Adatmigráció. Metamodell vezérelten?

Data migration. But metamodel driven?

KILIÁN Imre

7935 Ibafa, Gyűrűfű Cseresznyéskert.
Tel: +36 20 6655951
kilian@gamma.ttk.pte.hu

Abstract
The question of data-migration is as old as computation itself. If, instead of creating a unique software for the actual data formats, we create a general migration engine that uses also their metamodels as an input, then we gain valuable experience in the technology of metamodel-driven software. On the other hand this technology opens the wide possibilities of crossing metalevels and the technology of semantic bootstrap that is similar to syntactic bootstraps, and enables the porting of the entire solution under a different runtime environment.

Összefoglaló
Az adatmigráció kérdése egyidős a számítástechnikával. Ha azonban az adatformátumok közötti egyedi átalakító programok helyett azok metamodelljét is paraméterként használó általános átalakító motort készítünk, akkor ezzel kidolgozzuk a metamodell-vezérelt technológia egy alkalmazását. Másrészt a technológia rávilágít a metaszint-átmetszés nyitotta lehetőségekre, valamint a fordítóprogramok létrehozásánál használatos bootstrap módszer egy új, szemantikus változatára, ami a teljes megoldás újabb futtatókörnyezetbe történő átvitelét könnyíti meg.

Kulcsszavak: adatmigráció, modellvezérelt szoftver, metamodell

1. BEVEZETÉS

A számítástechnika kezdeteitől fogva forró kérdés az adatmigráció, ami alatt a fizikai adatátvitel mellett elsősorban az adatformátumok közötti átalakítást értjük. Ha a formátumátalakítást a szokásos módon – az adatok modelljei közötti átalakítást közvetlenül beprogramozva valósítják meg, az csupán azt a követelményt jelenti, hogy az átalakítást végző algoritmusnak a bemeneti modell kritériumait teljesítő minden adathalmazt át kell tudnia alakítani a konkrét adattartalomtól függetlenül.

A jelen írásban az adatmigráció általánosabb megközelítését mutatjuk be. Eszerint a modell-átalakítás szabályait nem az aktuális adatmodellekhez kötve és valamely konkrét (imperatív) programnyelven programozzuk be, hanem valamilyen *modell-leíró eszközzel* (pl. UML-lel, ill. OCL leíró nyelven). Ehhez az kell, hogy a rögzített forrás- és célformátum leírási módja is az említett modell leíró eszközzel legyen leírva, annak rögzített szemantikája alapján. Az adat- vagy végrehajtható programformátumok formális leírását az adott eszköz *metamodelljének* is nevezzük, amelyek tehát az átalakító program-motorunk adatai lesznek.

A metamodell-vezérelt adatmigráció esetében tehát nem a konkrét adatmodellek, hanem az őket leíró metamodellek közötti átalakító motort kell csupán beprogramozni, amelynek az egyes formátumok leírását, és az átalakítási szabályrendszert odaadva, már képes lesz konkrét adathalmazok közötti átalakítást végezni.

Megjegyzendő, hogy az alább leírt – *metamodell-vezérelt*, ill. *metaszint-átmetszéses* technológia fogalmai több okból is egyáltalán nem nyilvánvalóak – a továbbiakban igyekszünk a legtisztábban érthető megfogalmazással élni.

2. AZ OMG NÉGYSZINTŰ METAMODELL SZERKEZETE

Az Object Management Group nyereségmentes alapítvány az objektum-orientált technológia terén tesz megoldási javaslatokat és rögzít szabványokat. Szintén ők dolgozták ki az ún. *négyszintű metamodell szerkezet* elképzelését, ami a következőkből áll. [1].

A világ egyes konkrét objektumainak a leképezése egy-egy adatrekord, ami az objektum jellemzőit és annak értékeit tartalmazza (pl. Pityi Pál nézőponthegesztő adatait). Ez az *adatszint*, amelyeket a szoftverben létrehozott objektumpéldányok testesítenek meg.

