkain. (Rumbaugh ym. 2005, 503) Komposiittitila, joka ei sisällä rinnakkaista toimintaa, on korvattu ei-ortogonaalisella tilalla (non-orthogonal state). Ei-ortogonaalisen tilan alitiloja suoritetaan aina yksi kerrallaan. (Rumbaugh ym. 2005, 481) Kyseessä on käytännössä nimeämistavan muutos. Notaatioltaan ja merkitykseltään komposiittitilat ja ortogonaalitilat ovat täysin toisiaan vastaavia. Kuviossa 23 on kuvattu ortogonaalitila. Opiskelemassa tila sisältää kolme ortogonaalialuetta.



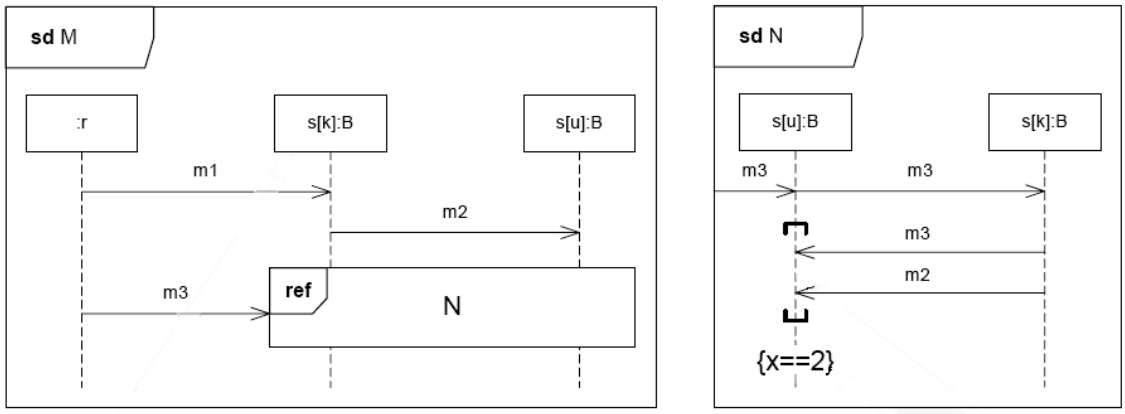
KUVIO 23. Ortogonaalitila (OMG 2005)

5.3 Yhteistoimintamalli

Yhteistoimintamalli on pysynyt lähes muuttumattomana UML:n versioiden 1.3 ja 2.0 välillä. Suurin muutos on nimen vaihdos collaboration diagram:sta communication diagram:ksi. Lisäksi viesti käsitteen notaatiota ja tyyppejä on hieman hieman muutettu.

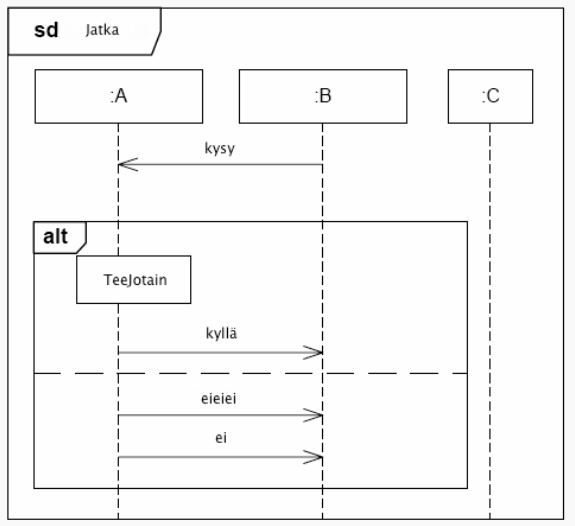
5.4 Sekvenssimalli

Sekvenssimalliin on tullut kaksi merkittävää uudistusta. Ne ovat vuorovaikutuskäyttö (interaction use) ja yhdistetty fragmentti (combined fragment). Vuorovaikutuskäyttö on viittaus toiseen sekvenssimalliin sekvenssimallin sisällä. Se mahdollistaa sekvenssimallien uudelleenkäytön. Kuviossa 24 on kuvattu vuorovaikutuskäyttö. Vasemman puoleinen N:llä merkitty laatikko on viittaus oikealla esitettyyn sekvenssimalliin.



