Szakterület-specifikus modellezés

ASZTALOS MÁRK, MADARI ISTVÁN, MÉSZÁROS TAMÁS, VAJK TAMÁS, MEZEI GERGELY

BME Automatizálási és Alkalmazott Informatikai Tanszék
{asztalos, istvan.madari, mesztam, tamas.vajk, gmezei}@aut.bme.hu

Kulcsszavak: modellalapú szoftverfejlesztés, VMTS, banki alkalmazás-fejlesztés, BankCredit modell

A modellalapú szoftverfejlesztés egyre nagyobb hangsúlyt kap napjainkban. A modellek az eddig megszokott dokumentációs célokon túl felhasználhatók a követelmények formális leírására, verifikálására illetve automatizált kódgenerálás forrásaként is. Modellek használatával nagyban növelhető az elkészített komponensek újrafelhasználhatósága és az informatikához kevésbé értő szakértők mélyebben bevonhatók a fejlesztésbe. A modellező keretrendszereknek az a célja, hogy egy adott szakterülethez illeszkedő modellező környezetet gyorsan el lehessen készíteni, amit ezután a szakterület ismerői közvetlenül használni tudnak. Cikkünkben bemutatásra kerül, hogyan lehet felépíteni egy szakterület-specifikus modellező környezetet egy metamodellező keretrendszerben: tárgyalásra kerülnek egy modellező nyelv elkészítésének lépései a nyelvi elemek és kényszerek specifikálásától kezdve az elemek kinézetének meghatározásán át egészen a kódgeneráló automatizmusok elkészítéséig.

1. Bevezetés

Az informatika fejlődése és a felhasználói igények folyamatos növekedése egyre nagyobb és összetettebb alkalmazások készítését tette lehetővé és szükségessé. A felhasználók megbízható, testreszabható és mindemellett gyors alkalmazásokat várnak el a fejlesztőktől. Más iparágakban – mint például az autógyártásban vagy a szórakoztató elektronikában – a tervezési és gyártási folyamatok automatizáltsága már jó néhány éve elérte azt a szintet, ahol az említett követelmények alacsony költség mellett biztosíthatók. A napjainkban használt programozási nyelvek és eszközök viszont nem bizonyulnak elégnek ahhoz, hogy ugyanezt az automatizáltsági szintet képesek legyünk biztosítani a szoftverfejlesztés területén is. A cél eléréséhez arra lenne szükség, hogy tovább növeljük mind a kész komponensek újrafelhasználhatóságát, mind a feladat megoldására használt absztrakciós szintet. Ezt a két kulcsfontosságú területet célozza meg a modellvezérelt szoftverfejlesztés.

Az aktuális kutatások arra törekszenek, hogy az elkészítendő alkalmazás minél nagyobb részét tudjuk automatizáltan generálni különböző, általában grafikus modellek alapján. Annak érdekében, hogy a modelleket készítő szakértő munkáját megkönnyítsük, a használt modellező nyelvet a szakterülethez kell igazítani, azaz a nyelvnek pontosan azt a területet kell lefednie, amire használni akarják. Ezáltal a felhasználónak nem kell általános célú modellező nyelveket (pl. Unified Modeling Language – UML [1]) ismernie és a feladatot a használt nyelvre leképeznie. Az így testreszabott nyelveket *szakterület-specifikus modellező nyelveknek (Domain-Specific Modeling Languages – DSMLs)* magát a munkafolyamatot pedig *szakterületi modellezésnek (Domain-Specific Modeling – DSM)* nevezzük. Fontos kiemelni, hogy ha két szakterület akár csak kismértékben is különbözik egymástól, akkor is érdemes lehet különböző modellező környezetet készíteni, hiszen így hatékonyabbá tehető az azt használók munkája. A szakterületi modellező nyelvek gyakran grafikusak, a modellt egy típusos, címkézett gráfként reprezentálják, ahol a csomópontok a rendszer elemei, és az élek definiálják a kapcsolatokat az egyes elemek között. A *metamodellezés* az egyik legnépszerűbb módszer a használható modell elemek és a köztük kialakítható kapcsolatok meghatározására.