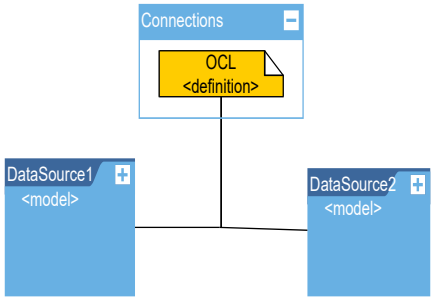
Amikor a világ egy szegmensére szoftvert írunk, vagy amikor egy adatbázisban a tárolás szerkezetét meghatározzuk, akkor az adatszint *modelljét* építjük fel. A modellszint nem, vagy csak kivételesen tartalmaz konkrét adatokat, mert az a világ adott alkalmazásban érintett összes adott típusú objektumát le kell írnia. A modell lehet tényleges modell is, ami önmagában nem futtatható, pl. az UML modellkészítési nyelven vagy valamely adatbázis sémanyelvén létrehozva, de egy futtatható programnyelven megírt program is a probléma egy modelljének tekinthető. A világ modellezett szegmensének az adatait a modell példányának is nevezzük.

Egyes modellek között átalakító műveleteket is lehetséges megvalósítani: ezek lehetnek pl. egy modell kódgenerátorai, vagy egyes programnyelvek közötti fordítóprogramok. Modellek általános kezeléséhez vagy átalakításához az adott modell-leíró formalizmus leírására van szükség, vagyis az alkalmazható modellelemek és azok összefüggéseinek leírására. Ezt nevezzük *metamodellnek*, ami ezért már nem is az adott alkalmazástól, hanem csak a *modellezési formalizmustól* függ. Egy adott modell tehát a metamodell példánya.

Olyan esetben, amikor ugyanazon szoftverben több, különféle modellezési formalizmust is kezelnünk kell (pl. egy fordítóprogramban), akkor az egyes metamodelljeiket célszerű egyöntetű módon tárolni, vagyis célszerű a metamodellek modelljét, a *meta-metamodellt* is rögzíteni. Az egyes kezelt metamodellek így a meta-metamodell példányai lesznek. Például az Eclipse alkalmazásépítő környezet tud effélét, az Eclipse Metamodel Framework (EMF) formalizmusa az általa kezelt metamodellek modellje, vagyis egy meta-metamodell. Ha az Eclipse rendszert valamilyen újabb programnyelv vagy modellezési eszköz kezelésére akarjuk felkészíteni, akkor annak metamodelljét éppen az EMF formalizmus segítségével adjuk meg.

3. MODELLEK ÉS METAMODELLEK ÖSSZEKAPCSOLÁSA

Eléggé kézenfekvő alkalmazói igény az *adatintegráció*, amikor az adatokat több, különféle rendszerben tároljuk, de egységesen akarjuk lekérdezni, vagy egymásba át akarjuk alakítani. Effélét a SILK projekt valósított meg. Az eredeti elképzelés szerint az egyes adatforrások lehetnek akár különneműek is, pl. relációs adatbázisok, OO adatbázisok vagy akár Excel táblák, de mind UML modellel leírhatók. Az egyes UML modellek elemei közötti összefüggések sztereotipizált asszociációkként és a hozzájuk kapcsolt megszorításokként voltak leírva. Erre az összekapcsolt modellre a lekérdező algoritmus egyesített lekérdezéseket futtatott, aminek az eredményét lehetett bármilyen kommersz alkalmazásban, tetszőleges hétköznapi célra használni. Míg az egyes adatforrások elérése hagyományos úton működött, magát az adategyesítést és az egyesített lekérdezést egy Prolog algoritmus hajtotta végre.