KUVIO 24. Vuorovaikutuskäyttö (OMG 2005)

Yhdistetty fragmentti on alue sekvenssimallin sisällä, joka koostuu operaattorista ja sekvenssimallin komponenteista. Operaattori määrää alueeseen sisältyville sekvenssimallin komponenteille jonkin erikoistoiminnon. Esimerkkejä näistä toiminnoista ovat mm. alt-operaattori, joka mahdollistaa kahden sekvenssimallin alueen vaihtoehtoisuuden. Loop-operaattori taas mahdollistaa sekvenssimallin alueen toimintojen toistamisen. Yhdistettyyn fragmenttiin liittyviä erilaisia operaattoreita on yhteensä kaksitoista kappaletta. (Rumbaugh ym. 2005, 235) Kuviossa 25 on kuvattu yhdistetty fragmentti. Alt-merkinnällä varustettu fragmentti sisältää kaksi vaihtoehtoista sekvenssimallin osaa.



KUVIO 25. Yhdistetty fragmentti (OMG 2005)

5.5 Yhteenveto

Tässä luvussa tutkittiin UML:n kehittymistä versioiden 1.3 ja 2.0 välillä. Eri dynaamiset mallit käytiin läpi yksi kerrallaan, samalla esitellen niiden uusia ominaisuuksia. Luvun tarkoituksena oli valaista miten oliomallintamisen kenttä on muuttunut UML 2.0:aan tultaessa. Tämä on tärkeää jatkotutkimuksen kannalta, jos halutaan tutkia dynaamisen mallintamisen kenttään laajemmin.

Taulukko 4 sisältää uusia dynaamisten mallien käsitteitä. Suurin osa muutoksista löytyi toimintamallista. Muutokset koskivat lähinnä poikkeusten ja keskeytysten käsittelyä. Toiseksi eniten muutoksia löytyi sekvenssimallista Sekvenssimallin muutoksia ovat mm. viittaukset toisiin sekvenssimalleihin sekä erilaiset silmukka- ja vaihtoehtorakenteet. Tilamallin muutokset koskivat lähinnä alitilojen ja rinnakkaisten tilojen kuvausta. Yhteistoimintamalliin ei ollut tullut paljon muutoksia UML 2.0:aan tultaessa. Ainoat muutokset olivat olemassa olevien käsitteiden uudelleennimeämisiä.

Mallien välisiä yhteisiä käsitteitä löydettiin vain yksi.Tämä käsite on alueen käsite. Sitä oli käytetty toimintamallin puolella kuvaamaan silmukkamaista rakennetta. Tilamallin puolella alueella kuvattiin rinnakkaisia toimintoja ja sekvenssimallin puolella sillä kuvattiin toistoa, vaihtoehtoisia toimintoja sekä muita erikoistoimintoja.

TAULUKKO 4. Dynaamisten mallien käsitteet.

	Toimintamalli	Tilamalli	Yhteistoimintamalli	Sekvenssimalli
Tappi	X			
Poikkeus	X			
Keskeytys	X			
Alue	X	X		X
Tietovirta	X			
Tietovarasto	X			
Kontrollivirta	X			
Vuorovaikutuskäyttö				X

6 YHTEENVETO

Tämän tutkimuksen tavoitteena oli tuottaa integroitu metamalli UML:n dynaamisista malleista, ja tutkia sen avulla, missä määrin dynaamiset mallit olivat käsitteistöltään johdonmukaisia, ja missä määrin mallit olivat käsitteellisesti päällekkäisiä. Lisäksi haluttiin tutkia miten UML on kehittynyt ajan mittaan. Näin saatiin kartoitettua käyttäytymisen mallintamista laajemmalla alueella.

