hyväksymispäivä arvosana

arvostelija

Malliperustainen ohjelmistokehitys ja malliperustainen arkkitehtuuri

Henri Karhatsu

Helsinki 4.2.2009

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section	Laitos – Institution – Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytiede

Tekijä – Författare – Author
Henri Karhatsu

Työn nimi – Arbetets titel – Title
Malliperustainen ohjelmistokehitys ja malliperustainen arkkitehtuuri

Oppiaine – Läroämne – Subject
Seminaari: Palvelusuuntautuneet järjestelmät

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Seminaari	4.2.2009	16

Tiivistelmä – Referat – Abstract

Tässä seminaaripaperissa tutustutaan malliperustaisen ohjelmistokehityksen (MDE) taustoihin sekä esitellään sen perusperiaatteet ja -käsitteet. Näitä ovat mallit, metamallit, metametamallit sekä mallimuunnokset. Lisäksi listataan MDE:n höytyjä, joista keskeisin liittyy abstraktiotason nostamiseen.

Yksi MDE:n variantti on nimeltään malliperustainen arkkitehtuuri (MDA), joka paperissa myös esitellään. Sen oleelliset käsitteet ovat alustariippumaton malli (PIM) ja alustariippuvainen malli (PSM). Näistä ensimmäisen on korkeammalla abstraktiotasolla ja se sisältää järjestelmän mallit kuvattuna niin, ettei kuvauksissa oteta kantaa toteutusalustaan. Alustariippumattomasta mallista pystytään luomaan koodi jollekin toteutusalustalle, jolloin puhutaan alustariippuvaisesta mallista.

Seminaaripaperissa tutustutaan myös yhteen MDA-työkaluun nimeltä AndroMDA. Se on avoimen lähdekoodin MDA-muunnostyökalu, joka pystyy luomaan UML-malleista lähes valmista sovelluskoodia.

ACM Computing Classification System (CCS):
D.2.2 Design Tools and Techniques

Avainsanat – Nyckelord – Keywords
malliperustainen ohjelmistokehitys, malliperustainen arkkitehtuuri, MDE, MDA, UML, MOF, AndroMDA

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto 1

2 Malliperustainen ohjelmistokehitys, MDE 2

2.1 Ohjelmistokehitys ja abstraktiotasojen nousu 2

2.2 Malliperustaisen ohjelmistokehityksen peruskäsitteet 3

2.3 MDE:n hyötyjä 6

3 Malliperustainen arkkitehtuuri, MDA 8

3.1 MDA:n abstraktiotasot 8

3.2 MDA:n peruskomponentit 10

4 AndroMDA – esimerkki MDA-työkalusta 13

5 Yhteenveto 16

Lähteet 17

1 Johdanto

Ohjelmistokehityksessä pyritään jatkuvasti tekemään entistä laadukkaampia ohjelmistoja entistä halvemmalla. Samanaikaisesti käyttäjien ja asiakkaiden vaatimukset kasvavat niin, että ohjelmistoista tulee koko ajan monimutkaisempia. On kuitenkin kyseenalaista, pystyykö tällä hetkellä voimassa oleva olioteknologia vastaamaan näihin tarpeisiin.

Ohjelmistokehityksen historiassa tuottavuutta ja laatua on pystytty parhaiten parantamaan abstraktiotasoa nostamalla. Olio-ohjelmoinnista seuraava taso ylöspäin on malliperustainen ohjelmistokehitys, jota tässä seminaaripaperissa käsitellään. Sen keskeisin elementti ovat mallit, joiden avulla kuvataan rakennettavan järjestelmän sovelluslogiikka. Varsinainen sovelluskoodi generoidaan automaattisesti mallien perusteella.

Seminaaripaperi jakaantuu kolmeen osaan. Aluksi tutustutaan malliperustaiseen ohjelmistokehitykseen yleisellä tasolla. Toisessa osassa käsitellään yhtä sen varianttia, malliperustaista arkkitehtuuria. Lopuksi tutustutaan malliperustaisen arkkitehtuurin työkaluun nimeltä AndroMDA.

Malliperustaisesta ohjelmistokehityksestä käytetään yleisesti lyhennettä MDE (*model-driven engineering*). Vastaavasti malliperustaisen arkkitehtuurin lyhenne on MDA (*model-driven architecture*). Näitä lyhenteitä käytetään kattavasti myös tässä paperissa.

2 Malliperustainen ohjelmistokehitys, MDE

Tässä luvussa tutustutaan aluksi lyhyesti ohjelmistokehityksen historiaan ja siihen, miksi malliperustainen ohjelmistokehitys (MDE) on syntynyt. Sen jälkeen esitellään MDE:n perusperiaatteet ja tutustutaan MDE:n keskeisiin komponentteihin eli abstraktiotasoihin, mallimuunnoksiin sekä mallien välisiin suhteisiin. Lopuksi luetellaan joukko MDE:n höytyjä.

