Modelēšanas rīki un programmatūras koda ģenerāciju (MDA)

Valdis Vītoliņš

Ievads

Sen zināma izpratnes plaisa starp biznesa cilvēkiem, kas nosaka prasības, un tehniskajiem cilvēkiem, kas izstrādā produktus. Konkurencei biznesā pieaugot, jaunas sistēmas nepieciešams izstrādāt ātri - līdz ar biznesa maiņu (apvienošanās un sadalīšanās, jaunu tirgu atvēršana, u.tml.). Ja produkta izstrādātāji neizprot, kā biznesa prasības pārvērst programmā, izmaiņas prasītajos termiņos un kvalitātē realizēt nav iespējams.

No otras puses, programmatūra paliek arvien sarežģītāka, tāpēc tās izstrādē jāveido arvien vairāk "pārvaldības slāņi". Ja skaitliskos aprēķinus varēja izstrādāt mašīnkodos, tad sarežģītu lietojumprogrammu bez objektorientētas pieejas izstrādāt nevar. Pielietojot programmatūru arvien sarežģītāku biznesa uzdevumu risināšanā, arī objektorientētā pieeja nav pietiekama efektīvai programmu izstrādes pārvaldībai. Lai tiktu galā ar sarežģītu biznesa sistēmu, kas ne tikai pasīvi saglabā lietotāju ievadītos datus, bet pati aktīvi vada biznesa procesu, nepieciešams jauns abstrakcijas līmenis.

Pārvaldot programmatūras izstrādi augstākā abstrakcijas līmenī, no vienas puses iespējams vienkārši pārvaldīt sarežģītu sistēmu izstrādi, bet no otras puses, šāds abstrakcijas līmenis arvien vairāk pietuvinās biznesa cilvēku izpartnei par to, kā sistēmai jāstrādā.

Ja agrāk biznesa modelēšana un programmu dizaina izstrāde bija atdalītas darbības, tad tagad tās ir iespējams apvienot. OMG [1] modeļa vadāma arhitektūra (model driven architecture – MDA [2]) ir paņēmiens, kā pārskatāmā veidā pārvaldīt heterogēnu un izkliedētu sistēmu izstrādi, pieņemot, ka prasību nemitīga maiņa ir neatņemama izstrādes procesa īpašība. Tas nozīmē, ka iespējams veidot programmas augsta līmeņa dizainu, jau prasību analīzes laikā. Un, ja prasības mainās, arī programmas dizainu var izmainīt uzreiz, mainot prasības.

Šobrīd vairumam šķiet pieņemami, ka programma netiek veidota konkrētai aparatūrai. Programmas tiek rakstītas augsta līmeņa programmēšanas valodās, un tādas lietas kā procesora un atmiņas pārvaldība tiek atstāta kompilatora/interpretatora pārziņā. Programmas tiek rakstītas lielākoties procedurālās (C, C++, Java) programmēšanas valodās, kurās tiek norādīts, kas kādā kārtībā ir jāizdara, lai iegūtu vajadzīgo rezultātu.

