Acta Agraria Kaposváriensis (2006) Vol 10 No 3, 135-142
Kaposvári Egyetem, Állattudományi Kar, Kaposvár
University of Kaposvár, Faculty of Animal Science, Kaposvár

Modellvezérelt modelltárház szövegszerű betöltés-mentés művelettel

Kilián I.
PTE-TTK Informatika tanszék, 7624 Pécs, Ifjúság u. 6.

ÖSSZEFOGLALÁS

A cikk meghatározza a modellvezérelt technológia fogalmát, amelyben az alkalmazói adattömeg mellett az adatok egy modelljét is tárolják, módosítják, mentik és/vagy betöltik. A szoftver ezen rétegét modellszintnek, az ezt végző szoftverösszetevőt pedig modelltárháznak nevezzük. Ha magát a modelltárházat is modellvezérelt módon akarjuk megvalósítani, az a modellvezérelt alkalmazói szoftverekéhez hasonló előnyöket kínál. A modellszint futásidőben is módosítható, az egész felépítmény sokkal rugalmasabb és könnyebben testre szabható, mint egy monolitikus módon felépített szoftveré. Az UML négyrétegű metamodell szerkezetének alsó három szintje még nyilvánvaló – az alkalmazói adatok, az alkalmazás modellje, ill. a metamodell, a negyedik réteg igazi értelmét azonban éppen a cikkben részletezett modellvezérelt modelltárház létrehozása teheti világossá. Mivel egy ilyen szoftvercsomag modellszintje maga a metamodell, ennek a metaszintjét, vagyis a meta-metamodellt kell beprogramoznunk rögzítetten és változtathatatlanul. A cikk egy ilyen modelltárház kérdéskörét és szerkezetét elemzi, és a szövegszerű betöltés és mentés példáján keresztül bemutatja ennek két modellvezérelt műveletét.
(Kulcsszavak: objektum orientált tervezés, UML, modellvezérelt szoftverkészítés, metamodell)

ABSTRACT

Model-driven model repository with textual save and load operations

I. Kilián
University of Pécs, Faculty of Sciences, Department of Informatics, H-7624 Pécs, Ifjúság u. 6.

The article defines the concept of model-driven software, that is beyond the mass of application data, the software model of this data is also stored, modified, saved and/or loaded. This part of the software is called the model level, and the corresponding component is called the model repository. The idea, that the model repository itself should be also designed in a model driven way, has similar advantages, like ordinary model-driven software has. That is, its model level can be modified in run-time, and the architecture enables a greater flexibility and customizability, than the monolithic way of software development. The lowest three of the four layer metamodel structure of UML is straightforward –the application data, its software model and the metamodel. The point of the fourth layer can be just understood when we want to create a model-driven model-repository. The model level of such a software is the metamodel itself, that means the meta-metamodel is the layer that has to be programmed in, hard-coded and unchangeable. The article analyses the structure of such software, and as an example presents two model-driven operations: the textual loading and saving of models.
(Keywords: object oriented design, UML, model driven software, metamodel)

BEVEZETÉS

Majd tíz éves már az Object Management Group által a köztudatba dobott Unified Modelling Language (UML) szabványgyűjtemény. A szabványgyűjtemény az objektum-orientált technológia szellemi öregjeinek külön-külön létrehozott tervezési módszertanait foglalta egy közös, egyesített szerkezetbe (*Rumbaugh et al.,* 1999). Az UML a tervezendő szoftver „modelljét" készíti el, majd a modell egyes vonatkozásait különböző „nézetek", vagy diagramok segítségével jeleníti meg. Az UML azóta „de facto" szabvánnyá vált az objektumorientált világban, annyira, hogy egy szoftvergyártók egész sor CASE eszközt építettek rá. Azóta magának a szabványnak is továbbfejlesztései, ill. specializációi születtek.