Jokaisesta UML:n dynaamisesta mallista annettiin yleiskuvaus, kuvattiin keskeisimmät käsitteet ja tehtiin metamalli. Metamalleista muodostettiin integrointimenettelyä käyttäen globaali metamalli, jossa oli kaikki tärkeimmät UML:n dynaamiseen mallintamiseen liittyvät käsitteet ja käsiterakenteet. Integrointitarkastelulla pyrittiin vastaamaan kahteen tutkimusongelmaan. Näistä ensimmäisessa kysyttiin, ovatko UML:n dynaamiset mallit käsitteistöltään keskenään johdonmukaisia. Tämän tarkastelun yhteydessä huomattiin, ettei dynaamisten mallien käsitteiden välillä ollut merkittäviä epäjohdonmukaisuuksia. Nimeämisristiriitoja löytyi kaksi kappaletta ja rakenneristiriitoja yksi kappale.

Toisessa integrointitarkasteluun liittyvässä tutkimuskysymyksessä tutkittiin, missä määrin UML:n dynaamiset mallit ovat käsitteellisesti päällekkäisiä. Päällekkäisyyksiä havaittiin pääasiassa tila- ja toimintamallien välillä sekä sekvenssi- ja yhteistoimintamallien välillä. Sekvenssi- ja yhteistoimintamallit ovatkin keskenään semanttisesti ekvivalentteja. Kaikkien tai melkein kaikkien mallien välisiä yhteisiä käsitteitä olivat tila, olio ja toiminto.

Tutkimuksessa vertailtiin lisäksi UML:n eri versioiden käyttäjäoppaita ja käsikirjoja. Tämän vertailun avulla haluttiin laajentaa käsitystä käyttäytymisen käsitteistä oliomenetelmissä ja tutkia mitä uusia käsitteitä UML:ään on tuotu version 1.3 jälkeen.

Suurin osa uusista käsitteistä oli tullut toimintamalliin. Toimintamallin uudet käsitteet mahdollistivat poikkeusten ja keskeytysten käsittelyn, tietovirtojen ja –varastojen käytön sekä silmukkamaiset rakenteet. Tilamallissa komposiittitiloihin liittyvää käsitteistöä oli muokattu uudelleen aluekäsitteen avulla. Sekvenssimallin uudet käsitteet taas mahdollistivat mallien uudelleenkäytön sekä erilaiset silmukka- ja vaihtoehtorakenteet. Yhteistoimintamalliin ei ollut tullut varsinaisia uusia käsitteitä.

Paige, Ostroff ja Brooke (2000) esittävät yksikäsitteisyys (uniqueness) periaatteen mallinnuskielen suunnittelulle. Yksikäsitteisyys tarkoittaa sitä, että mallinnuskielessä ei ole päällekkäisiä ja tarpeettomia ominaisuuksia. Samaa asiaa on tarkasteltu tämän tutkielman toisessa tutkimuskysymyksessä. Paigen ym. mukaan sekvenssi- ja yhteistoimintamalli ovat keskenään semanttisesti ekvivalentteja. Tämä näkökulma oli esitetty jo UML:n käyttöoppaassa (Booch, Rumbaugh & Jacobson 1999, 249). Paige ym. esittävät lisäksi ristiriidattomuus periaatteen mukaisesti UML:n rationalisointia. UML:stä pitäisi heidän mukaan tehdä yksinkertaisempi versio, jolla voitaisiin mallintaa yksi asia yhdellä tavalla. Rationalisoinnnin tuloksena olisi ydin-UML (core-UML). Tässä tutkielmassa muodostettu globaalia metamallia voidaan pitää askeleena sitä kohti.

Simons & Graham (1999) esittävät joukon UML 1.3:een liittyviä ongelmia. Niitä on niputettu neljään kategoriaan, joista kolme eli epäjohdonmukaisuudet (inconsistency), monimerkityksellisyydet (ambiquity) ja riittämättömyydet (adequacy) ovat tämän tutkielman kannalta olennaisia. Epäjohdonmukaisuus- ja monimerkityksellisyysongelmia käsiteltiin tämän tutkielman ensimmäisessä tutkimusongelmassa. Tämän tutkielman havainnot ja Simons & Grahamin havainnot epäjohdonmukaisuuksien ja monimerkityksellisyyksien osalta olivat erilaisia, mutta eivät toisaan pois sulkevia. Riittämättömyys kysymyksiä taas tarkasteltiin tämän tutkielman kolmannessa tutkimusongelmassa, kun tarkasteltiin miten UML oli laajentunut versiossa 2.0. Myöskään riittämättömyyskysymyksissä ei tehty samoja havointoja kuin Simons & Graham. Havainnot eivät kuitenkaan ole toisiaan poissulkevia.

