Alkalmazott Matematikai Lapok **26** (2009), 1-7.

MODELLVEZÉRELT SZOFTVEREK KÉSZÍTÉSE I.

KILIÁN IMRE

A „modellvezérelt szoftverkészítés" (Model Driven Architecture, MDA) kifejezést az OMG használta a hasonló című projektjében [3]. A szóban forgó projekt nevében azonban a modellvezéreltség, mint jelző inkább a „szoftverkészítésre" vonatkozik. Az OMG elképzelésének kiindulási pontja szerint ugyanis a szoftvert a gépi és egyéb környezeti viszonyoktól teljesen elvonatkoztatva, független módon tervezzük meg, és a különböző szoftver és hardver környezeteket leíró metamodellek és átalakítási szabályok segítségével környezetfüggő modellé alakítjuk.

A jelen írás egy másik, de az előzőtől nem független és nem is idegen megközelítést alkalmaz. A modellvezérelt jelző nem a szoftverkészítésre vonatkozik, hanem az elkészített szoftverre magára. A futó szoftver az, amely magát a vizsgált célterület modelljét is tartalmazza és kezeli, és a tárolt modell alapján bizonyos egyöntetűen megvalósítható műveleteket végez.

1. Kétszintű szoftverek

A „modellvezéreltség" mellett alkalmazható másik jelző a „kétszintűség", ami szintén jól magyarázza a megoldást. Hagyományosan készített szoftverek ugyanis „egyszintűek". A szoftver maga testesíti meg a modellt, hiszen annak átalakított, „kódgenerált" változata, amihez a fejlesztő még hozzákapcsolja az egyedi részeket, felhasználó felületet, egyes algoritmusok egyedileg megírt részeit és másokat. A szoftver maga az általa által kezelt, átalakított és a számítógép tárában futásidőben létrejövő adatok metaszintjének tekinthető.

A „meta-" előtag görög eredetű, jelentése valami felett álló, valamin túl fekvő (pl. metagalaxis, metafizika). Modellezési értelemben a „meta-" valamilyen adattömeg leírását, kezelését, ill. a felette értelmezett műveletek szintjét jelenti, ezért egy szoftver mindig az általa kezelt adatok metaszintje. Az egyszintű szoftverek bonyolultsága részben a szoftver modelljébe van betervezve, másrészt viszont a bonyolultság az osztályfüggvények megvalósításába van bekódolva, amit a szoftver terv már nem tartalmaz.

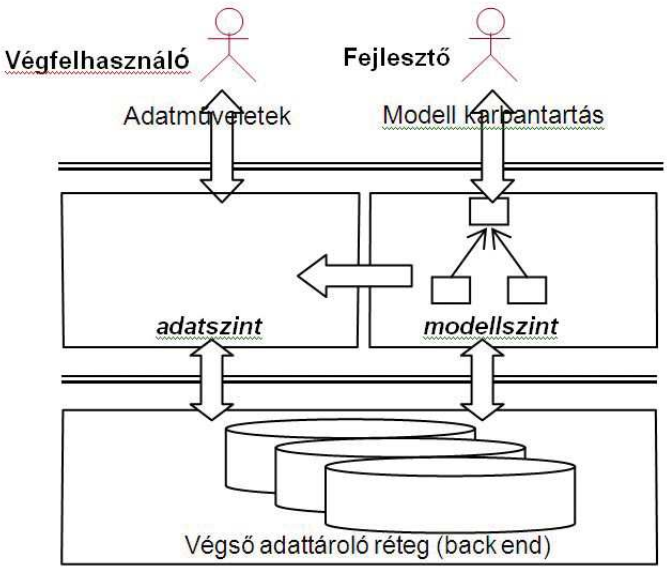
A hagyományosan készített egyszintű szoftverek olyan korlátokat állítanak a saját fejlődésük elé, melyek kétszintű felépítménnyel feloldhatók:

1. Nehéz változtathatóság: a modellszint rögzített, a legkisebb újabb bővítési elképzelés esetén is modellbővítés, a megvalósítás nyelvén újabb programcsomagok megírása, lefordítása és szerkesztése szükséges.

