Válasz Dr. Gyimóthy Tibor professzor úr bírálatára

MTA értekezés címe: Modelltranszformációk validálása és alkalmazása

Megköszönöm Dr. Gyimóthy Tibor professzor úrnak értékes észrevételeit, több ponton is megfogalmazott elismerő szavait. Külön köszönöm a szakterület-specifikus modellezéshez kapcsolódó tervezési minták bevezetéséhez, valamint a szakterület-specifikus tulajdonságok transzformáció közbeni validálásához tett észrevételeit. Megtisztelő számomra, hogy magas színvonalúnak értékeli az értekezésben bemutatott eredményeket. Köszönöm kérdéseit és megjegyzéseit, melyek megválaszolásával lehetőségem nyílt az eredmények néhány további részletének bemutatására is.

A dokumentumban dőlt betűtípussal szedve szerepelnek a bírálatból átvett kérdések, majd ezt közvetlenül követik a válaszaim.

---

1. kérdés: *A forrás modellekből cél modellekbe történő modelltranszformációk mellett lehetőség van a fordított irányú transzformációkra is (reverse engineering). Milyen eredmények vannak a szakirodalomban az ilyen jellegű transzformációk vizsgálatára, és van-e szerepük a modelltranszformációk verifikációja területén?*

A szoftver területen a *reverse engineering* egy rendszer elemzésének folyamata azzal a céllal, hogy a rendszer komponenseit és egymáshoz való viszonyait azonosítsuk, valamint más formában, többnyire magasabb absztrakciós szinten ábrázoljuk.

A fordított irányú transzformációk ugyanúgy modelltranszformációk két szakterület/szakterületi nyelv között, ahol speciális esetben a forrás és a cél nyelv megegyezhet. Tehát az alap eredmények, az esetek jelentős részében, használhatók mindkét irányban is (*forward engineering*, valamint *reverse engineering*).

A *forward engineering* majd a *reverse engineering* egy teljes fejlesztési kört tudnak együtt, egymást kiegészítve lefedni. A teljes kör bejárásával és a kiindulási állapothoz történő visszaérkezéssel, az ellenőrzésben, a helyesség igazolásban egymást tudják validálni.

A *forward engineering* tipikusan a magasabb absztrakciós szintről az alacsonyabb absztrakciós szinten található reprezentációba konvertálja a modellt, a *reverse engineering*, pedig többnyire alacsonyabb absztrakciós szintű reprezentációból magasabb absztrakciós szintre alakít át. Ilyen formában a két típusú modelltranszformáció eltérő jellegű szabályokat alkalmaz. Ugyanakkor ezen szabályoknak és a szabályokat teljes transzformációba integráló folyamatleírásoknak a verifikálása/validálása az esetek többségében megegyező eszközökkel és módszerekkel történik.

A szakterületek közti leképzés több esetben azért történik, mert egy adott ellenőrzést/vizsgálatot a forrás nyelvben nem tudunk megtenni. Az esetek többségében valamilyen matematikai formalizmusra történik ekkor a leképzés, ahol a szabályrendszer felállítható, az ellenőrzés elvégezhető. Hiba esetén a feladat nem csak megmutatni az ellenőrzést végző formális környezetben a hibát/vizsgálati eredményt, hanem feladat mindezt visszatranszformálni az eredeti domainbe a megfelelő nyelvi elemekhez/paraméter értékekhez, ezzel segítve a modelltranszformáció kidolgozójának munkáját. Ez a visszatranszformálás egy jellegzetes

feladat a modellfeldolgozók validálása/verifikálása területen, mivel ezen szempontoknál gyakori az ellenőrzési célokat szolgáló, más domainbe történő leképzés. Az ilyen jellegű visszatranszformálás is egy speciális esete a *reverse engineering* transzformációknak.

---

2. kérdés: *Hagyományos fejlesztési projekteknél (pl. nyílt forráskódú Java projektek) általában ezres- tízezres nagyságrendben készülnek egységtesztek. A generált, teszteléshez használt bemeneti modellek száma a tapasztalatok alapján milyen határok között mozog egy szakterület-specifikus nyelv esetén?*

A tesztelést segítő bemeneti modellek száma függvénye mind a tesztelt modelltranszformációnak, mind az alkalmazott forrás és cél nyelveknek. A meghatározó szempont a tesztelendő modelltranszformáció komplexitása: a modelltranszformációs szabályok darabszáma, azok mérete (LHS és RHS méretek), valamint a modelltranszformáció algoritmusát leíró vezérlés modell (*control flow model*). A kidolgozott megoldásban lehetőség van akár a teljes modellfeldolgozó tesztelésére, akár egy kijelölt részének tesztelésére.