Metamallien käsitteiden valinnalle ei ollut hyvää kriteeristöä. Mukaan otettiin suurin piirtein ne käsitteet jotka löytyivät The UML User Guidesta (Booch, Rumbaugh & Jacobson 1999) sekä The UML Reference Manualin esimerkeistä (Rumbaugh, Jacobson & Booch 1999, 527-520). Joitain käsitteitä jätettiin kuitenkin mallintamatta. Esimerkiksi toimintamallista jätettiin pois signaalien lähettäminen ja vastaanotto (Rumbaugh, Jacobson & Booch 1999, 141) ja tilamallista jätettiin mallintamatta historiatilat (Booch, Rumbaugh & Jacobson 1999, 300). Käsitteiden pois jättö oli välttämätöntä, sillä muuten metamalleista olisi tullut liian monimutkaisia.

Integrointityön aikana tehtiin muutamia huomiota Batinin menetelmän sopivuudesta tutkimusmenetelmäksi. Ensinnäkin Batinin integraatiotekniikka oli alunperin tarkoitettu ER-mallien integrointiin. Tässä työssä sitä sovellettiin UML:n luokkakaavioihin. ER-mallit ja UML:n luokkakaaviot olivat kuitenkin niin lähellä toisiaan ettei ongelmia syntynyt. Toinen huomio oli, että Batinin tekniikka oli tarkoitettu useamman henkilön tekeminen mallien johdonmukaiseen yhdistämiseen. Koska tässä työssä mallintajia oli vain yksi, ei eri mallien välillä ollut suuria eroja. Lisäksi ristiriidat mallien välillä johtuivat pääasiassa mallintajan tekemistä nimeämis- ja käsitevalinnoista. Ne eivät välttämättä kuvaa UML:n todellisia johdonmukaisuusongelmia. Integrointilähestymistapa on siis epäluotettava tapa johdonmukaisuusongelmien selvittämiseen, koska se riippuu liikaa mallintajan subjektiivisista valinnoista.

Työn tuloksia voidaan käyttää pohjana, kun lähdetään tutkimaan yleisemmin dynaamisia malleja. Kun eri oliomenetelmien dynaamisten mallien metamalleja integroidaan tämän työn globaaliin metamalliin, voidaan selvittää mitä käyttäytymisen käsitteitä löytyy UML:n ulkopuolelta. Globaali metamalli voisi toimia pohjana, kun muodostettaisiin yleinen globaali metamalli käyttäytymisen käsitteistä oliomenetelmissä. Yleisen globaalin metamallin muodostamista on pohjustettu tässä työssä tutkimalla UML 2.0:n tuomia uusia käsitteitä dynaamisten mallien kentälle.

LÄHTEET

Batini C., Ceri S. & Navathe S. B. 1992. Conseptual Database Design: An Entity Relationship Approach. Redwood City, California: The Benjamin/Cummings Publishing Company.

Batini C., Lenzeri M. & Navathe S. B. 1986. A comparative analysis of methodologies for database schema integration. Computing surveys 15(4), 323-364.

Booch G. 1994. Object-Oriented Analysis and Design with Applications, 2nd Edition. Redwood City, California: Benjamin/Cummings.

Booch G., Rumbaugh J., & Jacobson I. 1999. The Unified Modeling Language User Guide. USA: Addison-Wesley.

Booch G., Rumbaugh J., & Jacobson I. 2005. The Unified Modeling Language User Guide Second Edition. USA: Addison-Wesley.

Chen P. 1976. The entity-relationship model – toward a unified view of data. ACM Tranactions on Database Systems 1(3), 9-36.