2. A modellek programnyelvekbe történő leképezésének használatos megoldásai esetenként túlságosan nehézsúlyúak, de esetenként korlátokat is állítanak (pl. objektum orientált nyelvek esetében a többszörös öröklődés tiltásával). Ennek következményeképpen már a tervezés folyamán ügyelni kell egyes megvalósítási részletekre is, pl. ha a tervből a programkódba átvivő gépi leképezés nem szolgáltat elegendően „pihesúlyú" programobjektumokat, akkor már a modellben sem építhető fel a probléma igényeinek megfelelően aprólékos osztályszerkezet. Vagyis már a tervezés során is figyelembe kell vennünk egyes, a feladat belső lényegén és összefüggésein túlmutató szempontokat.
3. Egyedileg meg kell valósítani minden olyan eljárást, amelyek az egyes egymással is összefüggő modellosztályok elemeire vonatkozóan adatokat gyűjtenek össze, ill. alakítanak át, noha ezek egyöntetű, magasszintű adatelérő, ill. -átalakító nyelven történő megfogalmazása kézenfekvőbb és egyszerűbb lehetne.
4. Ha mégis a modellszint tárolásához hasonló adatszerkezetek ábrázolása szükséges a szoftverben, akkor az is egyedi megoldással történik, ami egy egységes megoldáshoz képest lényegesen megnövelheti a fejlesztés befektetésigényét, és növeli a hibakockázatot.

A kétszintű vagy modellvezérelt szoftverek (1. ábra) mindezen hátrányok kiküszöbölésére törekszenek. Ezekben a program természetesen ugyanannyira kötött, mint más szoftverekben. A szoftver által kezelt adatok azonban két részre különíthetők el. Az egyik a modellszint vagy metaszint, a másik pedig a tényleges adatok szintje. Ezekre a következők a jellemzők:

- A modellszinten tároljuk a világ vizsgált részének modelljét, és ez vezérli az adatszintet. Ehhez a modellhez tartozhatnak egyedileg beprogramozott függvények (a modellek osztályfüggvényei vagy módszerei) is, amelyek az adatszintről önműködően hívódnak meg, de erre a képességre nincs mindig szükség.
- Az adatszinten vagy példányszinten folynak azok az üzleti modellnek megfelelő adat-átalakítások, amelyek a háttértár rétegét a megjelenítés rétegével összekapcsolják. Ezek a műveletek nincsenek közvetlenül beprogramozva, hanem csak néhány tipikus és általános művelet van megvalósítva. Az adatszint nyitott annyiban, hogy a modellszintről vezérelve objektumpéldányokat hozhatunk létre, és ugyanígy végrehajthatjuk a rajtuk definiált műveleteket, ill. a modellszinten beprogramozott függvényeket (dinamikus típuslekötés és hívhatóság).
- Kétszintű programfelépítmény létrehozása során technikai jellegű dolog ugyan, de igen fontos említést tenni a programfutást túlélő (perzisztens) objektumok tárolását megvalósító végső adattároló (back end) rétegről is. Itt tároljuk az adatszint objektumait, de a modell-leíró elemeket is.



1. ábra. Kétszintű szoftverfelépítmény

A rendszerhez a végfelhasználó az adatszinten keresztül kapcsolódik. Itt modellvezérelt általános célú eszközök állnak rendelkezésre, amelyek megjelenítő felületen keresztül működtethetők.