A metamodell gyakorlatilag egy szakterület-specifikus nyelv modellje, ami meghatározza azt a követelményhalmazt *(absztrakt szintaxis)*, amit a nyelv segítségével készített modelleknek mindig teljesíteniük kell. Ez a nyelv elemein kívül egyéb, szakterület-specifikus kényszereket (pl. Object Constraint Language – OCL [2]) is tartalmazhat. A nyelvi elemek kinézetét és viselkedését *(konkrét szintaxis)* a metamodell nem határozza meg, az általában szintén speciális modellező nyelvvel támogatott vagy a modellező rendszer API-ján keresztül lehetséges.

Egy DSML önmagában nem elégséges az automatizált program előállításhoz: elengedhetetlen szakterület-specifikus modellfeldolgozók alkalmazása, ami a magas szintű modellekből alacsony szintű modelleket (legvégül forráskódot) generál. Továbbá annak érdekében, hogy a kódgenerálást minél jobban leegyszerűsítsük, célszerű keretrendszert készíteni, amire a generált kód épülhet: ez a keretrendszer fogja össze az elérhető külső szolgáltatásokat, illetve azokat az üzleti funkciókat, amelyek a modellezett szakterületre jellemzők. Így a modellfeldolgozónak lényegében konfiguráló forráskódot kell előállítania.

Egy szakterületi modellező környezet elkészítése általában nem egyszerű feladat. Szükséges hozzá a modellezett terület szakértőjének (akinek gyakran nincs

informatikai tapasztalata) és az eszközt fejlesztő mérnököknek az együttműködése. Fontos, hogy pontosan felmérjék a követelményeket és ez alapján készítsék el a modellező nyelve(ke)t, mivel a szakterület-specifikus nyelv és a rá épülő feldolgozók élesen behatárolják a nyelv által lefedhető területet. Ugyanakkor más szemszögből nézve a modellezés és a programszintézis akár egy nagyságrenddel [3] is gyorsulhat a hagyományos eszközökkel történő fejlesztéshez képest.

Cikkünkben egy speciális, banki szakterület példáján keresztül demonstráljuk a modellalapú fejlesztést: a probléma specifikálásától a modellező környezet megalkotásán keresztül egészen a futtatható alkalmazás generálásáig. A példán keresztül mutatjuk be az általunk fejlesztett *Visual Modeling and Transformation System (VMTS)* [4] modellező keretrendszer képességeit.

Esettanulmányunkban egy gyakorlati életben előkerülő problémával foglalkozunk. Önkormányzatoknál, bankoknál, biztosítóknál gyakran felmerülő probléma, hogy a folyamatosan változó jogszabályok miatt a használt munkafolyamat-támogató szoftvereket a fejlesztőknek folyamatosan aktualizálniuk kell. Hasonlóképpen, ha például egy bank egy új pénzügyi terméket (például egy új típusú hitelt) kíván bevezetni, vagy egy meglévő kondíciós listáját módosítani (például a pozitív adósok listájának bevezetése), akkor szintén a fejlesztőkhöz kell fordulni, hogy a megváltozott igényekhez igazítsák az alkalmazások működését.

Ilyen problémákra nyújt megoldást a cikkben ismertetett munkafolyamat-támogató modellező környezet. Segítségével a munkafolyamat lépéseit és az azt megvalósító program felhasználói felületét vizuálisan tudjuk megtervezni, az elkészült modellek alapján pedig a kész alkalmazás automatikusan generálható. Az elbírálási döntést támogató alkalmazások generálásához szükségesek a bemeneti modellek, esetlegesen konfigurációs fájlok, valamint a fordításhoz szükséges nem generált forrásfájlok. Ezen bemenetekből a generálási folyamat végén egy fordítható projekt áll rendelkezésre. A generálási folyamat automatizált, ezáltal a fejlesztők igénybevétele nagymértékben csökkenthető, optimális esetben a módosításokat a szakterületi szakértő önmaga, fejlesztői segítség nélkül is el tudja végezni.