Az egyik ilyen a „modellvezérelt szoftverkészítés" fogalmát rögzítő Modell Driven Architecture szabványgyűjtemény (*Miller and Mukerji*, 2001; *Miller and Mukerji*, 2003). Ez a szabvány azonban a kifejezés utótagját hangsúlyozza, azaz azt tárgyalja, hogyan hozható létre különböző platform és gépfüggetlen UML modellekből egy immár konkrét hardver- és szoftver környezetbe beültethető modell, ill. azután ebből hogyan hozható létre maga a szoftver. A jelen írásban viszont a kifejezés előtagjára helyezzük a hangsúlyt: nem a szoftverkészítés modellvezérelt, hanem a kész szoftver. Vagyis azt tárgyaljuk: hogyan, milyen elvek szerint szervezhetők a modellszintet is magukban foglaló, „kétszintű" szoftverek, mit jelent, ha már a modellszintet is modellvezérelt módon akarjuk létrehozni. Legvégül pedig egy ilyen szoftver kísérleti létrehozásáról, ill. a megvalósítás gyakorlati tapasztalatairól számolunk be.

KÉTSZINTŰ, MODELLVEZÉRELT SZOFTVEREK ÉS A VARÁZSPÁLCA

A hagyományos szoftverkészítés egyszintű, monolitikus. A szoftverkészítés során kézhez kapjuk, vagy a megrendelővel való konzultációk során előállítjuk a szoftver leírását, specifikációját, ennek alapján elkészül a szoftver terve, vagy modellje, amely alapján a programozók beprogramozzák a szükséges műveleteket. Ez a folyamat nem teszi lehetővé a szoftver modelljének futásidejű módosítását, ez azonban a szoftverek nagy részénél nem is követelmény.

Egyes megrendelők azonban mindjárt a „varázspálcát" is be akarják programoztatni, ill. egyes esetekben a követelmények olyan mértékben rugalmas adatkezelést szabnak meg, amelyre nem lehet, de legalábbis nem érdemes az említett kötött szerkezetű, monolitikus szoftver fejlesztési modellt használni. Ilyen esetekben sokszor érdemes egy „modellvezérelt szoftver" létrehozását megfontolnunk.

Egy kétszintű, modellvezérelt szoftver az *1 ábrán* látható részekből áll.

- A *modellszinten* a szoftver kezelte adatok modelljét tároljuk, A modell futásidőben is módosulhat, ezért ezen a szinten olyan műveleteket kell megvalósítanunk, amelyek a modell betöltését, mentését, esetleg az inkrementális módosítását lehetővé teszik. Ezen lehetőségek azonban csak képzett felhasználók által használhatók. A modellszint műveleteit megvalósító szoftver összetevőt *modelltárháznak* nevezzük.
- Az *adatszinten* modellvezérelt adatműveletek megvalósítása. Ezen műveletek megvalósítása erősen támaszkodik a modellben tárolt információkra, azokat értelmezi, interpretálja szemben pl. a modellből szoftvert létrehozó CASE eszközök lefordított, kompilált megoldásaival.
- Mindkét réteg támaszkodik egy *végső adattároló rétegre*, amely a programfutást túlélő (perzisztens) információk tárolásáért és kezeléséért felelős. Ez a réteg tárolja az adatszinten keletkező és használt adatokat, vagyis a példányinformációkat is, és itt tároljuk a példányoknak megfelelő modelleket magukat is.

1. ábra.