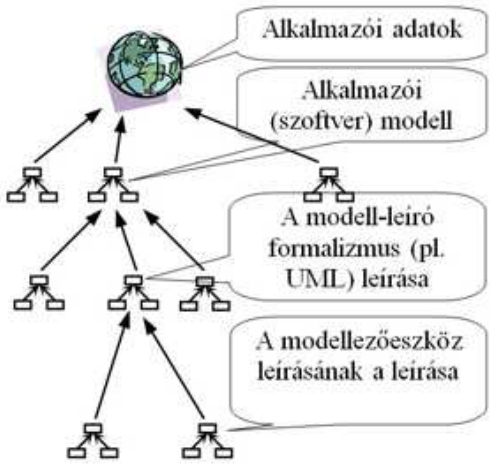
A modellszinten a modellező/fejlesztő elsődlegesen egy modellépítő és -karbantartó eszközön keresztül kapcsolódik a rendszerhez. Ez az eszköz lehet grafikus jellegű, de mindenképpen lehetővé kell tennie a modellek betöltését, mentését, és osztályfüggvények (módszerek) létrehozását. Vagyis egy olyan integrált fejlesztői környezetre van szükség, amelyik programkód szerkesztésére, majd dinamikus fordításra-betöltésre is alkalmas.

Az adat és a modellszint ilyen kettéválasztásának a legfőbb haszna, hogy az alkalmazás létrehozása során elsődlegesen a célalkalmazás logikai/formális modelljére összpontosíthatunk, a programozási részletkérdésekkel nem kell törődnünk. Az adatszint maga is adhat eszközöket egyéb alkalmazás- és környezetfüggő dolgok (pl. felhasználói felület) automatikus vagy félautomatikus elkészítésére, de mindez a modellszinttel semmilyen módon nem keverhető össze.

Modellvezérelt rendszerek esetében tehát a célszoftverek adatai, vagy azok egy része nem rendelkeznek kötött modell-leírással, hanem a modell maga is a program adatainak része, amely a többi adat szerkezetét írja le. A lehetséges modellek leírására alkalmas a metamodelljük, amely immár lehet kötött is, és amely igen erősen meghatározhatja a ténylegesen felismert és értelmezett modellek, valamint az azok alapján feldolgozható adatok és adatműveletek körét.

2. Az OMG négyrétegű metamodell szerkezete

Az Object Management Group egy objektum orientált technológiai szabványokat kidolgozó nyereségmentes alapítvány, amelynek munkatársai többek között az UML tervezőnyelvet is létrehozták. Tevékenységük legvégső modelljeként a négyrétegű metamodell szerkezetet állítják (2. ábra). Ez az „alkalmazói adatok/alkalmazói modell/ metamodell/meta-metamodell négyes", amit az ábrán látható fával jellemezhetünk. A fa csomópontjai között a „példánya" viszony áll fenn, vagyis az alkalmazói adattömeg példánya az alkalmazói modellnek, ez utóbbi modell viszont példánya a metamodellnek, ill. a metamodell példánya a meta-metamodellnek. Általánosságban a faszerkezet n-edik szintjén levő modellek példányai az $n+1$-edik szintről feléjük mutató modelleknek. A faszerkezet is jelzi, hogy a modellalkotási folyamattól függően egy adott példány-adatkészlethez többféle modell is létrehozható. A fa magassága elvben nem korlátos, vagyis a modellalkotás folyamatát elvben bármilyen sokszor ismételhetjük, a gyakorlatban azonban a meta-meta szintnél távolabbi elvonatkoztatásnak nincs jelentősége. Tekintsük ennek a fának a szintjeit:



2. ábra. Az UML négyrétegű metamodell szerkezete

- Az alkalmazói adatok szintjén a szoftver által kezelt konkrét adatokat, személyeket, számlákat stb. értjük.
- Az alkalmazói adatok példányai az alkalmazói szoftver modellben leírt osztályoknak. Ezt a szoftver tervezője hozza létre, ebből generáljuk ki a futó szoftver programkódját is. A konkrét adatok részben a számítógép tárában, részben háttértáron jönnek létre. A programkód maga a modell egy másik megjelenési formája, ami tartalmazza a később példányfüggően

létrejövő adatelemeket, valamint a példányfüggetlen eljárások kódját.