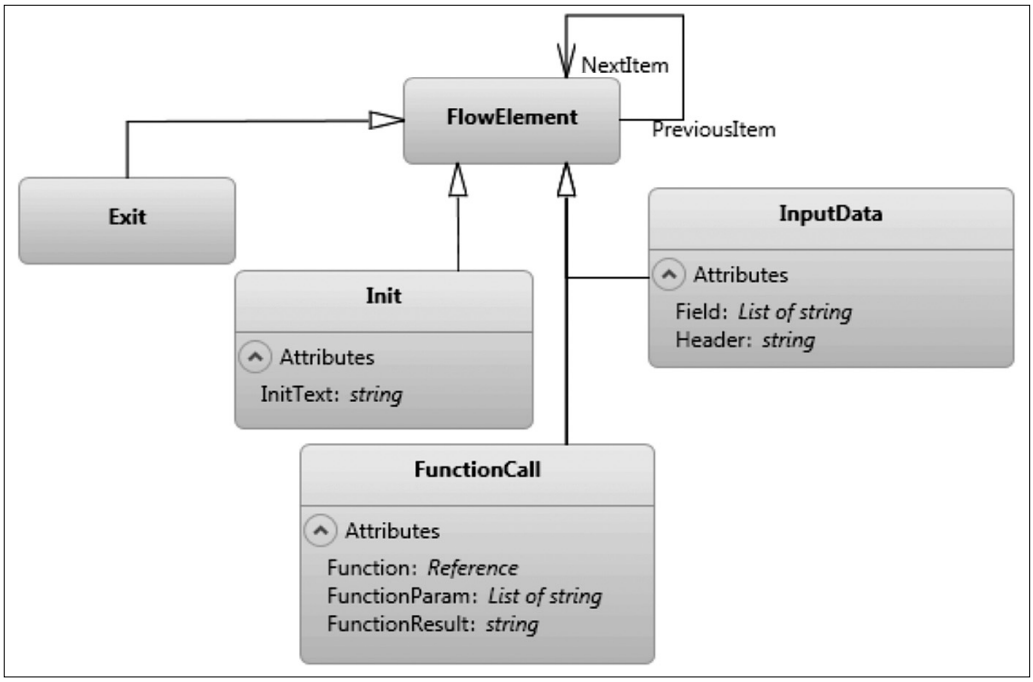
2. Modellezés VMTS környezetben

A VMTS egy gráfalapú metamodellező keretrendszer, kiegészítve a modellek szerkesztését, feldolgozását, transzformálását támogató komponensekkel. Metamodellek segítségével saját modellező nyelvet definiálhatunk, amelyben ezután a konkrét modelleket szerkeszthetjük. A metamodellek és modellek szerkesztésére egy vizuális szerkesztőfelület, a *VMTS Studio* alkalmazás áll rendelkezésre. Egy metamodell (szakterület) definiálása után a példánymodellek szerkesztéséhez használt felület testreszabható, a Studio funkciói kiterjeszthetők.

A modellek feldolgozásához készített komponensek között a legfontosabbak a gráfújraírás-alapú modelltranszformációs motor, illetve a sablonalapú automatikus kódgenerálás. A modellek nem csak a Studio alkalmazáson keresztül, hanem egy, a VMTS keretrendszer által nyújtott interfészen keresztül, programozottan is elérhetők. A modellek perzisztens tárolásához használt komponensek cserélhetőek, így tetszőleges adattárolási módot (pl. XML alapú állományok, adatbázis) implementálhatunk, illetve a VMTS képes már modellező környezetekkel együttműködni (pl. Matlab Simulink [5]).

2.1. A BankCredit szakterület bemutatása

A modellalapú fejlesztés első lépéseként a metamodellt kell definiálnunk, majd a példánymodellek elkészítése során lehetőség van azok kinézetének átalakí-



1. ábra
A BankCredit metamodell

tására. Végül a modellekből kódot generálunk, amelyből futó alkalmazást fordítunk. A *BankCredit* szakterülettel banki hitel-elbírálási folyamatokat kívánunk modellezni. A végcél, hogy olyan alkalmazást generáljunk a modellekből, melyek megkönnyítik a kezdeti hitelelbírálási folyamatot, azaz, a bekért adatok alapján segítenek eldönteni, hogy az adott paraméterekkel kaphat-e hitelt a bank ügyfele, vagy sem.