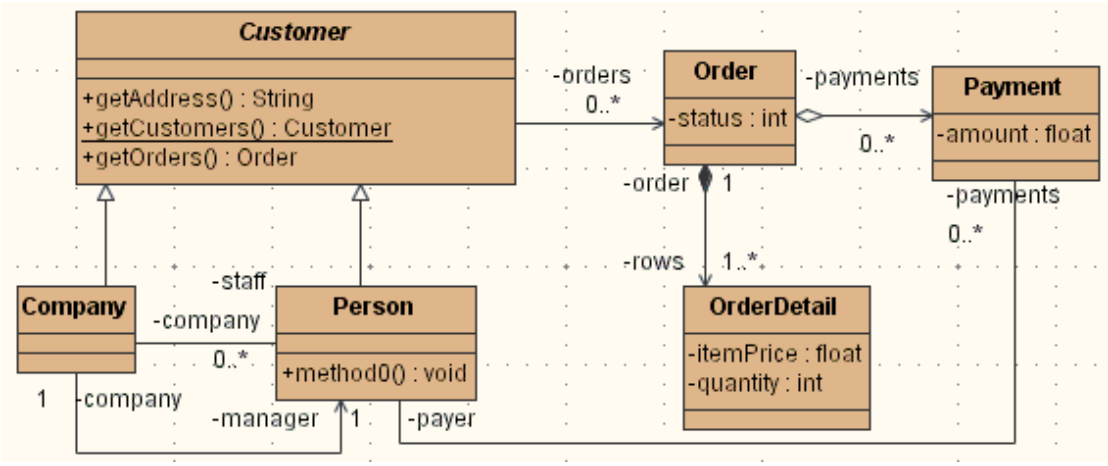
MDA pamatideja ir tāda, ka programma tiek rakstīta ne tikai neiedziļinoties aparatūrā, bet neinteresējoties pat par to, kādā programmēšanas valodā un ar kādiem paņēmieniem tā tiek implementēta. Programma tiek rakstīta deklaratīvā valodā, kas pasaka, kam ir jābūt, bet nenorāda to, kā to var izdarīt. Programmai veido platformas neatkarīgu modeli (PIM - platform independent model). Konkrēta algoritma implementācija un platformai specifiskais modelis (PSM - platform specific model), paliek kompilatora/interpretatora pārziņā.

Šāda pieeja ļauj veidot gatavus biznesa orientētus risinājumus (patterns), kurus var implementēt dažādās vidēs. Tas uzlabo risinājumu atkalizmantošanas iespējas.

Praktisks eksperiments

MDA prakse pagaidām vēl ir tālu no teorijas. Šobrīd no modeļa vēl nav iespējams izveidot pilnvērtīgu darbojošos programmu. Tomēr jau tagad ir iespējams izmantot dažas MDA idejas, piemēram, izveidot statisku programmatūras dizainu (klašu diagrammu), un no tās ģenerēt koda sagataves kā arī dažu rutīnas metožu (piem., get.. un set..) implementācijas.

Modelēšanas rīku darbības analīzei tika izmantots vienkāršs bet loģiski “bagāts” modelis - klašu diagramma, kas nosaka programmas statisko dizainu. Modelī ir abstraktas un vienkāršas klases, ar atribūtiem un metodēm (ieskaitot instanču un klašu metodes), vienā un abos virzienos navigējamas relācijas, ģeneralizācija, agregācija un dekompozīcija.



UML klašu diagramma koda izstrādes pārbaudei

Kā programmēšanas valoda tika izmantota Java, un kā izstrādes vide – Eclipse [3] 2.1, vai 3.0. Modelēšanas rīki tika ņemti no [4], uzmanību pievēršot tikai plaši pazīstamiem komerciāliem, vai arī bezmaksas un/vai atklātā koda rīkiem.

Koda sinhronizācija ar modeli (klašu diagrammu) iespējams veikt divos veidos. Vienkāršāk ir pa tiešo analizēt programmatūras kodu, bet šāda metode nestrādā lielos projektos ar sarežģītu katalogu (pakotņu) struktūru. Sarežģītāk, bet vispārīgāk to var veikt pastarpināti ar XMI failu. Tādā gadījumā programmatūras izstrādes vide sinhronizē kodu ar XMI failu, un tad XMI fails tiek importēts/eksportēts modelēšanas redaktorā. XMI pārvaldībai Eclipse vajadzīgs EMF [5] spraudnis (plugin). Diemžēl, dažādi redaktori izmanto dažādas XMI versijas (1.1, 1.2, 2.0). un XMI tiek aprakstīta izmantojot dažādas metadatu shēmas. Tāpēc praktiski jaukt dažādus redaktorus vienā projektā nav iespējams.