- Az OMG elképzelés harmadik szintjén az alkalmazott tervezési nyelv, esetünkben az UML metamodellje áll. Ez a nyelv által biztosított modellelemeket és azok összefüggéseit írja le, (esetünkben az osztály, tulajdonság, kapcsolat stb. fogalmakat). Ha a kétszintű rendszerünkben a modellszinten rögzített metamodellt kívánunk használni, akkor a legcélszerűbb a metamodellt kiegészíteni, esetleg abból levezetni a saját modellelemeinket, majd azt megvalósítani.
- A legmagasabb elvonatkoztatási szinten a meta-metamodell áll. Ezt akkor használatos, ha valamilyen modellvezérelt szoftverben még a metamodell sem rögzített. Vagyis olyan esetben, ha a szoftvert „háromszintűként" használva, különböző metamodellű rendszerekre kívánjuk használni, ill. esetleg, ha az egyetlen metamodell az idők során változna, fejlődne.

3. A modellszint műveletei

Modellvezérelt szoftverek kritikus pontja a modellszint megvalósítása. Ezen a szinten ugyanis a modellkarbantartó műveleteket kell megvalósítani, amelyeket csak a fejlesztést végző szakemberek működtethetnek.

A modellkarbantartás tipikus műveletei:

- Modell mentés-betöltés, amelynek során valamiféle szöveges jellegű modellleíró nyelv és a modelltárház közötti adatcsere történik. Ez az adatcsere maga is lehet modellvezérelt abban az esetben, ha a modelltárház maga is modellvezérelt. Ilyenkor a harmadik szoftver szinten található metamodell a szöveges modell-leíró nyelv nyelvtanát is tartalmazza. A modellszinten ezek után csak a szöveg-generáló és a szöveg felismerő műveletek futnak, amelyek a metamodellhez kapcsolt nyelvtanleírás alapján dolgoznak.
- Modelleken végzett inkrementális szemléletű szerkesztési műveletek, amelyek során a modell egyes elemein kisebb mérvű módosításokat végzünk.
- Modellellenőrzés, amely kétféle felfogásban működhet. A modell statikus szemantikus ellenőrzése alatt a modellre vonatkozó, a metamodellben rögzíthető megszorítások ellenőrzését értjük (pl. hogy nem hivatkozhatunk nem definiált modellelemre). Ez a lépés viszonylag kézenfekvő műveleteket tartalmaz, elsősorban formai vonásokat ellenőriz, és könnyen elvégezhető. A modell teljes ellenőrzése alatt a modellelemek formális logikai átalakítását és a létrejövő logikai állításrendszer ellentmondásmentességének vizsgálatát értjük. Megjegyezzük, hogy ez a feladat matematikai logikai értelemben nem eldönthető, vagyis nem készíthető olyan szoftver, ami egy modell tartalmi ellentmondásait teljes körűen fel tudná deríteni. Biztató részeredményről számol mégis be a [8] írás.

Hasonló modelltárházat valósított meg a SILK projektum [6], amelynek végső célja azonban nem a modellvezérelt szoftverek technológiájának kifejlesztése volt, hanem különböző modellű rendszerek közötti intelligens adat integráció megvalósítása. A munka a lezárása után azonban további kutatási tevékenységek kiindulópontjaként is szolgált.

4. Modellvezérelt alkalmazások

Modellvezérelt alkalmazások alatt az adatszinten megvalósítható műveletek bizonyos köreit értjük. A következő szakaszban ilyen lehetőségeket villantunk fel.

Adatműveletek

Modellvezérelt adatelérő és -módosító eszköz az ODMG OQL lekérdező nyelve [4]. Ez a relációs adatbázisok SQL kezelőnyelvének egyfajta kiterjesztése objektum-orientált adatbázisokra. A modellvezérelt adatkezelés lényege, hogy csak az objektum-orientált és a nyelvi alapműveletek vannak rögzítetten megvalósítva. Ezen túl minden más művelet modellvezérelt, még a számtani vagy logikai alapműveleteket is a modellben megadott műveletmegvalósítások hajtják végre.