2.1 Ohjelmistokehitys ja abstraktiotasojen nousu

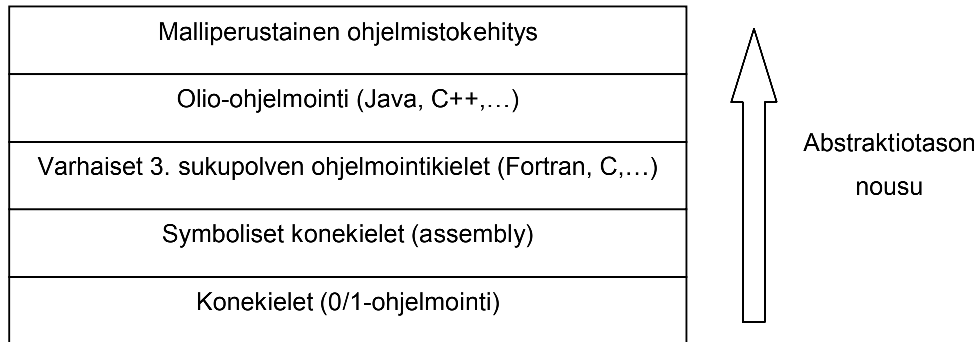
Jotta voi ymmärtää hyötyjä, joita malliperustaisessa ohjelmistokehityksessä haetaan, on tärkeää ymmärtää hieman ohjelmistokehityksen historiaa. Aivan ensimmäiset ohjelmoijat joutuivat ohjelmoimaan käyttäen natiiveja CPU-komentoja [Fra03]. Käytännössä tämä tarkoitti ohjelmoimista ykkösien ja nollien avulla. Vaikka näin päästiinkin tehokkaisiin ohjelmiin, oli tällainen luonnollisesti hidasta ja sovellusalueet hyvin rajattuja.

Ohjelmoinnin ensimmäinen merkittävä kehitysaskel oli, kun keksittiin symbolinen konekieli assembly [Fra03]. Sen avulla ohjelmoijat pystyivät kirjoittamaan virhealttiiden ja vaikeasti muistettavien nolla-yksi-numerosarjojen sijaan enemmän kirjoitetun kielen kaltaisesti. Kehitys jatkui edelleen, kun keksittiin ensimmäiset niin sanotut kolmannen sukupolven ohjelmointikielet kuten Fortran [Sch06]. Ne suojasivat ohjelmoijaa monilta konekielten monimutkaisuuksilta. Esimerkiksi yksinkertainen PRINT-komento Fortran-kielessä korvasi kymmeniä tai jopa satoja rivejä assembly-koodia [Fra03].

Modernit kolmannen sukupolven ohjelmointikielet kuten C++, Java ja C# ovat esimerkkejä olio-ohjelmointikielistä. Verrattuna edeltäjiinsä niissä keskeistä on koodin uudelleenkäyttö, joka ilmenee muun muassa luokkakirjastoina ja sovelluskehyksinä [Sch06] sekä komponenttipohjaisuutena ja suunnittelumallien hyödyntämisenä [Fra03].

Yhteistä ohjelmointikielten kehityksessä on ollut se, että jokainen kehitysaskel on tarkoittanut abstraktiotason nousua [Sch06]. Toisin sanoen ohjelmoijat ovat koko ajan siirtyneet kauemmaksi laitteistotasolta. Ohjelmistokehityksen tehokkuudessa tämä on samalla tarkoittanut sitä, että ohjelmistojen elinkelpoisuuden parametrit – tuotanto-kustannukset, laatu ja pitkäikäisyys – ovat parantuneet [Fra03].

Tällä hetkellä olioparadigma on edelleen hallitseva periaate. On kuitenkin kyseenalaista, voidaanko sen avulla ratkaista enää jatkuvasti monimutkaistuvien ohjelmistojen haasteita ja voiko se enää vastata tuotantokustannusten, laadun ja pitkäikäisyyden paineeseen [Fra03]. Vaikuttaa siltä, että näiden elinkelpoisuusparametrien nostaminen uudelle tasolle tarkoittaa samalla sitä, että myös abstraktiotasoa täytyy nostaa [Béz05]. Yksi mahdollisuus ratkaista tämä haaste on siirtyä oliokeskeisyydestä malli-keskeisyyteen eli malliperustaiseen ohjelmistokehitykseen (kuva 2.1).

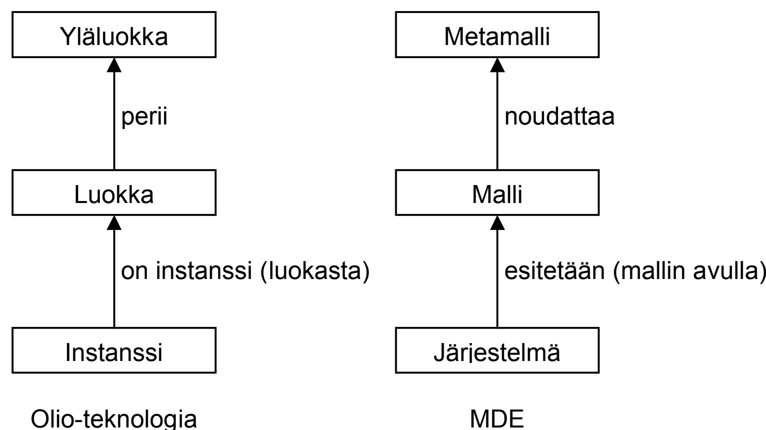


Kuva 2.1. Abstraktiotason nousu ohjelmointikielten kehityksessä

2.2 Malliperustaisen ohjelmistokehityksen peruskäsitteet

Kun olio-ohjelmoinnissa keskeistä on ”kaikki ovat olioita”-ajattelu, voidaan malli-keskeisessä ohjelmistokehityksessä vastaavasti sanoa, että ”kaikki ovat malleja” [Béz05]. Tällä tarkoitetaan siis sitä, että kun oliomaailmassa kaikista reaalimaailman asioista pyritään muodostamaan luokkia ja olioita, niin MDE:ssä vastaava tehdään malleja käyttäen. Kyse on MDE:n perusperiaatteesta, joka ohjaa malliparadigmaa yksin-kertaisuuteen ja yleisyyteen [Béz05].