Mivel a metamodell példányai egy-egy folyamatot kell, hogy leírjanak, definiálunk kezdő- és végállapotot, továbbá az adatbekéréshez szükség van csomópontokra, amelyek lehetővé teszik tetszőleges típusú paraméterek beolvasását (pl. az ügyfél számlaszáma). Ezeken felül a hitel-elbírálási folyamat során szükség van a bekért adatok alapján üzleti/logikai műveletek elvégzésére (pl. BAR lekérdezés) és ez alapján döntéseket kell hoznunk. Mivel ezeket a műveleteket nehézkes lenne grafikusan megadni, egy olyan csomópontot is definiálunk, amely valamilyen módon lehetőséget nyújt a modellen kívül megadott metódusok hívására.

Ahhoz, hogy ezt a négy, logikailag teljesen különböző elemet egységesen tudjuk kezelni a folyamat megadása közben, egy közös őst, egy *FlowElement* nevű elemet definiáltunk. Ehhez az elemhez csatlakozik egy irányított hurokél, amely lehetővé teszi a FlowElement elemek egymás után kötését. Az él multiplicitását mindkét oldalon 0..*-ra állítottuk, hogy megoldható legyen egy elemhez több követő, illetve több megelőző elem definiálása. Ha egy csomópontnak több követő állapota van, akkor azok közül az élen megadott feltételek teljesülése alapján választ a rendszer.

Az elkészült metamodell a *1. ábrán* látható.

A kezdőpont *(Init)* egyetlen szövegmezővel rendelkezik, amelyben megadható a kezdőoldalra kerülő szöveg. A befejező állapotot az *Exit* csomópont reprezentálja, amelynek nincs egyetlen attribútuma sem. A bekérő elem *(InputData)* egy fejlécüzenetet és mezőneveket tartalmaz. A mezőnevek a bekérendő változókat reprezentálják, amelyeket a példánymodellben lehet megadni a típusukkal együtt. A típusok egy előre definiált értékkészletből kerülhetnek ki, így lehetőség van egész és racionális számok, szövegértékek, felsorolások megadására, továbbá speciális mezők is használhatók, mint például a bankszámlaszám-mező, amihez később a kódgenerálás során automatikus számjegyellenőrzést (Check Digit Verification – CDV) biztosítunk. A folyamat megadásának fontos része, hogy a bekért adatokat fel kell dolgozni, az üzleti logika megadására a *FunctionCall* csomópont ad lehetőséget. Ezek a függvények paraméterként használhatják a bekért adatokat *(FunctionParam)*, és eredményként visszaadhatnak egy értéket *(FunctionResult)*, amit a korábban leírt mezőnevek egyikében tárolhatunk el. Magát a meghívandó függvényt egy referenciával tudjuk megadni, amely külső függvényre mutat. Ez a függvény lehet egy DLL-ben definiált metódus, vagy egy webszolgáltatás.

Természetesen vannak olyan, a modellekre alkalmazandó megkötések, amelyeket nehézkes grafikusan leírni vagy feleslegesen bonyolulttá tennék a metamodellt, ezért lehetőség van Object Constraint Language nyelven megfogalmazott kényszerek megadására is. Ezek a szabályok objektumorientált nyelven teszik lehetővé megkötések leírását, mint például, hogy az *Init* csomópontokba, illetve az *Exit* elemekből nem mutathatnak élek.

2.2. Absztrakt és konkrét szintaxis

A szakterület-specifikus modellezés során a metamodellek definiálásával egy szakterület-specifikus környezetet hozunk létre, amiben a későbbiekben implementálhatjuk a modelleket. Egy ilyen környezet használata során elvárható, hogy az adott terület szakértői is eligazodjanak a használatában és tudjanak modelleket készíteni, vagy módosítani. Ebben segíthet, ha a környezet megjelenítését és funkcióit testre szabhatjuk.