A nyelvek mérete, a nyelvi elemek és a kialakítható példány modellek bonyolultsága szintén hatással van arra a körülményre, hogy egy modellfeldolgozó teszteléséhez mennyi és milyen komplexitású tesztesetre van szükség. A K+F projektjeinkben a szakterületi metamodellek többsége néhány tíz elemmel rendelkezik. Egy-egy ipari szabvány vagy környezet esetén néhány száz elemű metamodellekkel találkozunk. Például a Simulink modellek feldolgozása esetén nagyságrendileg 400 elemű metamodellel dolgozunk.

A tesztelést támogató, bemeneti modell generáló megoldás számos esetben úgy kerül alkalmazásra, hogy a kívánt feltételeknek megfelelő bemeneti modellt generálunk, majd ezen modell másolatát manuálisan módosítjuk speciális tesztesetek kielégítésére. Egy-egy generált bemeneti modellből több másolat készül, majd a másolatok több speciális teszteset kiszolgálására kerülnek átalakításra. Az átalakítások jelentik a csomópontok és élek felvételét/törlését, valamint a paraméterértékek módosítását. Ezen átalakítások mértéke a teljes tesztesethez képest jelentéktelen, tehát több tíz elemet tartalmazó modellben 1-2 elemet érint.

A fenti variálhatóságok miatt egy-egy modellfeldolgozó teszteléséhez kapcsolódóan volt példa a néhánytól a közel 100 darab bemeneti modell generálására. A kézzel történő teszteset pontosítás következtében a tesztesetek száma, a másolás és pontosítás műveletek révén, tapasztalataim szerint, többször eléri a generált tesztesetek 4-5-szörösét.

---

3. kérdés: *A napjainkban nagyon elterjedt agilis szoftverfejlesztési folyamat teremt-e új kihívásokat és lehetőségeket a modell alapú szoftverfejlesztési módszerekben?*

A modellalapú szoftverfejlesztés során a szoftverrendszert, vagy annak adott komponenseit, modellek segítségével definiáljuk, majd a rendszer forráskódját és további szoftvertermékeket az ún. modellfeldolgozók generálják. A modellfeldolgozók révén a modellalapú szoftverfejlesztési módszerek hatékony lehetőséget jelentenek és hatékony eszközöket is kínálnak a fejlesztési folyamatban.

Az agilis fejlesztési folyamat, a hagyományos vízesés modellhez képest, több szempontból szervezi eltérően a fejlesztés folyamatát. A rövid iterációs működés, minden ciklusban tartalmazza az elemzési, tervezési, fejlesztési, tesztelési, integrálási lépéseket. Ennek megfelelően, minden iterációban módosításra kerülnek a modellek és alkalmazásra kerülnek a modellfeldolgozók.

Itt a modellalapú megközelítés azon előnye játszik meghatározó szerepet, hogy maga a modell jelenti az elsődleges reprezentációt, továbbá egyúttal jelenteni tudja a folyamatosan frissülő dokumentációt is (illetve a dokumentáció generálása esetén annak kiinduló reprezentációját).

A folyamatosan változó, pontosodó igények végig kísérik az agilis fejlesztési folyamatokat. Ennek megfelelően a specifikáció és a megvalósítás is iteratívan fejlődik. A naprakészen tartásban és az iteratív fejlesztésben jó eszköz a modellalapú fejlesztési módszer, hiszen a számos iterációhoz kapcsolódó változások követése a modellek szintjén jól áttekinthető, valamint a modellek leképzése az implementációs szintre tud automatizált, illetve részben automatizált módon mind a hatékonyságra, mind a kód minőségére pozitív hatással lenni.

---

4. kérdés: *A kidolgozott folyamatot támogató eclipse alapú rendszerben pl. a Use case modellek támogatása mennyire felel meg az UML szabványnak, illetve mik a legjelentősebb hozzáadott értékek egy általános UML modellezést támogató eszközhöz képest?*

Az értekezésben tárgyalt *Eclipse* alapú megoldásban több szakterületi nyelv együttesen támogatja a specifikálási fázist. Ezen nyelvek közül egy a *use case*-ek leírását teszi lehetővé. Minden nyelv esetén a kialakítás célja volt mind a hatékony modellezési folyamat támogatása, mind pedig az automatikus feldolgozhatósághoz szükséges formális jelleg biztosítása a modellek számára. A *use case* modell az UML szabványból indul ki, ugyanakkor több ponton egyszerűsíti a specifikációt, elérhetővé illetve kötelezően kitöltendővé teszi a feldolgozásra hatással lévő paramétereket.