Malliperustaisen ohjelmistokehityksen tärkein käsite on luonnollisesti malli. Sillä on tässä yhteydessä kaksi keskeistä ominaisuutta. Ensinnäkin sanotaan, että malli esittää (*represents of*) järjestelmää [BJT05]. Tämä liittyy läheisesti malli-sanan yleiseen määritelmään, jonka mukaan malli on yksinkertaistettu mutta kuitenkin tarkka ja uskottava esitys järjestelmästä [BJB08]. Toisekseen mallin sanotaan noudattavan (*conforms to*) metamallia [BJT05]. Metamallilla tarkoitetaan kieltä, jonka avulla voidaan kuvata malli, eli metamalli on siis mallia yhden abstraktiotason korkeammalla oleva käsite. Käsitteet *esitys* ja *noudattaa* voidaan rinnastaa olioteknologian käsitteisiin *instanssi* ja *periä* (kuva 2.2).



Kuva 2.2. Olioteknologian ja MDE:n peruskäsitteet [Béz05]

Mallin ja metamallin suhdetta voidaan selventää muutamilla esimerkeillä [Béz05]. Esimerkiksi kartan avulla voidaan mallintaa todellisuutta. Kartta on siis esitys todellisuudesta. Kartan selitteiden (värit, symbolit, mittakaava jne.) taas voidaan ajatella olevan metamalli. Kartta (malli) siis noudattaa selitteitään (metamalli). Vastaava malli-metamalli-suhde on olemassa ohjelman ja sen ohjelmointikielen sekä vaikkapa XML-dokumentin ja DTD:n välillä.

Mihin malleja sitten malliperustaisessa ohjelmistokehityksessä oikeastaan käytetään? Luonnollisesti niiden avulla kuvataan eli mallinnetaan rakennettavaa järjestelmää, mutta miten tämä eroaa esimerkiksi siitä, että vaikkapa UML:n (*Unified Modeling Language*) luokkakaavion avulla kuvataan luokkien välisiä suhteita niin, että kaavion perusteella osataan toteuttaa osa järjestelmästä?

MDE:n yksi keskeisimmistä ominaisuuksista on, että siinä mallit eivät ole vain suunnittelua ja varsinaista toteutusta tukevia informaation välineitä vaan MDE:ssä malleja käytetään ikään kuin ohjelmointikielinä [Fra03]. Tämä tarkoittaa siis sitä, että toteutettavan järjestelmän ominaisuudet ”ohjelmoidaan” suoraan malleihin sillä tarkkuudella, että niistä voidaan generoida ohjelman lähdekoodi. Lisäksi verrattuna perinteiseen mallien käyttöön, kaikki ohjelmaan tehtävät muutokset pyritään tekemään suoraan malleihin, joista voidaan jälleen generoida päivitetty koodi [Béz05]. Perinteisestihän esimerkiksi UML-kaavioiden päivittäminen unohtuu helposti, kun järjestelmää muokataan.

Lähdekoodin generoimisessa mallien perusteella on paljon samaa kuin CASE-työkaluissa (*computer-aided software engineering*) 1980-luvulla. Niissä oli tavoitteena luoda järjestelmästä graafinen esitys esimerkiksi tilakoneen avulla ja generoida sen perusteella koodi. CASE-menetelmät eivät kuitenkaan koskaan saavuttaneet suurta suosiota. Ensinnäkin yhtä yleistä esitystä oli vaikeaa linkittää silloisille alustoille, joista puuttuivat muun muassa kaikki QoS-palvelut (*quality of service*) kuten läpinäkyvä jakelu, vikasietoisuus ja tietoturva. Lisäksi generoitu koodi oli liian monimutkaista ja graafiset esitykset joustamattomia erilaisiin sovellustarpeisiin. [Sch06]

Verrattuna 1980-luvun CASE-ohjelmointiin MDE:ssä on tiettyjä muutoksia, jotka tekevät siitä edeltäjiään käyttökelpoisemman. Ehkä tärkein niistä on se, että MDE:ssä ei käytetä yhtä yleistä notaatiota mallinnukseen vaan kuvaus voidaan tehdä kohdealueen semantiikan ja syntaksien mukaan [Sch06]. Lisäksi kuvan 2.2 periaate luo kokonaan uuden tavan ajatella, kuinka tällaisia arkkitehtuureja ja sovelluskehyksiä rakennetaan [Béz05]. Myös ulkoiset tekijät ovat suotuisampia uusille CASE-työkaluille. Koska nykyiset alustat (*platforms*) ovat paljon monipuolisempia toiminnallisuuksiltaan ja QoS-ominaisuuksiltaan verrattuna 1980–90-lukujen vastaaviin, MDE-työkaluilla voidaan generoida aikaisempaa ylemmän tason kuvauksia [Sch06]. Esimerkiksi sen sijaan että jouduttaisiin ohjelmoimaan suoraan käyttöjärjestelmätason ohjelmointirajapintoja käyttäen, voidaankin hyödyntää väliohjelmistojen rajapintoja.