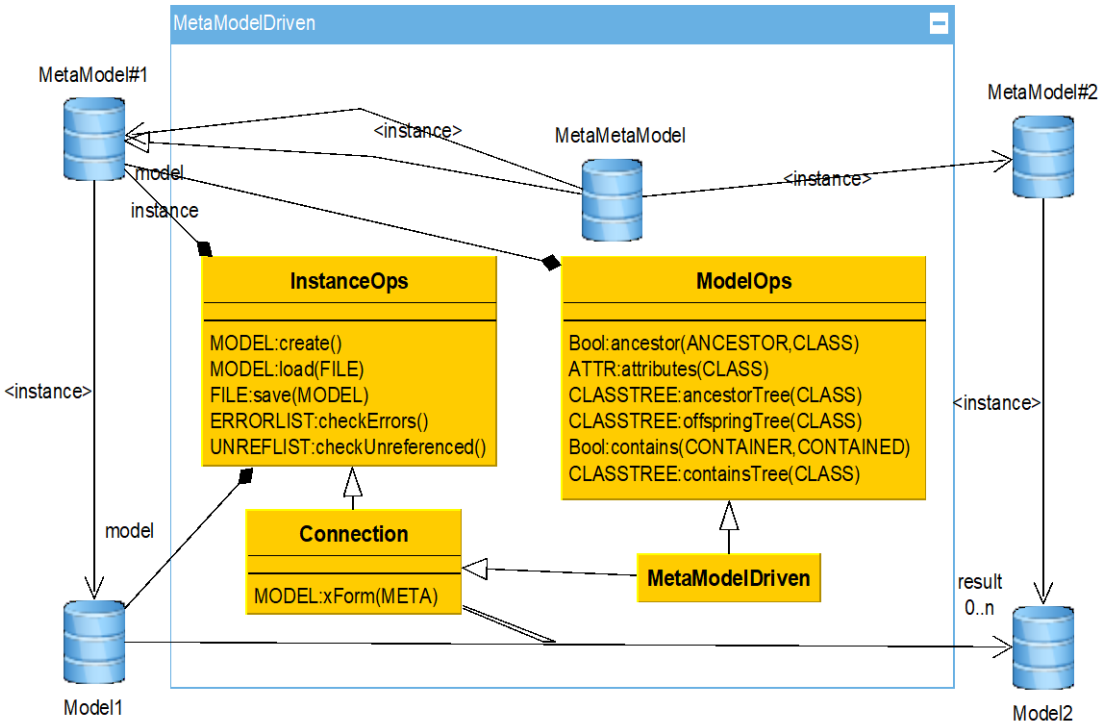
1. ábra. Adategyesítés a SILK projektben

A SILK csak közös, méghozzá az *OO metamodellel kompatibilis metamodellű rendszerek* összekapcsolására volt képes, és a működése során csupán a modellekben megadott típusú objektumok futásidejű létrehozására és egyesített lekérdezések dinamikus végrehajtására volt képes. Megjegyezzük, hogy ezeken túl még igen sokféle művelet értelmezhető és megvalósítható.

A gondolat egyik továbbfejlesztése az, amikor az egyes objektumok *külön metamodellhez* tartoznak, és az alapműveleteken – adott metamodellben adott típusú objektumok létrehozásán és törlésén túl még sok más műveletet is megvalósítunk. Ebben a megközelítésben az egyes (forrás-, ill. cél-) metamodellek között határozunk meg átalakítási szabályokat, célszerűen az UML kapcsolatok létrehozásával, ill. azokhoz OCL megszorítások rendelésével.

4. METAMODELL-VEZÉRELT SZOFTVEREK FELÉPÍTÉSE

Az alábbi ábra részletesen bemutatja egy metamodel-vezérelt átalakító motor működését.



2. ábra. Metamodell-vezérelt adatmigráció terve

A metamodelleken magukon is meghatározhatók különféle műveletek (`class ModelOps`), amelyek csak a metamodellek példányain, vagyis magukon a modelleken dolgoznak. A metamodell legtöbbször tartalmaz *taxonómiát* és/vagy *partonómiát* is, az ezeket kezelő műveleteket is tipikusan itt valósítjuk meg.

Az értelmezett műveletek másik – és fontosabb – köre egy `Model` osztályhoz tartozó objektumon, vagyis egy nyers adatpéldányon van értelmezve, ahol egy másik kontextuselem a modell maga, ami a `MetaModel` osztály egy példánya (`class InstanceOps`).