Modellvezérelt szoftver szerkezete

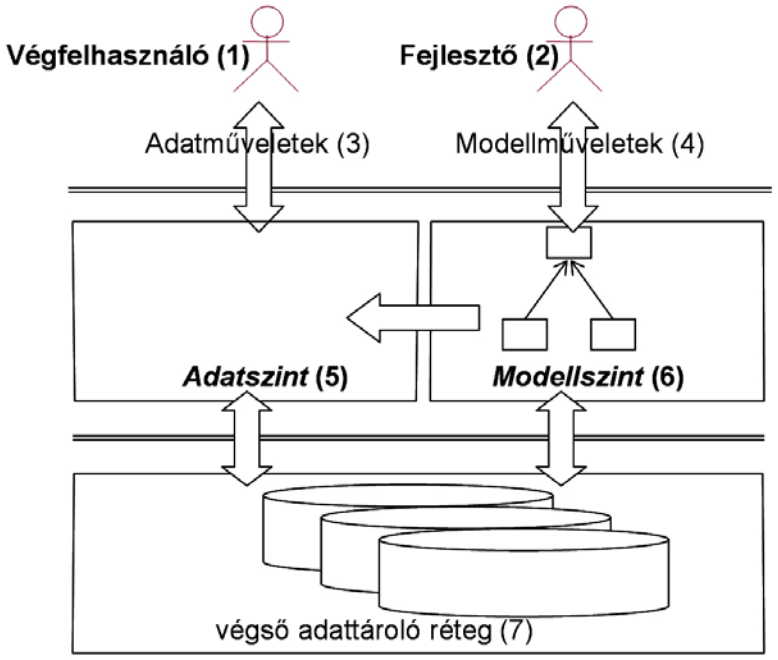

Figure 1. The structure of a model-driven software

End user(1), Developer(2), Data operations(3), Model management(4), Back end data storage(5)

Jellegzetesen modellvezérelt szoftverek a relációs adatbázis-kezelő szoftverek, bár ezek tervezésekor ezt a fogalmat még alig ismerhették. Itt a felhasználó, ha csak közvetve is, a konkrét adatokat lekérdező, ill. módosító (Data Query Language/DQL, Data Manipulation Language/DML) műveletekkel találkozik, amelyek futása erősen a beépített modell alapján, ill. annak vezérlésével történik. A modellszinten találhatók az adatsémát megadó (Data Definition Language/DDL) műveletek, amelyeket jellegzetesen a fejlesztők, ill. az alkalmazásokat készítő szakember használhatnak.

Melyek lehetnek egy modellvezérelt szoftver készítésének az indítékai és mik az előnyei? Akkor érdemes a modellszintet ilyen módon elkülöníteni és az adatműveleteket a modellel interpretáltan vezérelni, ha az alábbi feltételek fennállnak:

- ha nemcsak a program adatszintjén, de a modellszinten is kívánunk műveleteket végezni
- vagyis ha a modell maga futásidőben is módosulhat
- ha a módosuló modell maga is eredménye/terméke a szoftvernek
- ha a szoftvert többféle környezetbe tervezzük, amelyek között a különbség éppen a modell különböző kialakításával ragadható meg. Vagyis a különböző telepítések esetében a szoftvert részben a modell megadásával szabhatjuk testre.

A MODELLTÁRHÁZ ÉS AZ UML 4 RÉTEGŰ METAMODELL SZERKEZETE

A modelltárház esetében tehát az adatszint az alkalmazói szoftver modellje. Ha közvetlen megvalósítást tervezünk, akkor tehát az alkalmazói modell modelljét, vagyis a szoftver tervező rendszerünk metamodelljét kell beprogramoznunk. Emiatt ez a megközelítés a nem ad lehetőséget a metamodell semminemű módosítására, vagy későbbi változtatására.

A modelltárház megvalósításánál a modellvezérelt megközelítést alkalmazásának több előnye is lehet. Ilyen esetben még a metamodellt is elkülönítve tároljuk, amely tárolás alapja a rendszer valamilyen elvonatkoztatás útján kapott meta-metamodellje, vagyis az UML metamodell szerkezet legelvontabb rétege.

A modellvezérelt szoftverek általános jellemzésekor felsoroltakon túli előnye egy ilyen megoldásnak, hogy *lehetőséget ad a metamodell változtatására* is. Erre vagy akkor lehet szükség, ha a követelmények nem teszik szükségessé a teljes metamodell ábrázolását. Az is előfordulhat, hogy a szoftver kifejlesztéséhez a metamodell bizonyos mértékű bővítése, pl. esetleg új metatulajdonságok hozzávétele szükséges.