Edellä mainittu kohdealueen semantiikan kuvaaminen viittaa kohdealuekohtaisiin mallinnuskieliin (*domain-specific modeling languages, DSML*). DSML:t ovat

metamallien avulla kuvattuja kieliä, joiden ”tyyppijärjestelmät formalisoivat sovelluksen rakenteen, käyttäytymisen ja vaatimukset” [Sch06]. Käytännössä tämä tarkoittaa sitä, että jokaiselle kohdealueelle sovitaan oma mallinnuskieli, joka sisältää juuri tälle kohdealueelle kuuluvia käsitteitä. Näin ollen pystytään generoimaan koodia, jota tämän kohdealueen sovelluksessa tarvitaan.

MDE:en liittyy vielä yksi keskeinen termi eli mallimuunnokset tai mallitransformaatiot (*model transformations*). Niillä tarkoitetaan sitä osaa malliperustaisessa ohjelmistokehityksessä, jossa malli muutetaan joko toiseksi malliksi (*model-to-model, M2M*) tai tekstiksi (*model-to-text, M2T*) [StC08]. Tässä yhteydessä lähdekoodi luetaan tekstiksi. Muuntaminen tapahtuu muunnosmoottoreiden ja -generaattoreiden (*transformation engines / generators*) avulla. Niiden tehtävänä on analysoida mallia ja luoda sen perusteella erilaisia tuotoksia (*artifacts*) [Sch06]. Näitä ovat muun muassa lähdekoodi, syötteet ohjelman simulointia varten sekä vaihtoehtoiset mallikuvaukset.

2.3 MDE:n hyötyjä

Kuten luvussa 2.1 todettiin, MDE on keino nostaa järjestelmien elinkelpoisuutta eli pienentää tuotantokustannuksia sekä parantaa rakennettavan järjestelmän laatua ja pitkäikäisyyttä. Tässä luvussa käsitellään tarkemmin näiden parametrien taustalla olevia asioita.

Keskeisin MDE:n höydyistä on jo edellä todettu abstraktiotason nouseminen. Käytännössä tämä näkyy esimerkiksi siinä, että sovelluskehittäjän ei tarvitse puuttua moniin järjestelmän rakenteellisiin yksityiskohtiin. Näillä tarkoitetaan sellaisia osia järjestelmästä, joita varsinainen sovelluslogiikka ei tarvitsisi mutta jotka toimiva ohjelmisto vaatii. Ilmiö on sama kuin olio-ohjelmoinnin puolella sovelluskehysten käytössä.

Kun malleista pystytään rakentamaan tuotoksia automaattisesti, se auttaa pitämään sovellustoteutuksen ja analyysi-informaation konsistenttina [Sch06]. Eli koska sovelluksen koodi luodaan suoraan mallien perusteella, on se yhteneväinen niiden kanssa – toki olettaen, että mallimuunnokset toimivat oikein.

Yksi MDE:n hyödyistä on se, että samoista malleista pystytään luomaan erilaisia toteutuksia. Generoitu koodi voi olla esimerkiksi HTML:ää, XML:ää, WSDL:iä tai IDL:ää [Fra03]. Vastaavasti toteutusympäristöksi voidaan poimia vaikkapa erilaisia Javaan liittyviä komponentteja taikka esimerkiksi Microsoftin .NET-ympäristö. Tähän liittyy se hyöty, että ainakin periaatteessa mallit ovat siirrettävissä uusille alustoille teknologioiden kehittyessä [Fra03].

MDE on hyödyllinen myös siksi, että sen avulla voidaan huomata virheitä hyvin aikaisessa vaiheessa [Sch06]. Tämä perustuu siihen, MDE-työkalut tarkastavat mallien rakenteen mallimuunnosten aikana ja varmistavat, että ne noudattavat kohdealueen metamallia.

3 Malliperustainen arkkitehtuuri, MDA

Luvussa 2 käsiteltiin malliperustaista ohjelmistokehitystä, MDE:tä. Tässä luvussa tutustutaan malliperustaiseen arkkitehtuuriin (*model-driven architecture, MDA*). Näistä MDE on yleinen lähestymistapa ohjelmistokehitykseen, kun taas MDA on eräs MDE:n muunnelma [Béz05]. MDE siis sisältää MDA:n mutta ei ole rajoitettu siihen [BBJ07].

MDA:n taustalla on Object Management Group -organisaatio (OMG), joka on kehittänyt muun muassa UML:n ja CORBA:n [OMG09]. OMG julkaisi ensimmäisen *white paper*in MDA:sta vuonna 2000 [ks. Sol00] ja varsinaisen MDA-oppaan vuonna 2003 [ks. MDA03].

3.1 MDA:n abstraktiotasot

Ennen kuin käsitellään MDA:n komponentteja ja niiden suhteita tarkemmin, on syytä puhua eri abstraktiotasoista, joilla MDA:ssa toimitaan. Kyse on käsitteistä, jotka OMG on luonut mutta jotka ovat sittemmin levinneet yleisempäänkin MDE-termistöön.