A VMTS keretrendszerben a példánymodelleket alapesetben az úgynevezett absztrakt szintaxis felhasználásával jelenítjük meg (például ahogy az 1. és 2. ábrán láthatjuk), mely azonos minden *VMTS Studio*ban készített modellre. Ezzel a megjelenítéssel tudunk új modelleket létrehozni, szerkeszteni, attribútumaikat beállítani.

A VMTS rendszerben az egyes szakterületekhez különálló szakterület-specifikus szerkesztő környezetet definiálhatunk. Magát a szakterületet a metamodellel definiáljuk, a környezet elsősorban a példánymodellekhez készített konkrét szintaxis definíciójából áll. A konkrét szintaxis a modelleknek egy szakterület-specifikus megjelenítése, ami magában foglalja az egyedi kinézetet, illetve a VMTS rendszerbe beépített egyedi szerkesztőfelületeket is, melyekkel a modellek elemeinek tulajdonságait állíthatjuk.

A VMTS rendszer kiegészíthető külső komponensekkel, melyek a VMTS Studio felhasználói felületébe (új menűelemekként, eszköztárakként stb.) beépülve kiterjesztik a VMTS szolgáltatásait szakterület-specifikus funkciókkal. A konkrét szintaxist, komplexitásától függően, lehetőség van programozottan, a fejlesztőrendszer API-ján keresztül, vagy egyszerűbb esetben modellezetten definiálni. Az előbbi esetben tetszőleges felhasználói felület elkészítése lehetséges, amit .NET környezetben egy speciális attribútumokkal ellátott szerelvénybe fordítunk. Az utóbbi esetben egy speciális modellel, a VMTS környezeten belül, programozás (és programozói tudás) nélkül is definiálhatunk testreszabott megjelenítést *(Visual Plugin Developer – VPD* komponens [6]). Ez a megoldás gyorsabb, ugyanakkor csak egyszerűsített megjelenítést tesz lehetővé. Az entitások megjelenítése egyszerű alakzatokból állítható össze, ezeken különböző színű kitöltéseket, háttérképeket alkalmazhatunk, továbbá megjeleníthetjük bizonyos attribútumok értékeit egy-egy rögzített helyen.

A továbbiakban bemutatjuk a banki folyamatokat leíró metamodell egy példánymodelljét, illetve azt a konkrét szintaxist, amit ehhez a szakterülethez definiáltunk. Az *2/a. ábrán* látható egy példánymodell, mely egy hitelelbírálási kérelem folyamatát írja le. Az absztrakt szintaxist használó megjelenítésen látszik, hogy semmilyen szakterület-specifikus részt sem tartalmaz.

A 2/a. ábrán lévő modell konkrét szintaxissal történő definíciója a *2/b. ábrán* látható. Az új felület egyértelműen mutatja az elemek típusát (pl. szöveges mező), és a mező nevét is. A konkrét szintaxis tartalmaz továbbá speciális szerkesztő ablakokat is, például, az *InputData* típusú entitások mezőinek a beállítására. A megjelenő ablakban természetesen csak azok a mezők választhatók ki, melyeket korábban már definiáltunk a modellben, ezzel biztosítjuk, hogy ne tudjunk hibás modellt összeállítani, melyben nem létező modellre hivatkozunk. Az így kialakított környezetben tehát hatékonyabban lehet elkészíteni a modelleket, a környezet megjelenítése illeszkedhet a szakterületen használt speciális jelölésekhez, ezáltal szakértők számára könnyebben, intuitív módon is használható. Az egyedi környezet emellett lehetővé teszi speciális feltételek ellenőrzését, ezáltal biztosíthatjuk, hogy nem megfelelő modelleket ne tudjunk definiálni, valamint hibás modell esetén a hiba okát jelezni tudjuk a felhasználónak.

3. Szoftvertermékek generálása

A modellalapú szoftverfejlesztési módszerekben nélkülözhetetlenek a modellfeldolgozó programok, melyek segítségével automatikusan tudunk módosítani/létrehozni modelleket, illetve modellekből forráskódot generálni. A VMTS által nyújtott egyik lehetőség a modellek feldolgozására a sablon *(template)* alapú kódgenerálás.