Az alkalmazói adatok/alkalmazói modell/metamodell/meta-metamodell négyest az OMG alapján *négyrétegű metamodell szerkezetnek* nevezzük. A *2. ábrán* látható fa csomópontjai között a „példánya" viszony áll fenn: az alkalmazói adattömeg példánya az alkalmazói modellnek, míg ez utóbbi modell példánya a metamodellnek. Általánosságban a faszerkezet n-ik szintjén levő modellek példányai az n+1-ik szintről feléjük mutató modelleknek. A faszerkezet azt is jelzi: a modellalkotási folyamattól függően egy adott példány- adatkészlethez többféle modell is létrehozható. A fa magassága elvben nem korlátos, a gyakorlatban azonban a meta-meta szintnél távolabbi elvonatkoztatásnak nincs jelentősége.

2. ábra

Az UML négyrétegű metamodell szerkezete

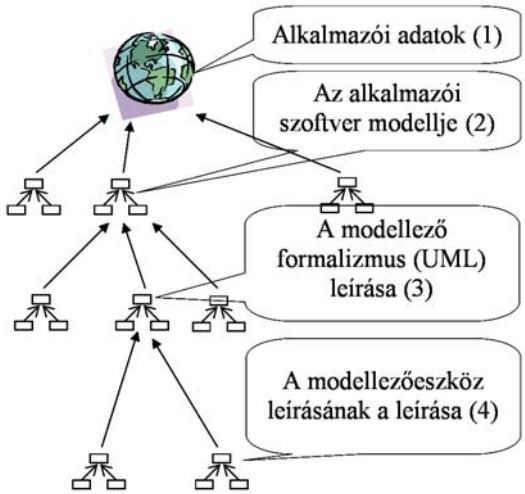

Figure 2. The four layered metamodell stack of UML

Application data(1), Application (software) model(2), Description of the modelling formalism (UML)(3), Description of the description of modelling tool(4)

METASZINTEK KEVEREDÉSE.

Modellvezérelten elkészített modelltárház esetében a következő érdekes kérdések merülhetnek fel.

1. Vajon összeegyeztethető-e a meta-metamodell a metamodellel magával. Ha a metamodell leírási módjában sajátmagát, vagy legalábbis valamely részhalmazát alkalmaztuk, akkor igen. Ilyen esetben a meta-metamodell adatszintje, vagyis a metamodell szintén összeegyeztethető a metamodell adatszintjével, vagyis az alkalmazói modellel, azzal, amelyet egy konkrét alkalmazói programhoz készítünk az adattárház segítségével.
2. Ha ez így van, akkor viszont maga a metamodell bevetíthető, mint a kezelt alkalmazói modell egy újabb csomagja, amelynek valószínűleg valamilyen rögzített nevet adunk. (Pl. az UML az OCL metamodellje számára az UML_OCL csomagnév használatát javasolja).
3. Ha a metamodellt magát, mint a tárolt és kezelt modell egyik csomagját tekintjük, akkor vajon módosítható-e a modelltárház eszközeivel maga a metamodell is? A válasz erre igenlő: ha egyszer a formátumok összeegyeztethetőek, és a metamodell bevetíthető a tárolt modell csomagjai közé, akkor bizonyára ugyanolyan eszközök szükségesek a módosításához is, mint bármelyik másik alkalmazói modellelem esetében.
4. Vajon mi történik egy ilyen módosítás hatására? A kérdés érdekes, a válasz és a megoldás nem kevésbé. A helyzet hasonló az önmagát akasztó hóhér, a magát operáló orvos, vagy a saját különböző részegységeit javító szerszámgép esetéhez. Bár egyes függelékeit a szerszámgép és az önjavító szoftver is kijavíthatja ilyen módon, általánosságban szólva azonban a *viselkedés megjósolhatatlan*. Éppen ezért az önoperálás nem szokásos, mert az, éppúgy, mint az önjavítás könnyen a javítandó rendszer összeomlásával járhat. A gyakorlatban ezért a hasonló eszközöket, pl. a modelltárházba bevetített metamodell-csomagot a legcélszerűbb *módosítás- és írásvédetten kivitelezni*. A *3. ábra* ezt mutatja be SILan modelleíró nyelven (*Benkő,* 2000), amely egy UML alapon készült, egyébként C-hez hasonló nyelvtani szerkezeteket használó nyelv.