Ide tartoznak a létrehozó, (`create`), a betöltő és mentő, (`load, save`), és a különféle ellenőrző (`checkErrors, checkUnreferenced`) műveletek.

Az `InstanceOps` osztály kiterjesztése a `Connection` osztály, ami a fenti – egykontextusú – alapműveleteken túl a relációs felfogásban egy másik adattömeg-modell párosra is vonatkozik Az ábrán kapcsolóosztályként feltüntetett elem a tervben nem szimmetrikus. A kapcsolat maga is egyirányú, és a jelzett `xForm(META)` művelet ezért *procedurálisan értelmezendő*. A paraméter modell a metamodelljének egy példánya, ami megszabja az eredmény adathalmaz típusát, magát az eredmény adathalmazt viszont eredményparaméterként kapjuk meg.

A tervhez képest átalakító művelet helyett szükség lehet *szimmetrikus átalakítási reláció* meghatározására is (ld. a Prolog megvalósítást alább). Ez legegyszerűbben a fenti művelet kétirányú alkalmazásával megvalósítható meg. Teljesen határozatlan relációkiértékelésre – amikor a világ összes lehetséges modell-példány párosa létrejön – nincs szükség.

5. METASZINT ÁTMETSZÉS

Egyes számítási rendszerekben – gyakran csak ösztönösen – alkalmazzák a *metaszint átmetszés* megoldását. Ez nem jelent mást, mint egy modellszint és adatszint *egyöntetű ábrázolását*, Az ilyen ábrázolással *önleíró adatszerkezetek* hozhatók létre, vagyis olyanok, amelyek a saját leírásukat is – legtöbbször valamilyen elkülönített módon tartalmazzák.

Tulajdonképpen a Neumann elv is ugyanezt rögzíti, ami végsősoron utat nyitott a szoftver programokon működő programok, a betöltőprogramok, vírusok, fordítóprogramok és mások létrehozásának. Végsősoron az élővilág a DNS szekvenciáival is hasonló elveket követ. Mind a biológiai sokféleség, mind a számítástechnika megfigyelt fejlődése alátámasztja azt a hipotézist, miszerint az önleíró szerkezetek a fejlődés kulcsai, amelynek *perspektivái lényegileg beláthatatlanok*.

A mi esetünkben az átmetszhetőség feltétele az, hogy a modellek és a metamodellek szerkezete egymással kompatibilis legyen. Még pontosabban a metamodell ábrázolása feleljen meg a modellábrázolásnak, vagyis a *meta-metamodell legyen részhalmaza a metamodellnek*. Ez azt is megköveteli, hogy a hasonló adatszerkezet mellett egy *címzési egységet* is használni kell – beszéljünk pl. modulokról. Így a metamodell lényegileg az általa leírt modellszerkezet egy külön modulja lehet. Az ábrán ez úgy jelenik meg, hogy a `MetaMetaModel` osztály kiterjeszti a `Metamodel` osztályt, vagyis részhalmaza annak. Így az `InstanceOps` osztályban leírt példányműveletek ugyanúgy használhatók nyers adatpéldányokra (a modelljük vezérletével), mint modellekre a metamodelljük vezérletével.