Halttunen V., Lehtinen A. & Nykänen R. 2006. Building a Conceptual Skeleton for Enterprise Architecture Specifications. Teoksessa Kiyoki Y. (toim.) Information Modeling and Knowledge Bases XVIII. IOS Press.

Harmsen F. 1997. Situational Method Engineering. The Netherlands: University of Twente.

Henderson-Sellers B. & Bulthuis A. 1997. Object-oriented Metamethods. USA: Springer-Verlag.

Heym M. & Österle H. 1992. A reference model for information systems development. Teoksessa Kendall K., Lyytinen K. & DeGross J. (toim.) Proceedings of the IFIP WG 8.2 Working Conference on the Impacts on Computer Supported Technologies on Information Systems Development, Minneapolis, Minnesota, June 14-17. Amsterdam: North-Holland, 215-239.

ISO 1990. International Standard. Information Resource Dictionary Standard (IRDS) – Framework ISO/IEC 10027.

Jacobson I., Christerson M., Jonsson P. & Övergaard G. 1992. Object-Oriented Software Engineering: A Use Case Driven Approach. Wokingham, England: Addison-Wesley.

Jarke M., Klamma R. & Lyytinen K. 2003. Meta Modeling. Julkaisematon working paper. Jyväskylä: Jyväskylän yliopisto.

Jarke M., Pohl K., Weidenhaupt K., Lyytinen K., Marttiin P. & Tolvanen J.-P. 1998. Meta modeling: A formal basis for interoperability and adaptability. Teoksessa B. Krämer, M. Papazoglou & H. Schmidt (toim.) Information Systems Interoperability. Somerset: John Wiley and Sons, Research Studies Press,229-264.

Kaivola T. 1991. Näkemysten integrointi. Pro Gradu -tutkielma. Jyväskylä: Jyväskylän yliopisto.

Kelly S., Lyytinen K. & Rossi M. 1996. Metaedit+: A Fully configurable Multi-User and Multi-Tool CASE and CAME Environment. Teoksessa Vassilou Y. & Mylopoulus J.(toim.) Proceedings of the 8th Conference on Advanced Information Systems Engineering, Heraklion, Crete, Greece, May 20-24. Springer, 1-21.

Kleppe A., Warner J & Bact W. 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Boston: Addison-Wesley.

Koskimies K., Koskinen J., Maunumaa M., Peltonen J., Selonen P., Siikarla M. & Systä T. 2004. UML työvälineenä ja tutkimuskohteena. Tietojenkäsittelytiede 21, 19-51.

Koskinen M. & Marttiin P. 1997. Process support in MetaCASE: implementing the conceptual basis for enactable process models in MetaEdit+. Teoksessa J. Ebert & C. Lewerentz (toim.) Software Engineering Environments. Los Alamitos: IEEE Computer Society Press, 110-123.

Kruchten P. 2003. The Rational Unified Process: An Introduction. USA: Addison-Wesley.

Leppänen M. 1994. Metamodelling: concepts, benefits and pitfalls. Teoksessa J. Zupancic & S. Wrycza (toim.) Proceedings of the Fourth International Conference on Information Systems Development - ISD'94, Bled, Slovenia, 126-137.

Leppänen M. 2000. Towards a Method Engineering Method with an Emphasis on the Consistency of ISD Methods. Teoksessa Siau K.(toim.) Proceedings of the Fifth CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design Stockholm, Sweden, June 5-6.

Metacase. 2006. Metaedit+ 4.5 [online]. Saatavilla www-muodossa <http://www.metacase.com> [viitattu 30.3.2007]

Nijssen G. & Halpin T. 1989. Conceptual Schema and Relational Database Design: A Fact Oriented Approach. Sydney: Prentice-Hall.

Objecteering Software. 2006. Objecteering/UML 5.3 [online]. Saatavilla www-muodossa <http://www.objecteering.com/downloads.php> [viitattu 30.3.07]

OMG. 2000a. Meta Object Facility (MOF) Specification 1.3 [online]. Saatavilla www-muodossa <http://www.omg.org/cgi-bin/doc?formal/00-04-03.pdf> [viitattu 15.12.02].