3. ábra

Az UML metamodellt tartalmazó felhasználói modell

```
readonly package UML_META {
%...a metamodell elemei
%...
};
package UserModel {
%...az alkalmazói modell elemei
%...
};
```

Figure 3. The application model may also contain the UML metamodel

METAMODELL-VEZÉRELT MENTÉS ÉS BETÖLTÉS

A modelltárház adatszintjén elvégezhető modellvezérelt műveletek igen sokrétűek lehetnek. Mégis, talán a *legalapvetőbb a mentés-betöltés művelet*, amelyet még a legszegényebb megvalósításban és célszerűnek látszik kivitelezni.

UML modellek mentésére és betöltésére már az OMG is adott javaslatot: ez a Human Usable Textual Notation (*OMG*, 2004). A HUTN azonban nem a metamodellhez, hanem egy vezérlő adatszerkezethez köti a betöltés-mentés műveletpár megvalósítását, amelyben az adatszerkezet a művelet és a szöveg nyelvtanának paraméterezésére használható.

A metamodellvezérelt megoldás ennél rugalmasabb nyelvtanmegadást tesz lehetővé. Ehhez a következő feltételezéseket tesszük:

Modellhez kötött nyelvtanmegadás

A modellhez kötött nyelvtanmegadás *minden modellosztályhoz köt egy azonos nevű nemterminálist és az azt levezető környezetfüggetlen(-szerű) nyelvtani szabályt*. Egy ilyen szabály a modellosztályhoz rendelt nemterminálist annak tulajdonság- és részobjektumtípusaihoz rendelt nemterminálisokká, ill. egyes terminális szövegkonstansokká vezeti le.

A nyelvtant a környezetfüggetlen eszközöket természetes és intuitív módon kiterjesztő *metanemtermináliskészlet* segítségével adjuk meg. A nyelvtani szabály jobboldalának a megadása az adott metamodell-elemhez kötött név-érték pár (tagged value) segítségével történik. Az itt használható metanemterminális-készlet bizonyos nemterminálisok használatát segíti, elsősorban az osztálytulajdonságok és összetételek többszörösségeit és ismétlődéseit is figyelembe vevő elemkészletet tartalmaz. Jellegzetes elemei a következők:

`LISTOF(Nonterminal,Begin,Separator,End)`…ahol „Nemterminális” a tartalmazó osztály egy tulajdonságához vagy részobjektumához rendelt nemterminális, amelynek 1-nél nagyobb a többszörössége. „Begin”, „Separator” és „End” pedig a többszörös tulajdonságértékből képzett listát nyitó és csukó zárójel, ill. az elemeket elválasztó jel.

`IFTHEN(Cond,Then)`…ahol „Cond” egy Boolean értékű tulajdonságérték, „Then” pedig az a kifejezés, amelynek Cond teljesülése mellett a szabálybehelyettesítéskor elő kell fordulnia.

`IFTHELSE(Cond,Then, Else)`…az IFTHEN szerkezethez hasonló, amelyben .a feltétel hamis értékére is megadunk egy lehetséges generálandó szövegelemet.

A fentiek bemutatására nézzünk egy példát, ismét SILAN nyelven (*4. ábra*). A Package metamodell-osztálynak egy azonos nevű nemterminális felel meg. Az osztály „syntax” név-érték párja ezt a nemterminálist levezető szabály jobboldalát adja meg. A jobboldal elemei között találhatók tulajdonságnevek (comment, name), és a LISTOF metanemterminális, amely az „ownedElement” navigációs kifejezésből a „{” és „}” listazárójelekkel és a „;” elválasztójellel képzett listát írja le.

SZÖVEGGENERÁLÁS ÉS NYELVTANI ELEMZÉS

A két feladat közül a szöveggenerálás az egyszerűbb. Az ezt végző eljárás paraméterként kapja meg a kigenerálandó példányt magát, és a metaosztályból vett „syntax” név-érték pár értékét. A szöveggenerálás feladata ezen nyelvtanleíró string értelmezését jelenti, amelynek végrehajtása kézenfekvő.