Megjegyezzük, hogy az OQL nyelv logikai ereje igen távolra mutató, a teljes megvalósításra azonban nincs mindig szükség. Adott célra a teljes nyelvnél szerényebb követelmények és megoldások, vagyis csupán az OQL egy részhalmazának megvalósítása is elképzelhető.

Szövegszerű mentés-betöltés

Kézenfekvő művelet a rendszer adatállományának szövegszerű mentését és betöltését megvalósítani. Egy ilyen műveletkör a megvalósítást tekintve megegyezik a modellekre vonatkozó modellvezérelt mentés-betöltés művelettel is, azzal a különbséggel, hogy itt minden egy metaszinttel lejjebb szállt. Most nem a metamodell tartalmazza a modellre vonatkozó szintaktikus leírást, hanem a modell tartalmazza az adatokra vonatkozó hasonló leírást. Megjegyezzük, hogy adattartalom szövegszerű mentésére és betöltésére az OMG is adott egy paraméterezhető szabványjavaslatot [5], ami azonban a viszonylag merev paraméterezési konvenció miatt csak része a jelen tanulmányban javasoltnak.

Modellvezérelt modelltárház

Ha a modelltárház maga is modellvezérelt, akkor a tárház magát a modellt tekinti az adatszintjének, vagyis a modellszinten a metamodellt találjuk. Egy ilyen szoftverben nem a metamodell rögzített, hanem annak metaszintje, a metametamodell. Ennek a megközelítésnek az előnye a metamodell módosíthatósága, ill. esetleg több, különböző metamodellt használó rendszer felépíthetősége. A metametaszinten csak a legalapvetőbb betöltési-mentési, ill. szerkesztési műveleteket érdemes megvalósítani, a többi szint műveletei megegyeznek a tiszta kétszintű megoldáséval.

A tanulmány 2. részében (amely egy későbbi számban jelenik meg) az imént felvillantott megoldásokat részletesebben is bemutatjuk.

Hivatkozások

[1] Joaquin Miller-Jishnu Mukerji: *Model Driven Architecture*, OMG Document July 2001.

[2] Jon Siegel: *Developing in OMG's Model-Driven Architecture*, OMG White Paper, November 2001.

[3] Joaquin Miller-Jishnu Mukerji: *MDA Guide Version 1.0*, OMG Document July 2003.

[4] R. G. G. Cattell - D. Barry és mások: *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann Publishers San Francisco, USA 1999.

[5] *Human-Usable Textual Notation (HUTN) Specification Version 1.0*, OMG August 2004.

[6] *SILAN – the SILK language*, IQSOFT, Hungary 2000

[7] Rumbaugh - Jacobson - Booch: *Unified Modelling Language Reference Manual*, Addison-Wesley-Longman Inc. 1999.

[8] Kilián Imre: *Mixed strategy reasoning* An approach for resolution based verification of OCL constraints in UML models Pollack Periodica, Volume **2** Supplement Akadémiai Kiadó, Budapest 2007.

(Beérkezett: 2005. október 18.)

KILIÁN IMRE
PTE-TTK Informatika tanszék
7624 Pécs, Ifjúság u. 6.
kilian@gamma.ttk.pte.hu

MODEL DRIVEN SOFTWARE ARCHITECTURE

Imre Kilián

The expression of 'model driven software architecture' was first used by OMG in their MDA (Model Driven Architecture) project. The phrase 'model driven' refers in heir case to the process of software development. In their model software artifacts are designed independently from the software and hardware environments and platforms. Such platform independent models are later semi-automatically transformed to platform dependent models by using metamodels for the different platforms and transformation rules between them.

The present article uses the term in an other meaning. It refers to the runtime architecture of the software. This means that beyond the data required by the application the software contains also the domain model as an input of the software. The software maintains the model, and the model controls the calculation carried out by the software on the data including uniformly implemented data manipulations which realize the model.