MDA:ssa on kolme abstraktiotasoa [MDA03]: laskentariippumaton malli (*Computation Independent Model, CIM*), alustariippumaton malli (*Platform Independent Model, PIM*) sekä alustariippuvainen malli (*Platform Specific Model, PSM*). Kaikki nämä ovat tietyllä abstraktiotasolla olevia näkymiä järjestelmään.

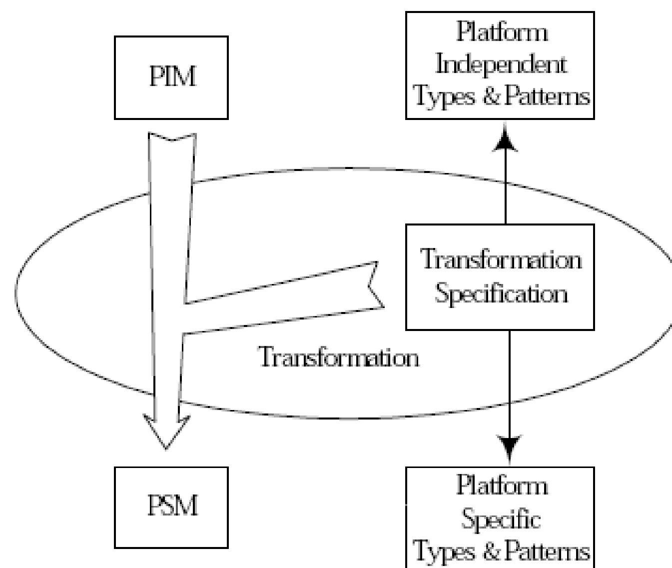
Korkeimmalla abstraktiotasolla on CIM, jonka tehtävänä on kuvata järjestelmän vaatimuksia mutta joka ei ota kantaa järjestelmän rakenteeseen. Se voi olla hyödyllinen kommunikaation väline sovellusalueen asiantuntijoiden ja suunnittelun asiantuntijoiden välillä. Joskus tällaista mallia kutsutaan kohdealuemalliksi (*domain model*) tai liike-toimintamalliksi (*business model*). [MDA03]

CIM:ää merkittävämpi abstraktiotaso on kuitenkin PIM. Sen tehtävänä on kuvata järjestelmän rakenne niin, ettei toteutusalustaan oteta mitään kantaa [MDA03]. Toisin sanoen PIM-tasolla olevia malleja ei ole sidottu esimerkiksi tiettyyn ohjelmointikieleen,

väliohjelmistoon tai tiedonesitystapaan [Fra03]. Tämä tarkoittaa samalla sitä, että PIM-tasolla kuvattu malli voidaan muuntaa usealle eri toteutusalustalle ilman, että itse mallia tarvitsee muokata. PIM sovitetaan johonkin arkkitehtuurityyliin. Yksi tavallisimmista arkkitehtuurityyleistä ovat kerrosarkkitehtuurit [BCK98], joita myös MDA:ssa usein hyödynnetään.

Alin abstraktiotaso PSM on jonkin alustan näkökulma järjestelmään [MDA03]. Tällä tasolla kerrotaan, kuinka alustariippumaton malli toteutetaan tietyssä alustassa tai yleensä usean alustan yhdistelmässä.

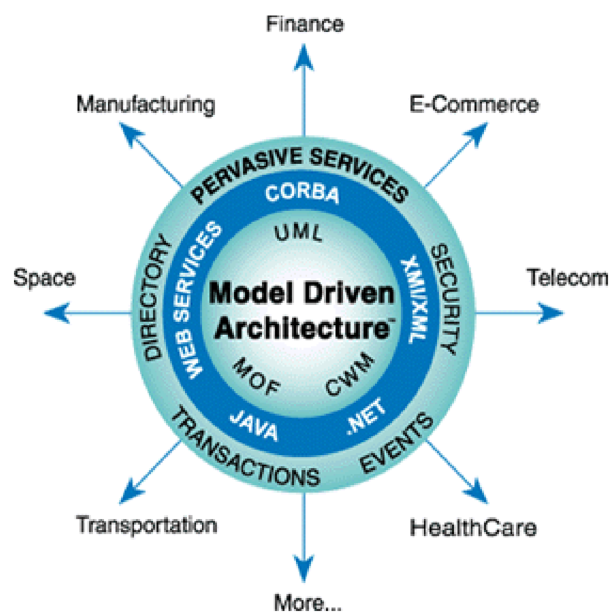
Abstraktiotasoihin liittyvät keskeisesti mallimuunnokset, joista kerrottiin luvun 2.2 lopussa. Kun järjestelmä on kuvattu sovellusriippumattomalla mallilla, siitä voidaan generoida koodi valitulle alustalle (kuva 3.1). Tämä edellyttää sitä, että on olemassa muunnosmääritelmät (*transformation specification*), jotka kertovat, mitä jokin mallin elementti PIM-tasolla tarkoittaa koodissa PSM-tasolla.



Kuva 3.1. MDA:n mallimuunnos PIM:n ja PSM:n välillä [MDA03]

3.2 MDA:n peruskomponentit

Kuvassa 3.2 on OMG:n näkemys MDA:n peruskomponenteista. Arkkitehtuurin ydin perustuu OMG:n mallinnusstandardeihin, joita ovat MOF, UML ja CWM [Sol00]. Kyseiset standardit ovat työkaluja, joita hyödynnetään PIM-tasolla. Kuvassa olevat Java, Web Services, CORBA, XMI/XML ja .NET vastaavasti sijoittuvat PSM-tasolle. On kuitenkin tärkeää huomata, että PSM-tasolla olevat teknologiat eivät ole ainoita mahdollisia. MDA:ta voi siis soveltaa myös muihinkin toteutusalustoihin.