A *Text Template Transformation Toolkit,* vagy *T4* [7] egy, a Visual Studio fejlesztőeszközbe beépített, de attól különállóan is használható technológia, melyben sablonok *(template)* definiálásával lehet kódot generálni. A kódgeneráláshoz egy speciális szintaktikájú szöveges állományt kell létrehozni, mely vegyesen tartalmaz sablonszöveg részeket *(text block)* és vezérlési logikát *(control logic)*. A vezérlési logika hagyományos, C#, vagy VB nyelvű program kódot jelent, ezzel írjuk le a feldolgozó program kódját. A szöveges blokkok tetszőleges szöveget tartalmaznak, melyek változatlan formában jelennek meg a kimeneti állományban. A sablon futásának kimenete mindig egy vagy több szöveges állomány. A sablonalapú szöveggenerálás jól használható például szöveges, illetve webes beszámolók és dokumentumok automatikus generálására, szoftverfejlesztés során automatikus forráskód generálására, illetve szakterület-specifikus modellezés folyamán a modellek feldolgozására.

A VMTS keretrendszer a T4 technológiára építve támogatja modellfeldolgozó sablonok készítését, szerkesztését, automatikus végrehajtását, illetve, amennyiben a generált szöveg maga is futtatható forráskód, akkor ennek a futtatását.

A BankCredit szakterület modelljeinek feldolgozására két sablonalapú modellfeldolgozót készítettünk. A két programmal automatikusan generált alkalmazások változtatás nélkül futtathatók, az egyik egy Windows operációs rendszeren futó asztali ablakos alkalmazás (a .NET környezetet használja), a másik egy webes szerverre telepíthető alkalmazás (Java környezetet használ), mely a *Google Web Toolkit* [8] csomagra épül. Ez a két megoldás jól szemlélteti a modellalapú szoftverfejlesztés néhány fontos előnyét:

(i) A modelleket egy szakterület-specifikus felületen állíthatjuk össze, anélkül, hogy a felhasználótól programozói tudást várnánk el.

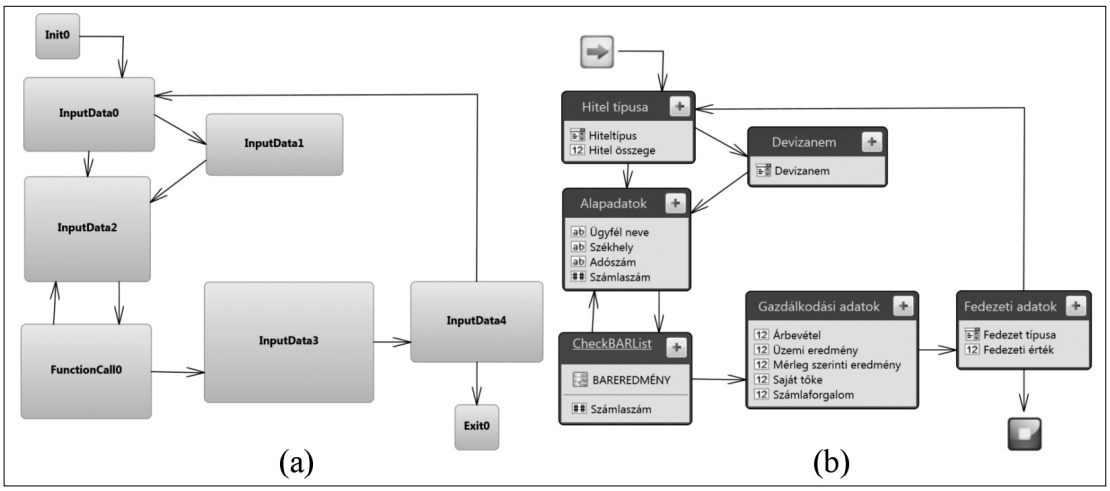
(ii) Az így előállított modellek mindegyikéből automatikusan generálunk futtatható alkalmazásokat. Ehhez a modellfeldolgozó programot egyszer kell elkészíteni, az egyes modelleknél nem kell külön kódot írni.