Redaktoru apraksts

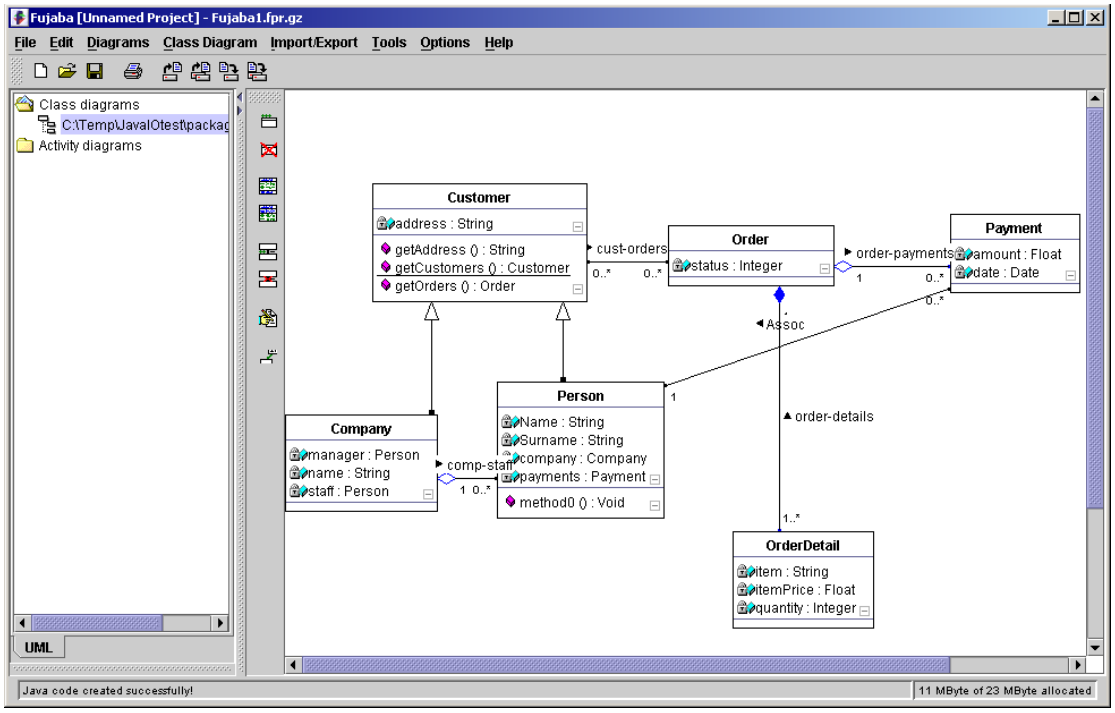
Rational Rose tagad ir vēl viens no IBM apakšuzņemumiem. Izplatītākais modelēšanas rīks, ar bagātām koda inženierēšanas iespējām. Vienkāršā variantā, datu apmaiņa notiek ar XMI failu. Ir pieejami dažādi papildus moduļi, kas padara caurspīdīgu koda pārvaldību, moduļu izstrādi web lietojumu serveriem, datu bāzēm u.tml. Ir tikai komerciālā versija.

Nomagic MagicDraw 7.x ir plašas modelēšanas iespējas. Šis redaktors tiek uzskatīts par vadošo koda pārvaldības jomā. Integrācija ar Eclipse notiek importējot/eksportējot XMI failu. XMI failā netiek iekļauta diagrammu izklājums (layout), tāpēc reversi inženierētajam vai sinhronizētajam kodam ir papildus jāpievieno klašu diagramma, kurā izklājums jāsakārto ar roku. Kā komerciāls rīks tas ir dārgs.