A megoldás fő hozzáadott értéke a hatékonyságot és a magas minőségű szoftvertermék előállítását támogató folyamat létrejötte. Mindezt úgy értük el, hogy évtizedes tapasztalatra alapozottan készültek el azok a forráskód szintű osztálykönyvtárak, amelyek függvényeit a generált kód használja. Tehát a funkciók jelentős része általános formában megtalálható a támogató osztálykönyvtárakban és keretrendszerekben. Ezen komponensek minőségét kvalifikált fejlesztők részvétele, valamint a kódok több projektben történő felhasználása, ebből következően alapos tesztelése fémjelzi. A forráskód generálás során jól paraméterezett függvényhívások sorozata kerül generálásra. Tehát a generálás az osztálykönyvtárak feletti, az adott követelményekhez igazodó specifikus réteg. A teljes megvalósítást tekintve ez a 30-40%-át jelenti a termékbe/szolgáltatásba kerülő forráskódnak.

A hozzáadott értékben a hatékonyságot a számos alkalommal újra használható modellfeldolgozók alkalmazása jelenti. Ezek a modellfeldolgozók céltudatosan illeszkednek a szakterületi nyelvekhez és ismerik az osztálykönyvtárakat, melyek funkcionalitásán az általuk generált projekt-specifikus kódréteg minősége alapszik.

5. kérdés: *A tervezési minták általában egy katalógus formájában jutnak el szélesebb közönséghez. A Graf IEC projektben, vagy más szakterületen készült ilyen katalógus?*

Igen, több szakterület esetén készültek szakterületi minták a VMTS-ben, amely minták katalógusként jelentek meg a keretrendszerben, valamint példákat mutató jelleggel a kapcsolódó publikációkban és teljes körűen a projektdokumentációkban. A mintákkal kiegészített szakterületek között található a mobilszoftverek fejlesztését célzó felhasználói felület leírását támogató nyelv, több UML nyelv, a VMTS-ben generált módon előállított Simulink nyelv (a Simulink metamodell felolvasásával és programozott módon a VMTS-ben történt létrehozásával), valamint a Graf IEC nyelv is.

Példákat, többek között, a következő publikációk tartalmaznak:

> L. Lengyel, T. Levendovszky, T. Mészáros, H. Charaf, Supporting design patterns in graph rewriting-based model transformation, 2nd Int. Working Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, pp. 25-32, 2007.
>
> T. Levendovszky, L. Lengyel, G. Mezei, T. Mészáros, Introducing the VMTS mobile toolkit., 3rd International Symposium on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2007. Kassel, Németország, pp. 587-592, 2008.
>
> T. Levendovszky, L. Lengyel, T. Mészáros, Supporting domain-specific model patterns with metamodeling, Software and System Modeling 8:(4), pp. 501-520, 2009.
>
> P. Fehér, T. Mészáros, P. Mosterman, L. Lengyel, Processing Simulink Models with Graph Rewriting-Based Model Transformation, ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems, MODELS 2012: Tutorials, Innsbruck, Ausztria, 2012.
>
> T. Mészáros, P. Fehér, L. Lengyel, Visual Debugging Support for Graph Rewriting-based Model Transformations, International Conference on Computer as a Tool, Eurocon 2013, Zagreb, Horvátország, pp. 482-487, 2013.

6. kérdés: *A transzformációk modellfeldolgozás közbeni ellenőrzésére vannak-e más eredmények a szakirodalomban, pl. a gráfminta alapú leírás alkalmazása erre a célra?*

A modelltranszformáció elemzését *statikusnak* mondjuk, ha az elemzés során a transzformáció definícióját, valamint a bemeneti és a kimeneti nyelvek definícióját (metamodelljét) használjuk fel. A *dinamikus* elemzés esetén egy adott bemeneti modellre vizsgáljuk a modellfeldolgozást, és eldöntjük, hogy az aktuálisan generált kimeneti modell esetén az elvárt tulajdonságok teljesülnek-e.

A statikus technika általánosabb, ugyanakkor megvalósítása jelentősebb kihívásokat jelent, hiszen minden lehetséges transzformáció ellenőrzésére fel kell készíteni.