3.2. OMG:n malliperustainen arkkitehtuuri [OMG09]

Luvussa 2.2. käsiteltiin malleja ja metamalleja. Siinä todettiin, että metamalli tarkoittaa kieltä, jonka avulla voidaan kuvata malli. Jotta erilaiset metamallit voisivat kommunikoida keskenään, tarvitaan lisäksi kieli, jolla voidaan kuvata metamalli. Tällaista kieltä kutsutaan metametamalliksi [BJT05].

MDA:n uniikki metametamalli on nimeltään MOF (*Meta Object Facility*) [Béz05]. Se on kieli jolla kuvataan UML-metamalli tai yleisemmin jokin muu metamalli [Fra03]. Vaikka OMG sisällyttääkin UML:n yhdeksi ydinkomponenteista, ei se siis ole ainoa vaihtoehto. Itse asiassa kuvassa 3.2 esiintyvä relaatiotietokantojen mallintamiseen

tarkoitettu CWM (*Common Warehouse Metamodel*) on myös MOF:n mukainen metamalli.

Edellä kuvatut mallityypit voidaan sijoittaa neljälle metatasolle (taulukko 3.1). MOF kuuluu ylimmälle eli M3-tasolle, metamallit sijaitsevat metatasolla M2, varsinaiset mallit tasolla M1 ja konkreettiset ilmentymät tasolla M0. Tasoa M4 ei tarvita, koska MOF kuvaa itse itsensä [Fra03].

Metataso	Kuvaus	Elementtiesimerkkejä
M3	MOF eli joukko käsitteitä, joilla voidaan määrittää metamalleja	MOF-luokka, MOF-attribuutti, MOF-assosiaatio
M2	Metamalli, instansseja MOF-käsitteistä	UML-luokka, UML-assosiaatio, CWM-taulu, CWM-sarake
M1	Mallit, instansseja M2-metamalleista	luokka ”Asiakas”, sarake ”Tilinumero”
M0	Oliot ja data eli instansseja M1-malleista	asiakas Matti Meikäläinen, tili 123456

Taulukko 3.1. MDA-metatasot [Fra03]

M3-taso mahdollistaa siis sen, että metamalli voidaan muuttaa toiseksi metamalliksi. Esimerkiksi UML-metamallin kohdalla tämä on toteutettu siten, että UML-kaavioissa oleva informaatio tallennetaan MOF:a noudattavaan XMI-muotoon (*XML Metadata Interchange*) [Fra03]. Yksi käytännön implikaatio tästä on, että sama XMI-esitys voidaan näin avata toisessa UML-editorissa. Vielä merkittävämpi asia on kuitenkin se, että XMI mahdollistaa UML-mallien linkittämisen toteutusteknologioihin [Fra03].

Vaikka UML ei ole ainoa mahdollinen metamalli, on se kuitenkin niistä selvästi tunnetuin [Béz05]. On siis syytä hieman tutustua tarkemmin siihen, kuinka UML-kaavioista voidaan todella luoda suorituskelpoista lähdekoodia. Tämä saattaa tuntua erikoiselta, kun ajattelee esimerkiksi UML-luokkakaaviota. Kuinka muunnostyökalut voivat ymmärtää, että tietyistä luokista pitää generoida tietynlaista koodia?

Vastaus edelliseen ovat UML-profiilit, joiden avulla UML:ää pystytään laajentamaan ja tekemään siitä kullekin sovellusalueelle sopivaa [Fra03]. UML-profiilit tarkoittavat sitä, että määritellään joukko stereotyyppejä ja lisätietomääreitä (*tagged values*), joita

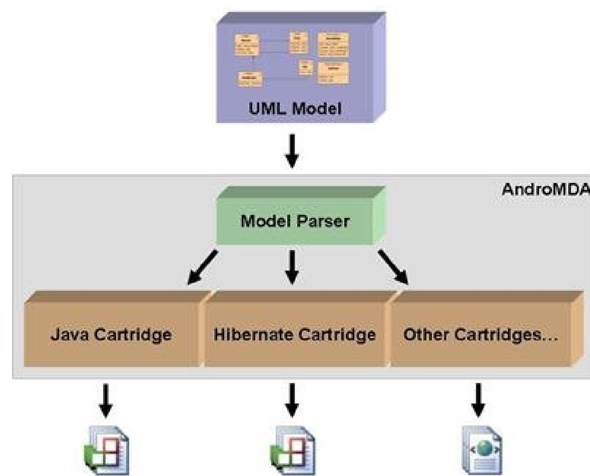
voidaan liittää UML-elementteihin. Esimerkiksi luokkaan voidaan liittää stereotyyppi *Entity*, jolloin muunnostyökalu osaa luoda kyseisestä luokasta tietynlaiset rajapinnat ja toteutukset ja käsitellä luokan attribuutteja tietyllä tavalla. Tämä ominaisuus tekee UML:stä mallinnuskielen, jota voidaan käyttää DSML:nä (ks. luku 2.2).