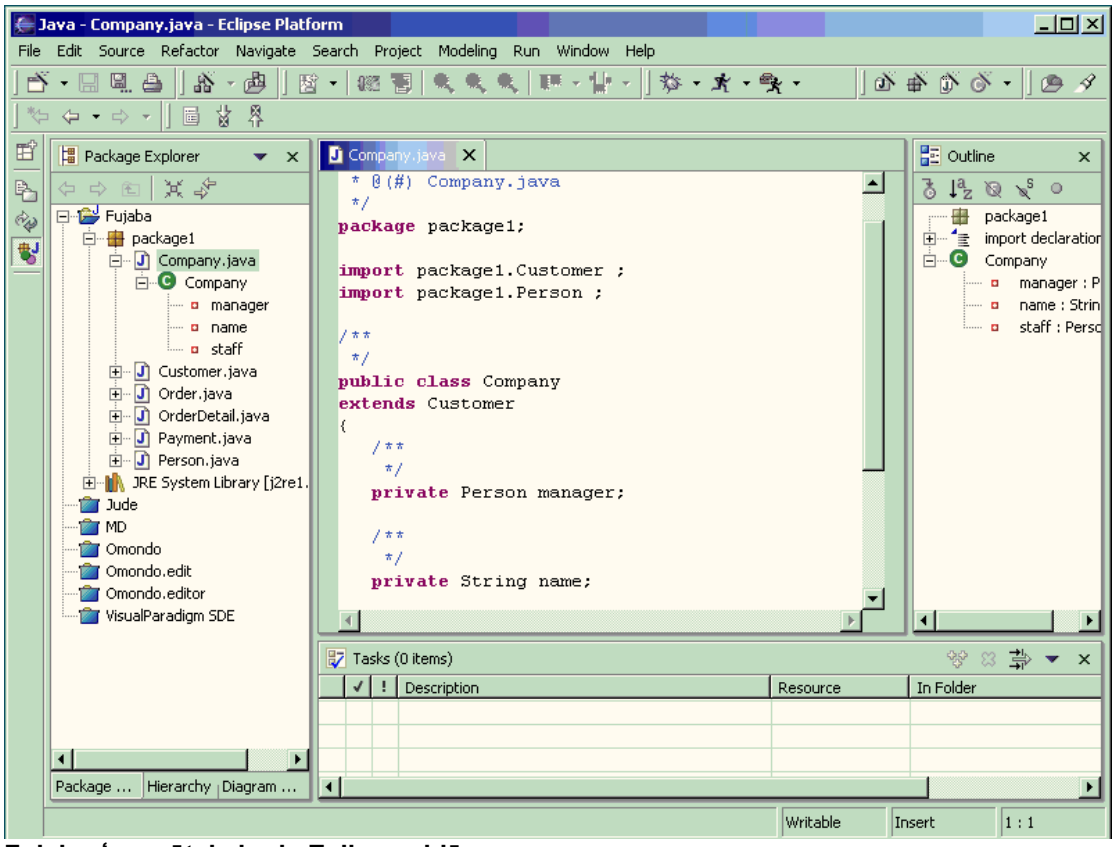
Poseidon for UML 2.5 no Gentleware AG. Rīks ir attīstīts no atklātā koda projekta ArgoUML. Modelēšanas iespējas ir salīdzināmas ar MagicDraw. Community versija ir bez maksas bet tajā ir tika modelēšanas redaktors. Automātiskas koda ģenerēšanas/sinhronizācijas iespējas ir komerciālajā versijā, kas ir padārga.

Jude no Eiwa System Management, [6] ir bezmaksas, bet nav atklātais kods. Strādā Javā un tai ir labas modelēšanas iespējas. Sinhronizācija notiek pa tiešo, izmantojot kodu. XMI usturēšana ir izstrādes stadijā.

Fujaba [7] ir atklātā koda redaktors. Bez klašu diagrammām var zīmēt arī stāvokļu un aktivitāšu diagrammas. Koda pārvaldība notiek pa tiešo, izmantojot kodu. Dokumentācija vācu valodā.