A szoftverek kódjának - jellemzően a feldolgozást végző algoritmusoknak, programozási nyelveken megfogalmazott transzformációknak - az ellenőrzése két lépésben történik. Első lépésként a forrásfájlok lefordításával a fordítók a szintaktikai helyességet ellenőrzik. Fordítás során gyakorlatilag nem találkozunk a szemantikát ellenőrző módszerrel. Az első lépést illetően gyakori módszer továbbá a statikus forráskódelemzés, melynek egy lehetséges hatékony eszköze a Szegedi Tudományegyetemen kifejlesztett Columbus technológia.

A második lépés a lefordított bináris állományok futtatása. Ekkor maga a futó program testesíti meg a transzformációt. A bemeneti paraméterek jelentik a modellt. A logikai összefüggések, tehát a specifikációnak megfelelő működés, a szemantikai ellenőrzés itt történik. Ezen ellenőrzések és az ellenőrzési feltételek a programkódban, illetve általa elérhető egyéb reprezentációkban kerülnek megadásra. A jelentős eltérés a gráfújraírás alapú modellfeldolgozókhoz képest a feldolgozást végző folyamat és az ellenőrzést végző logika definiálásának a szintje. Programkód esetén alacsony, gráfújraírás esetén pedig magasabb definiálási szintről beszélünk. Ez a magasabb szint és a gráfújraírás matematikai háttere jelenti az ellenőrzéshez szükséges formális apparátus meglétét, és ad az igazolásokhoz, bizonyításokhoz szilárd hátteret.

A kérdésben példaként szereplő gráfminta egy paraméterezhető struktúra (csomópontok és élek alkotta gráf részlet). Felépítése és alkalmazási területei részben hasonlítanak az értekezésben bemutatott, a szakterület-specifikus tulajdonságokat és szabályokat definiáló sikerfeltételekhez és negatív sikerfeltételekhez. Ennek megfelelően, a futtatás során történő kiértékelést biztosító végrehajtással együtt, alkalmasak a modellfeldolgozás közbenső ellenőrzésére.

Varró Dániel és munkatársainak eredményei alapján a VIATRA2 és az EMF-IncQuery alkalmaz gráfminta alapú lekérdező nyelvet, melyek jól támogatják mind az újrafelhasználhatóságot mind a kompozícionalitást. A megközelítés deklaratív leírásmódot követ, ami jól optimalizálható lekérdezéseket eredményez.

A VIATRA2 keretrendszer szintén támogatja a generikus transzformációkat. A VIATRA2 modellterében a példány modellek és a típusokat jelentő metamodellek tárolása azonos módon történik. A típusparaméterek típusváltozókat jelentenek, ennek megfelelően a behelyettesítés művelete egy futási időben végrehajtott gráfmintaillesztési lépést jelent.

A következő publikációk tartalmaznak a dinamikus megoldásokhoz kapcsoló további megfontolásokat, a bennük szereplő eredmények mind motiválták a kutatómunkámat:

Heckel R, Küster JM, Taentzer G (2002) Towards automatic trans- lation of UML models into semantic domains. In: Proceedings of the appligraph workshop on applied graph transformation, pp 11–22

Küster JM, Heckel R, Engels G (2003) Defining and validating transformations of UML models. In: IEEE symposium on human centric computing languages and environments, Auckland, New Zealand, pp 145–152

BarbosaP,RamalhoF,FigueiredoJ,JuniorA,CostaA,GomesL (2009) Checking semantics equivalence of MDA transformations in concurrent systems. J. Univ. Comp. Sci. 15(11):2196–2224

de Lara J, Taentzer G (2004) Automated model transformation and its validation with AToM3 and AGG. In: Diagrammatic representa tion and inference, lecture notes in artificial intelligence, vol 2980. Springer, Berlin, pp 182–198

Giese H, Glesner S, Leitner J, Schafer W, Wagner R (2006) Towards verified model transformations. In: ModeVVa06

Hulsbusch M, Konig B, Rensink A, Semenyak M, Soltenborn C, Wehrheim H (2010) Showing full semantics preservation in model transformation—a comparison of techniques. In: Integrated Formal Methods, vol 6396. Springer, LNCS, Berlin, pp 183–198

Mens T, Demeyer S, Janssens D (2002) Formalising behaviour preserving program transformations. In: Proceedings of the first international conference on graph transformation. Springer, Berlin, London, pp 286–301

Ismét megköszönöm a Bíráló felvetett kérdéseit és észrevételeit, valamint a bírálat elkészítésére szánt figyelmét, idejét. Bízom benne, hogy a fentiekben megnyugtató válaszokat tudtam adni ezen kérdésekre, észrevételekre.

Budapest, 2019. június 10.

Lengyel László