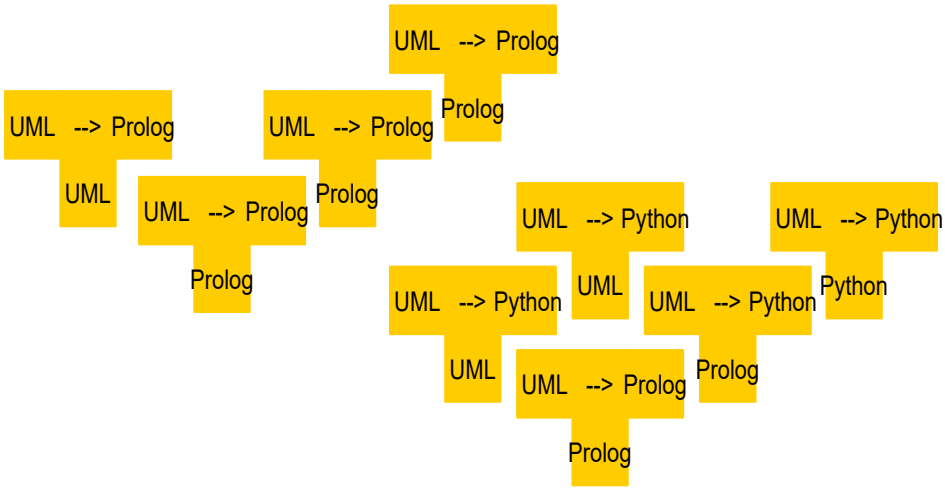
6. A MEGVALÓSÍTÁS SAROKKÖVEI

A megvalósítás SWI-Prolog rendszerrel készült. Egyrészt a Prolog a dinamikus adatszerkezeteivel pihekönnyűvé teszi a dinamikus szerkezetek, objektumok és/vagy hierarchiák létrehozását. Másrészt a nyelv relációs természete egy újabb különlegességre – a fordítóprogramok bootstrapjához hasonló szemantikus bootstrap megvalósítására ad lehetőséget.

Az alapcél a BIM IFC formátumának átalakítása volt – ezáltal egy automatizált adatátalakítási lehetőség megvalósítása az IFC épületmodellek és a 2024-es brassói SzámOkt konferencián bemutatott épületen belüli (inDoor) útkereső szoftver adatformátuma között [5]. Az alapcélt megvalósító szoftver jelenleg kísérleti állapotban van, tesztelés alatt áll.

7. SZEMANTIKUS BOOTSTRAP

Tételezzük fel, hogy létezik két metamodellünk, közöttük egy (pl. UML-OCL) átalakító szabályrendszerrel úgy, hogy a forrás metamodell az átalakító szabályrendszert írja le, a másik (a cél-) metamodell pedig valamilyen közvetlenül futtatható programnyelv modellje. Vagyis a szabályrendszer a metamodellek vezérletével egy forrásmodell futtatható programba történő leképezését teszi lehetővé. Ha most a szabályrendszert saját magára (kézzel) alkalmazva létrehozzuk annak a futtatható változatát, akkor megkaphatjuk az átalakító szoftver első változatát. Ezzel ismételten átalakítva az absztrakt (OCL) átalakítás-leírást, kapjuk a futtatható átalakító-szoftver második példányát. Az egyes generációk között ezután lehetőségünk van az absztrakt leíró nyelvet (OCL) akár fokozatosan bővíteni is. Ha több ilyen átalakító lépés után még mindig olyan szoftvert kapuk, ami gond nélkül működik, akkor mondhatjuk, hogy létrehoztunk egy modell-átalakító szoftvert (fordítóprogramot). A megoldás a fordítóprogramok szintaktikus bootstrap folyamatához hasonló, ezért *teljes szemantikus bootstrapnak* nevezzük, és az alábbi T (Tombstone) diagramokkal lehet ábrázolni [6]. A #0. változat a fordítás UML-OCL nyelvű kiindulási alakja. Ebből kézzel kapjuk a #1. alakot, ami immár futtatható nyelven áll elő (a példában Prolog nyelven). Ebből saját maga alkalmazásával állnak elő a további változatok, amelyek közben a forrásnyelvű (UML-OCL) alakban esetleg a nyelv fejlesztéseit/bővítéseit is beprogramozhatjuk.