Atklāto koda redaktors Fujaba

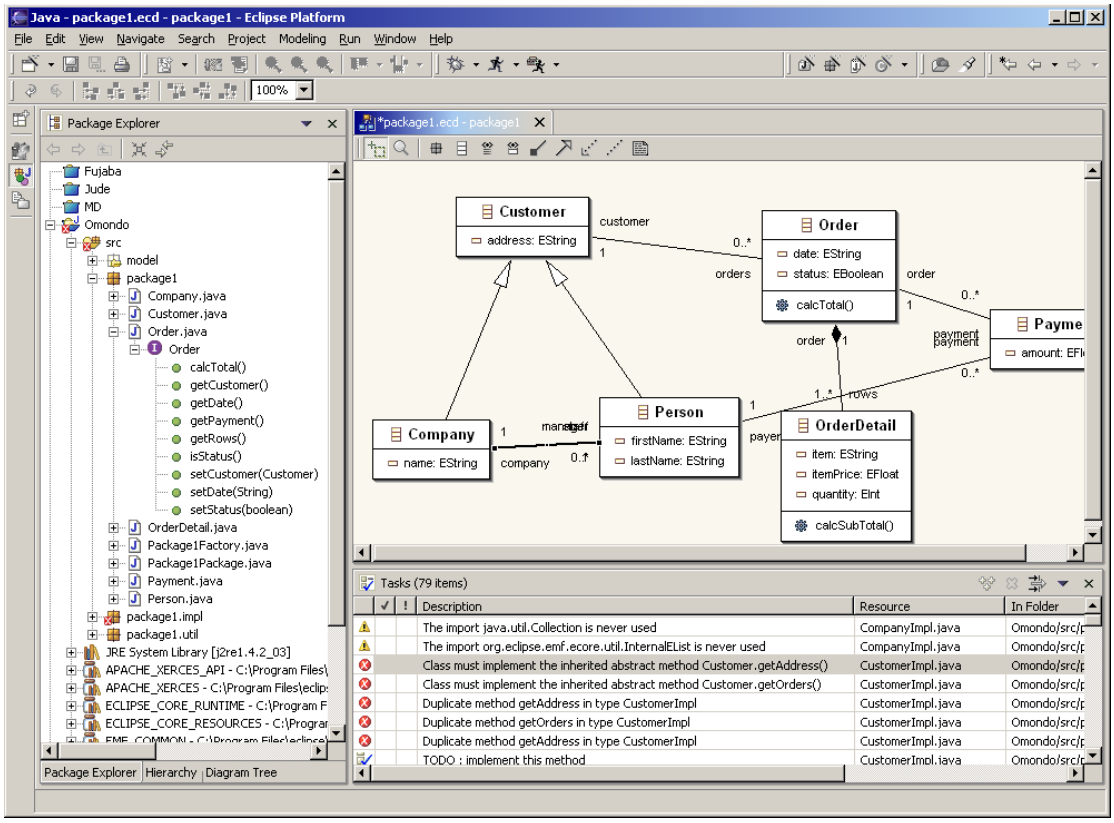


Fujaba ģenerētais kods Eclipse vidē

SDE [8] ir bezmaksas un komerciālās versijas. Tas strādā kā Eclipse spraudnis un nav atkarīgs no citiem spraudņiem. Bezmaksas versijā ir tikai modelēšanas redaktors, kas ir diezgan vājš. Koda pārvaldība ir par maksu. Tajā ietilps koda ģenerēšana un sinhronizācija caur XMI failu, ir pieejami moduļi arī datu bāzes shēmas ģenerēšanai un web lietojumu serveriem.

Omondo [9] arī strādā kā Eclipse spraudnis, bet tas ir atkarīgs no diviem citiem Eclipse spraudņiem. Abi vajadzīgie spraudņi ir atklātā koda projekti. Klašu repozitorija pārvaldībai

(XMI failam) Omondo izmanto EMF spraudni, un kā grafisko bibliotēku - GEF [10] spraudni. Koda pārvaldībai pietiek ar bezmaksas versiju, par maksu nāk klāt spraudņi web lietojumu serveriem un CVS sistēmai.



Pilnībā integrēta MDA izstrādes vide Eclipse ar Omondo

Rezultāti

Ļaujot kodu veidot ģeneratoram, kods ir tik labs, cik labs ir ģenerators. Pat vienkāršā piemērā pareizs kods brīžiem ģenerējās ar grūtībām. Tomēr jāteic, ka grūtības sagādāja tikai papildus iespēju izmantošana, piemēram Java koda bagātināšana ar papildus klasēm un metodēm, kas labāk implementē klašu diagrammas nianses.

Atklātā koda ideja uzņem apgriezienus un jau šobrīd ir iespējams izmantot dažas MDA idejas, tikai ar bezmaksas vai pat atklātā koda rīkiem. Piemēram, izmantojot Eclipse un Omondo bezmaksas versiju. Jāmaksā tikai tad, ja grib ērtības, vai arī papildus iespējas. Pimērām, web lietojumu serveru konteineri (EJB) koda pārvaldība CVS sistēmā, datu bāzes shēmas veidošana.