OMG. 2000b. OMG Unified Modeling Language Specification 1.3 [online]. Saatavilla www-muodossa <http://www.omg.org/cgi-bin/doc?formal/00-03-01> [viitattu 09.12.02].

OMG. 2001. OMG Unified Modeling Language Specification Version 1.4 [online]. Saatavilla www-muodossa <http://www.omg.org/cgi-bin/doc?formal/01-09-67> [viitattu 16.03.07].

OMG. 2003. OMG Unified Modeling Language Specification Version 1.5 [online]. Saatavilla www-muodossa <http://www.omg.org/cgi-bin/doc?formal/03-03-01> [viitattu 16.03.07].

OMG. 2005. Unified Modeling Language: Superstructure version 2.0 [online]. Saatavilla www-muodossa <http://www.omg.org/cgi-bin/doc?formal/05-07-04> [viitattu 10.12.2006]

Paige R.F., Ostroff P.J. & Brooke P.J. 2000. Principles for Modeling Language Design. Information and Software Technology 42, 665-675.

Pastor E.A. & Price R.T. 1997. Using Metamodels of Methodologies to Determine the Needs for Reusability support. Teoksessa M. Harandi (toim.) Proceedings of the 1997 symposium on Software reusability Boston, Massachusetts, United States, May 17–20. New York: ACM Press, 121–129.

Petri C.A. 1962. Kommunikation mit Automaten. Bonn: Institut fur Instrumentelle Mathematik.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. & Lorensen W. 1991. Object-Oriented Modeling and Design. Englewood Cliffs, N.J.: Prentice Hall.

Rumbaugh J., Jacobson I. & Booch G. 1999. The Unified Modeling Language Reference Manual. USA: Addison-Wesley.

Rumbaugh J., Jacobson I. & Booch G. 2005. The Unified Modeling Language Reference Manual, Second Edition. USA: Addison-Wesley.

Saeki M. 1995. Object-oriented Metamodeling. Teoksessa Papazoglou M. P. (toim.) OOER '95: Object-Oriented and Entity-Relationship Modeling, LNCS 1021. Berlin: Springer, 250-259.

Selic B., Ramackers G. & Kobryn C. 2002. Evolution, not Revolution. Communications of the ACM 45(11), 70-72.

Simons A. J. H. & Graham I. 1999. 30 Things That Go Wrong in Object Modeling with UML 1.3. Teoksessa H. Kilov, B. Rumpe & I. Simmonds (toim.) Behavioural Specification of Businesses and Systems. Dordrecht: Kluwer, 237-258.

Smolander K. 1991. OPRR - a model for modelling systems development methods. Teoksessa V-P. Tahvanainen & K. Lyytinen (toim.) Proceedings of the Second Workshop on The Next Generation of CASE-tools, Trondheim, Norway, May 11-12. Jyväskylä: University of Jyväskylä, 135-151.

Smolander K. 1993. GOPRR: a proposal for a meta level model. Jyväskylä: University of Jyväskylä.

Tigris.org. 2007. AgroUML 0.24 [online]. Saatavilla www-muodossa < http://argouml-downloads.tigris.org/argouml-0.24/> [viitattu 30.3.07]

Tolvanen J-P. 1998. Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence. Studies in Computer Science, Economics and Statistics 47. Jyväskylä: Jyväskylän yliopisto.

Tolvanen J-P & Rossi M. 1996. Metamodeling approach to method comparison: A survey to a set of ISD methods. Jyväskylä: Jyväskylän yliopisto.

van Hillegersberg J. & Kumar K. 1999. Using Metamodeling to Integrate Object-Oriented Analysis, Design and Programming Concepts. Information Systems 24(2), 113-129.

Welke R.J. 1992. The CASE repository: More than another Database Application. Teoksessa Cotterman W.W. & Senn J.A. (toim.) Challenges and strategies for research in systems development. Chichester:John Wiley & Sons, 181-214.

Venable J. 1993. CoCoA: A Conceptual Data Modeling Approach for Complex Problem Domains. USA: State University of New York at Binghampton.