3. ábra. Teljes és fél szemantikus boostrap T-diagramja

A leírt folyamat egyik változata, ha a futtatható alakba átalakító szoftver már működőképes, de egy harmadik megvalósítási környezetben szeretnénk futtatni, vagyis annak célformátumába (pl. Python) kell átalakítanunk. Ilyenkor az eredeti szabályrendszert kézzel átírjuk úgy, hogy az új célformátumba alakítson át. Ha ezt a forrás-szabályváltozatot a már meglevő végrehajtható alakba lefordító szoftverrel ténylegesen lefordítjuk, akkor kapunk egy futóképes szoftvert, amely a forrásformátumból az új célformátumba fordít, de sajátmaga még mindig a korábbi célformátumban fut (Prolog). Ha ezzel ezután sajátmagát mégegyszer lefordítjuk, akkor megkapjuk a szoftvert, ami már az új célformátumban fut, és ugyanerre a célformátumra fordít. Ez a megoldás a *szemantikus fél-bootstrap*, amivel újabb célformátumokba átalakító és ugyanott futó átalakító programokat lehet létrehozni.

8. EREDMÉNYEK, ÉRTÉKELÉS

A korábban említett inDoor épületen belüli útkereső szoftver prototípus [5] termékszintű hasznosításának a legkomolyabb akadálya az épületek szerkezetét leíró adatállomány kézi rögzítésének komoly erőforrás-igénye volt. A jelen megvalósítás ezért ennek az automatizálását tűzte ki célul metamodell vezérelt módon, az általános megközelítés kidolgozásával és alkalmazásával. Ez tehát éppen egy formátum-átalakítási (migrációs) feladatot jelent az épület elektronikus modellje, és a szoftver által használt formátumok között.

A megoldás természetesen feltételezi, hogy az említett elektronikus építészeti modell létezik, és megszerezhető, és ennek birtokában létrehozza az inDoor útkereső szoftver által igényelt (nem is csak kicsit) egyszerűsített adatformátumot. Az átalakítás tesztelés alatt áll, az első, legegyszerűbb IFC modellekből létrehozott inDoor adatállományok a cikk írásakor már megszülettek, és bizonnyal – a gyerekbetegségek kijavítása után – már bonyolultabb modellek is átalakíthatók lesznek.

A metamodell-vezérelt migráció természetesen nem varázspálca: egy egyedi átalakítóprogramba belevitt bonyolultság itt sem takarítható meg. Az egyedi megoldással szemben az átalakítás logikai bonyolultsága az UML-OCL leírásban van kódolva. A megoldás előnye az általánossága, ill. az, hogy az átalakítási bonyolultság egyértelműen külön van választva az átalakítóalgoritmus egyéb technikai elemeitől.

Az alkalmazás másik, már most is nyilvánvaló tapasztalata: az IFC formátum nem a legtakarékosabb – emiatt az átalakító programba optimáló lépéseket is be kell tervezni. Másrészt az átalakítás előtt – ill. esetleg utána – kézi adattisztításra is szükség lehet.

9. HIVATKOZÁSOK

[1] OMG Joint Revised Submission: Meta Object Facility (MOF) Specification *OMG Document 1997 Object Management Group*

[2] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks: EMF, Eclipse Modelling Framework, *Pearson Education 2nd Edition, Hoboken, New Jersey, USA, 2008.*

[3] Szeredi Péter, Benkő Tamás, Krauth Péter: SILK: alkalmazás-integrálás logikai alapokon. *Networkshop 2001. Nemzeti Információs Infrastrukúra Fejlesztési Program, 2001. (http://nws.iif.hu/ncd2001/docs/eloadas/82/index.htm elérés: 12-Sep-2025.)*

[4] Eastman, Charles; Fisher, David; Lafue, Gilles; Lividini, Joseph; Stoker, Douglas; Yessios, Christos *An Outline of the Building Description System*. *Institute of Physical Planning, Carnegie-Mellon University (September 1974). (*https://eric.ed.gov/?id=ED113833*, elérés: 6-Sep-2024.)*

[5] Kilián, Imre: A helyes utat keresve. *Erdélyi Magyar Műszaki Tudományos Társaság, SzámOkt 2024. konferencia kiadványa, Kolozsvár, pp. 94–97, 2024.*

[6] Patrick Closhen, Hans-Juergen Hoffmann, et al. 1997: *T-Diagrams as Visual Language to Illustrate WWW Technology*, *Darmstadt University of Technology, Darmstadt, Germany, 1997.*