Viena no svarīgām iespējām ir semantiski pareiza koda ģenerēšana. Klašu diagramma ir bagātāka par Java. Tajā var atšķirt atribūtu no relācijas, relācijai iespējams norādīt virzienu, atzīmēt agregāciju (sastāvdaļu instances netiek iznīcinātas kopā ar galveno objektu) vai dekompozīciju (iznīcinot galveno objektu, tiek iznīcinātas tā sastāvdaļas). Ģenerējot vienkāršu kodu ar klasēm aun atribūtiem, visas šīs nianses tiek pazaudētas. Kā likums, komerciālie rīki veido papildus kodu, kas ļauj šīs nianses saglabāt un pēc koda reversās inženierēšanas tiek iegūta tāda pati diagramma.

Otra svarīgākā iespēja ir persistento klašu (objekti saglabājas pēc programmas darbības) savietošana ar datu bāzi. Tāda iespēja šobrīd ir tikai komerciālajiem rīkiem (Rose, MagicDraw). Tiem ir moduļi, kas ģenerē ne tikai programmas kodu, bet arī datu bāzes shēmu. Kā bezmaksas risinājumu var izmantot Hibernate [11] projektu, bet tas piedāvā tikai standarta metodes un koda ģenerators ir jāizstrādā pašam.

Tā kā naviespējams visu apskatīt vienā rakstā, papildus informāciju var skatīt ari sekojošajās saitēs [12,13].

Kad tas atmaksājas?

- Lielos projektos, kur paralēli strādā daudz izstrādātāju ar dažādām zināšanām un prasmēm.
- Ja labs dizains ir svarīgāks par viduvēju implementāciju (raksturīgs lielām sistēmām).

- Ja grib atdalīt programmatūras dizaina izstrādi no kodēšanas.
- Ja ir jāveido līdzīgi, bet tomēr dažādi risinājumi (biznesa šablonu izmantošana).
- Ja laika jāizstrādā programma ar vēl neskaidrām prasībām (parasts gadījums, strādājot ar iterācijām).

Kad tas neatmaksājas?

- Ja nepieciešams veidot "viltīgu kodu" (raksturīga sistēmām ar kritisku veiktspēju).
- Ja nepieciešams tieša resursu pārvaldība (raksturīga iegultām sistēmām).
- Sistēmām ar procedurālu dizainu (jo MDA izmanto objektorientēto pieeju).

Kas tālāk?

Kā rāda vēsture, izgudrojot jaunu paņēmienu, vecie nepazūd pilnībā, lai arī jaunais aizņem lielāko nišu. MDA praktiskā realizācija šobrīd aprobežojas tikai ar sistēmas statiskās struktūras .modelēšanu, tāpēc paredzams, ka MDA ieņems arvien nozīmīgāku lomu.
Nākotnē tiks attīstītas sekojošas lietas:

- Lai no modeļa būtu iespējams ģenerēt pilnvērtīgu programmu, vairāk tiks izmantotas formālas deklaratīvās valodas, piemēram OCL. Tiks izstrādāti interpretatori/kompilatori, kas pilnīgu modeli pārveidos izpildāmā kodā.
- Biznesa sistēmās paplašināsies dinamisko modeļu izmantošana. Vairāk tiks izmantotas stāvokļu, aktivitāšu un sadarbības diagrammas un dzinēji, kas tās prot izpildīt. Tad biznesa process pilnībā noteiks sistēmas darbību un ar "Programmēšanu" nodarbosies biznesa analītiķi.
- Tiks atvieglota pāreja no vienas izstrādes vides uz citu - mainīt programmēšanas valodu, darbības vidi. Jau tagad izstrādā transformāciju valodas un dzinējus, kas tās pratīs veikt.

Saites

1 http://www.omg.org/
2 http://www.omg.org/mda/
3 http://www.eclipse.org/
4 http://www.objectsbydesign.com/tools/umltools_byCompany.html
5 http://www.eclipse.org/emf/
6 http://objectclub.esm.co.jp/Jude/jude-e.html
7 http://www.fujaba.de/
8 http://www.visual-paradigm.com/vpuml.php
9 http://www.omondo.com/
10 http://www.eclipse.org/gef/
11 http://www.hibernate.org/
12 http://www-106.ibm.com/developerworks/opensource/library/os-ecemf1/
13 http://all_about_uml.tripod.com/uml_tools.htm