4. ábra

A LISTOF nyelvtanleíró szerkezet alkalmazása az UML metamodellben

```
class ModelElement;

class Package:modelElement {
  attribute String comment;
  attribute String name;
  tagged value syntax='comment, „package", name,
LISTOF(ownedElement, „{", „;", „}")';
};

association {
  connection Package as owner navigable;
  connection ModelElement composite as ownedElement navigable;
};
```

Figure 4. Applying the LISTOF syntax description tag in the UML metamodel

A szövegfelismerés kicsit bonyolultabb feladat. Az ezt végző eljárás eredményképpen a felismert struktúrát is visszaadja. Ez a Prolog DCG eszközéhez hasonlatos módon működik, a különbség csupán a metanemterminálisok különleges kezelési módja.

TAPASZTALATOK ÉS JÖVŐBELI TERVEK

A leírt szoftver Prolog programnyelven készült az SWI-Prolog rendszer segítségével. Maga a programkód – a kísérleti jellegéből fakadóan is – párszáz – ezer sornál nem hosszabb. Ennek ellenére a program tesztelése és hibajavítása a szokásosnál lényegesen több bajjal járt. Ennek oka legvalószínűbben éppen a beépített metaszintugrásban keresendő, amely intuitíven a szoftver egy szinguláris pontjaként fogható fel. A szinguláris pont pedig, mint a komplex függvénytanban vagy a káoszelméletben, a szoftver viselkedésében divergenciát, megjósolhatatlanságot jelez, a szinguláris pont közelítése pedig általában instabilitást eredményez.

A dolog gyakorlati oldalát illetően, bár a leírt módszereket valamilyen konkrét ipari vagy mezőgazdálkodási folyamatra sosem alkalmaztuk, a szoftver-iparban történő alkalmazására rengeteg lehetőség nyílik. Többek között a leírt kikristályosított megoldások alapjául – előtanulmányként is – konkrét szoftverkészítési vagy -tervezési megrendelések szolgáltak.

A leírt eredményeket kétféle irányban lehetne továbbfejleszteni. Izgalmas kihívás lenne a metaszint-ugrást alkalmazó szoftverek elméleti hátterének tisztázása és kidolgozása matematikailag csiszolt formában.

A szerző alapvetően mérnöki hozzáállásának azonban inkább a metamodellvezérelt modell- és adatmigrációs eszközök további fejlesztése felelne meg, a „szemantikus bootstrapnak" nevezett, egyelőre még inkább csak ötletszinten megfogalmazott elképzelések kidolgozásával. Ennek teljes megvalósítása különböző metamodellű rendszerek közötti migrációs technika lehet, mindazonáltal korlátozott értelemben a leírt eredmények is tekinthetők a szemantikus bootstrap első megvalósíthatósági

tanulmányaként is, amelyben a célkörnyezet nem metamodellel, hanem csupán szövegszerű átalakítási szabályokkal van megadva.

KÖSZÖNETNYILVÁNÍTÁS

A leírt munka lazán kapcsolódik az IQSYS Rt által vezetett Sintagma kutatási projekthez, ez, ill. elődje a SILK projekt adta az elkészítéséhez a legfőbb ösztönözést is. Köszönetemet szeretném ezért kifejezni a projekt vezetőinek, munkatársaimnak, elsősorban Krauth Péternek és Szeredi Péternek.

IRODALOM

Miller, J., Mukerji, J. (2001). Model Driven Architecture. OMG Document.
Miller, J., Mukerji, J. (2003). MDA Guide Version 1.0. OMG Document.
Rumbaugh, J., Jacobson, I., Booch, G. (1999). Unified Modelling Language Reference Manual. Addison-Wesley-Longman Inc.
Benkő, T (2000). SILAN – the SILK language. IQSOFT, Hungary
OMG Specification (2004). Human Usable Textual Notation (HUTN) Specification. Object Management Group.

Levelezési cím (*Corresponding author*):

Kilián Imre
Pécsi Egyetem Természettudományi Kar, Informatika tanszék
7624, Pécs, Ifjúság u. 6.
University of Pécs, Faculty of Sciences, Department of Informatics
H-7624, Pécs, Ifjúság u. 6.
Tel.: 36-72-503-697
e-mail: kilian@gamma.ttk.pte.hu