(iii) Egy adott szakterületen implementált modellek platformfüggetlenek, így tetszőleges platformra (a példánkban .NET, illetve Java platformokra) tudunk alkalmazást fejleszteni.

3.1. BankCredit modellek sablonalapú feldolgozása

A továbbiakban bemutatjuk, hogy a két sablonalapú modellfeldolgozó program milyen alkalmazást generál BankCredit modellekből.

A *3. ábrán* látható az 2/a. (ill. 2/b.) ábrán bemutatott példánymodellből automatikusan generált *Windows Presentation Foundation* (WPF, .NET technológia) alapú, illetve *Google Web Toolkitre* (GWT, Java-platform) épülő két alkalmazás felhasználói felülete.



2. ábra BankCredit modellek szerkesztése (a) alapértelmezett megjelenítés használatával és (b) testreszabott szakterület specifikus környezetben

(i) A generált alkalmazásban, a modellben szereplő minden entitáshoz egy-egy ablak (weblap) készül. Az ablakokon (weblapon) megjelenített információ függ az adott elem attribútumainak értékétől.

(ii) Minden modellben egyetlen *Init* típusú entitás van, ebből generáljuk a kezdőképernyőt, amin az adott elem *InitiText* attribútumának értékét jelenítjük meg szövegként.

(iii) Minden ablakon (weblapon) megtalálhatóak a gombok, melyekkel a banki folyamat egyes lépései között tudunk navigálni. A *Cancel* gomb kilép az alkalmazásból, a *Back* visszaugrik az előző ablakra (weblapra), a *Next* pedig a modellben meghatározott él mentén továbblép a banki folyamat következő elemére.

(iv) Minden *InputData* entitáshoz, a generált ablakban bekérjük azokat az adatokat, melyeket a *Field* attribútumokon belül megadtunk. A képernyőn, a mezők nevei láthatók, illetve egy-egy szövegdoboz, ahol az adatok megadhatók (például *3/a. ábra*).

(v) A modellben található *FlowEdge* típusú relációkhoz nem generálunk ablakot (weblapot), mert ezek határozzák meg, hogy a *Next* gomb megnyomására melyik új ablakba (weblapra) navigáljunk. Ha több, az aktuális entitásból kiinduló élet is tartalmaz a modell, akkor azt fogjuk követni, aminek a *Condition* attribútumban megadott feltétele teljesül. Amennyiben egy élen a *Message* attribútumnak értéket adtunk, akkor egy szövegdobozban (szövegmezőben) az adott üzenetet megjelenítjük a felhasználónak *(3/b. ábra)*.

(vi) A *FunctionCall* típusú entitások esetében nem jelenítünk meg külön ablakot (weblapot). Ilyenkor a háttérben megtörténik a megfelelő művelet elvégzése (web-szolgáltatás meghívása), a művelet paramétereit a mezők adják és az adott *FunctionCall* elemben meghatározott mező fogja tartalmazni a visszatérési értéket is. Ezután automatikusan továbblépünk, követve a *FunctionCall* entitásból kiinduló éleket.

(vii) A folyamat végét jelenti, ha eljutunk egy *Exit* típusú entitáshoz, ekkor a program automatikusan befejezi a működését.