Vaikka MDA tarjoaa selkeän ajattelumallin siitä, kuinka ohjelmistoja voidaan rakentaa malliperustaisesti, ei se kuitenkaan itsessään ole vielä riittävä käytännön tasolle mentäessä. Toki teoriassa MDA:n periaatteiden mukaisesti kuka tahansa voisi rakentaa esimerkiksi omat mallimuunnostyökalut ja UML-profiilit, mutta käytännössä tämä ei olisi järkevää. Sen sijaan markkinoilla on useita erilaisia MDA-työkaluja, joita voidaan hyödyntää malliperustaisessa ohjelmistokehityksessä. Seuraavassa luvussa tutustutaan yhteen näistä.

4 AndroMDA – esimerkki MDA-työkalusta

Tässä luvussa tutustutaan AndroMDA-työkaluun. Tiedot perustuvat AndroMDA:n www-sivuihin [And09], ellei muuta ole mainittu.

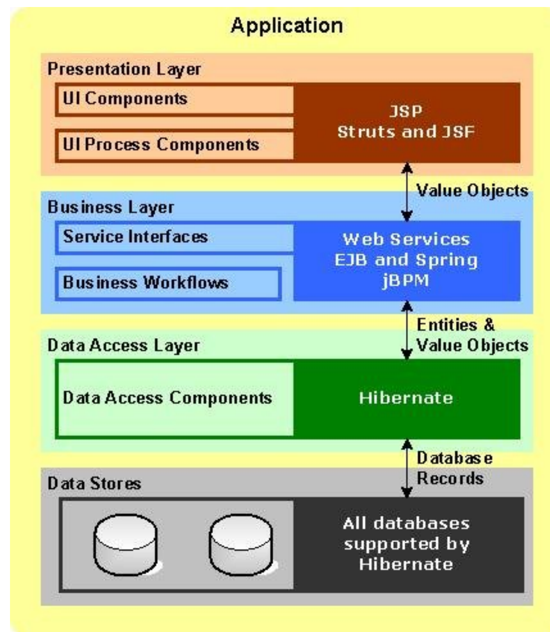
AndroMDA on avoimen lähdekoodin MDA-muunnostyökalu. Sen viimeisin versio 3.3 on julkaistu huhtikuussa 2008. Sivustolla on lisäksi suunnitelmat versiosta 4.0, mutta kyseisen version kehitys on toistaiseksi keskeytetty [AnF09].



Kuva 4.1. AndroMDA – UML:stä sovelluskoodiksi [And09]

AndroMDA:n keskeinen ominaisuus ovat siihen valmiiksi rakennetut mallimuunnokset eräille tunnetuille alustoille erityisesti Java-ympäristössä (kuva 4.1). Näitä ovat EJB, Hibernate, Spring ja Struts. Lisäksi Javan ulkopuolelta tuetaan .NET-ympäristöä sekä yleisesti Web services -dokumentteja. Käyttäjät voivat myös luoda omia malli-muunnoksia. AndroMDA kutsuu mallimuunnoksia termillä *cartridge*.

Kuvassa 4.2 esitetään AndroMDA:n avulla generoitujen sovellusten arkkitehtuuri. Se noudattaa tyypillistä kerrosarkkitehtuurityyliä sisältäen generoidun koodin kaikille neljälle kerrokselle. Samasta mallista saadaan siis koodi kaikkiin näihin. Jokaiseen kerrokseen on valittu suosittuja sovelluskehyksiä, joskaan niiden uusimpia versioita ei tueta.



Kuva 4.2. AndroMDA:n generoimien sovellusten arkkitehtuuri [And09]

Koodin generointi perustuu niin sanottuihin template-tiedostoihin. Jokaista mahdollista kohdetiedostoa (esim. Java-luokka tai XML-konfiguraatiotiedosto) varten on yksi template, joka kertoo, kuinka kyseinen tiedosto kirjoitetaan. Kyse on siis kuvan 3.1 muunnosmääritelmistä. Templatien lisäksi generoinnissa käytetään Java-luokkia, joita kutsutaan nimellä *Metafacades*. Ne ovat façade-suunnittelumallin mukaisia luokkia, jotka eristävät metamallien toteutuksen.

AndroMDA:n konfigurointi tapahtuu yhden XML-tiedoston avulla. Siinä määritellään käytettävät mallimuunnokset sekä niiden asetukset. Esimerkiksi EJB:n voi halutessaan ottaa pois päältä kommentoimalla yhden rivin. Lisäksi jokaiseen muunnosmääritelmään liittyy omat asetustiedostot.

Metamallien kuvauskieleksi on valittu UML. Sitä täydennetään UML-profiilien mukaisesti stereotyypeillä ja lisätietomääreillä. Kun AndroMDA:n generointimoottori löytää tunnetun stereotyypin, se valitsee sopivan mallimuunnoksen, joka generoi tarvittavan koodin. Koodin generoinnissa käytetään oletusarvoisesti Apachen Velocity -työkalua.

Koodin generointiin liittyy oleellisesti tietotyyppien linkitys (*mapping*) PIM-tasolta PSM-tasolle. Nämä kuvataan omissa tiedostoissaan. Esimerkiksi yleiselle String-tietotyypille on Java-linkityksessä annettu tietotyyppi `java.lang.String`, .NET:ssä `System.String` ja MySQL:ssä `VARCHAR(255) BINARY`.

Tyypillisiä AndroMDA:lla mallinnettavia asioita ovat luokkien suhteet luokka-kaavioiden avulla sekä käyttötapausten kuvaaminen aktiviteettikaavioilla. Näillä kahdella kaaviotyypillä pystyy mallintamaan tavallisen J2EE-sovelluksen tietokannasta web-käyttöliittymään.

AndroMDA:n www-sivuilta löytyy myös esimerkkisovelluksia, joiden avulla pääsee hyvin näkemään, miten MDA-sovelluksen rakentaminen käytännössä tapahtuu. Tällaista käyttäessä on jopa yllättävää havaita, että malliparadigma todella toimii ja että koodin generointi on pystytty viemään hyvinkin pitkälle. Pieni osa koodista pitää kirjoittaa itse, mutta esimerkiksi käyttöliittymät rakentuvat templatien avulla, kunhan ne vain on oikein mallinnettu aktiviteettikaavioilla ja lisätietomääreillä.

5 Yhteenveto

Tässä seminaaripaperissa on käsitelty malliperustaisen ohjelmistokehityksen taustaa, keskeisiä periaatteita ja hyötyjä. Tätä on täsmennetty malliperustaisen arkkitehtuurin ominaisuuksilla sekä käytännön tasolla tutustumalla AndroMDA-nimiseen MDA-generointityökaluun.

Tärkein havainto on se, että malliperustainen ajattelu ohjelmistokehityksessä voi potentiaalisesti nostaa ohjelmistokehityksen tuottavuuden uudelle tasolle. Toisaalta seminaaripaperissa ei pohdittu ollenkaan, minkälaisia haasteita ja käytännön ongelmia malliparadigmaan liittyy ja kuinka realistisia sen onnistumismahdollisuudet ovat.

Teknisestä näkökulmasta ajateltuna oleellista malliperustaisessa arkkitehtuurissa on kaksi abstraktiotasoa ja niiden välillä tapahtuvat mallimuunnokset. Alusta-riippumattomalla tasolla (PIM) kuvataan sovellus mallien avulla niin, ettei toteutus-alustoja ole valittu. Mallimuunnosten avulla päästään alustariippuvaiselle tasolle (PSM), joka käytännössä tarkoittaa suorituskelpoista koodia.

Mallien kuvaus tapahtuu kohdealuekohtaisia mallinnuskieliä (DSML) käyttäen. Näistä tunnetuin on UML, jota laajennetaan UML-profiileilla kohdealueen tarpeiden mukaan. UML on esimerkki metamallista, jonka tarkoituksena on kuvata varsinaisissa malleissa käytettävä kieli. Malli, joka kuvaa metamallien kielen, on nimeltään metametamalli. Malliperustaisen arkkitehtuurin metametamalli on nimeltään MOF.

Lähteet

And09 AndroMDA www-sivut. http://www.andromda.org. [4.2.2009]

AnF09 AndroMDA Forum www-sivut. http://galaxy.andromda.org/forum. [4.2.2009]

BBJ07 Jean B´ezivin, Mikaël Barbero, Fréderic Jouault. On the Applicability Scope of Model Driven Engineering. In *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07), 2007*. IEEE Computer Society.

BCK98 Bass K., Clements P. ja Kazman R., *Software Architecture in Practice*. Addison Wesley Longman Inc. 1998.

BJB08 Barbero, M.; Jouault, F.; Bezivin, J. Engineering of Computer Based Systems. In *15th Annual IEEE International Conference and Workshop on the March 31 2008-April 4 2008,* pages 277-286. IEEE Computer Society.

Béz05 Jean Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171-188, May 2005. [Myös http://www.sciences.univ-nantes.fr/lina/atl/www/papers/OnTheUnificationPowerOfModels.pdf]

BJT05 Jean Bezivin, Frederic Jouault, and David Touzet. Principles, standards and tools for model engineering. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pages 28-29, Washington, DC, USA, 2005. IEEE Computer Society.

Fra03 David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press, 2003.

MDA03 *MDA Guide Version 1.0.1*. Editors: Joaquin Miller and Jishnu Mukerji. 2003. OMG. ftp://ftp.omg.org/pub/docs/omg/03-06-01.pdf.

OMG09 Object Management Group www-sivut. http://www.omg.org. [4.2.2009]

Sch06 Douglas C. Schmidt. Model-Driven Engineering. *Computer*, 39(2):25-31, February 2006. [Myös http://www.computer.org/portal/site/computer/menuitem.5d61c1d591162e4b0ef1bd108bcd45f3/index.jsp?&pName=computer_level1_article&TheCat=1005&path=computer/homepage/0206&file=gei.xml&xsl=article.xsl&;jsessionid=J2yQpWcYvRvLL6pbDwHpfZJHyp8FRxLKRpGGlG3QlZ27xFDLlB5b!-1582182879]

Sol00 Richard Soley and the OMG Staff Strategy Group. *Model Driven Architecture*. Object Management Group. White paper. November 2007. http://www.omg.org/docs/omg/00-11-05.pdf.

StC08 Ashley Sterritt and Vinny Cahill. Customisable Model Transformations based on Non-functional Requirements. In *2008 IEEE Congress on Services 2008 - Part I*, pages 329-336. [Myös http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04578344]