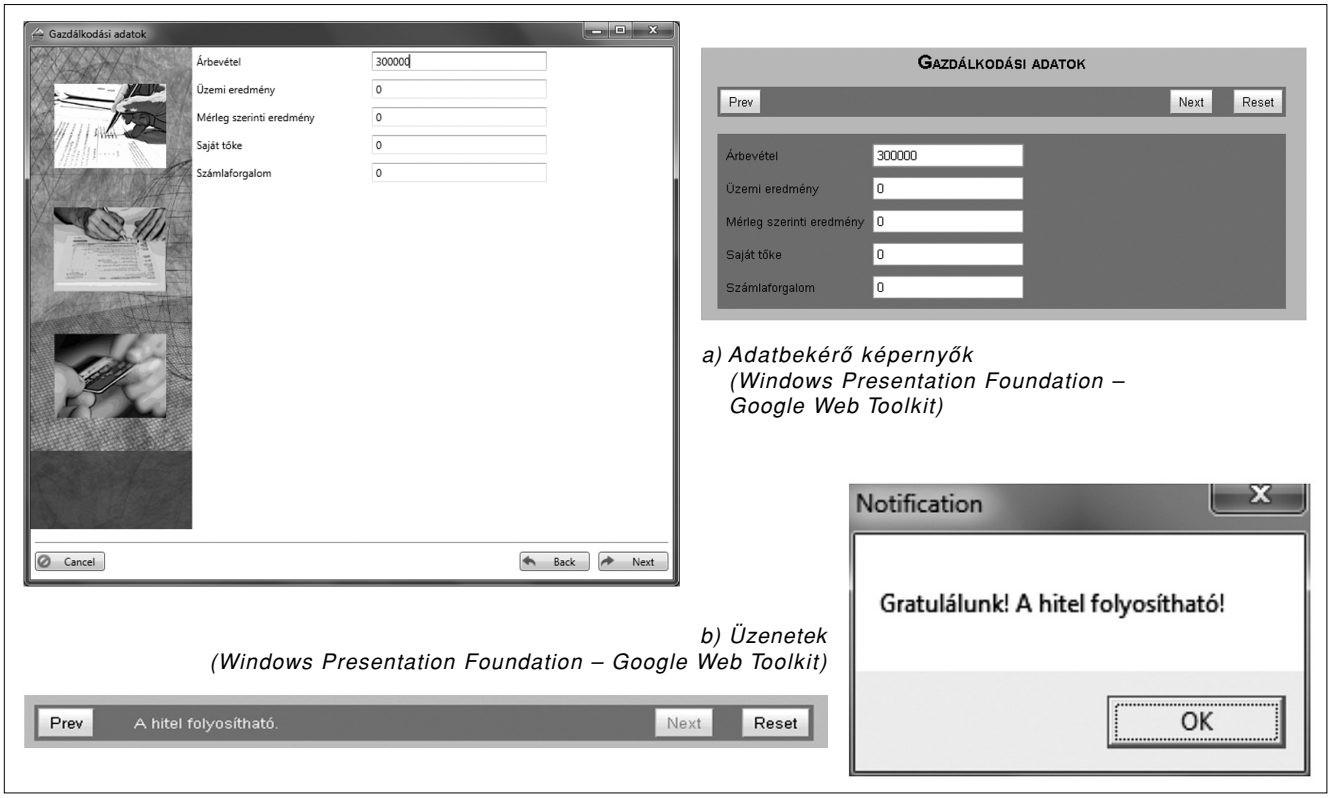
4. Összefoglalás

A szakterület-specifikus modellezés célja a kész komponensek újrafelhasználhatóságának és az absztrakciós szintnek az egyidejű növelése. Azáltal, hogy az adott feladatot abban a kontextusban oldjuk meg, ahol az megfogalmazódott, az adott területet ismerő szakértőt is be tudjuk vonni a fejlesztési folyamatba, hiszen számára ismerős környezetet használhat.

Cikkünkben bemutattuk egy szakterület-specifikus modellezőkörnyezet felépítésének lépéseit. Szemléltetésként egy banki hitelbírálati folyamat került ismertetésre, amelyhez könnyen használható és testreszabható modellező környezetet készítettünk.

Az esettanulmányon keresztül lépésenként bemutattuk, hogyan lehet meghatározni egy modellező nyelv elemeit, majd testreszabni azok kinézetét. Speciális, sablonalapú modellfeldolgozó programok segítségével de-

3. ábra Az automatikusan generált alkalmazások felhasználói felületei

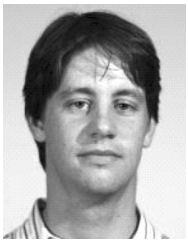


a) Adatbekérő képernyők (Windows Presentation Foundation – Google Web Toolkit)

b) Üzenetek (Windows Presentation Foundation – Google Web Toolkit)

monstráltuk, hogyan lehet automatizáltan platformfüggetlen alkalmazások kódját generálni. A tárgyalt módszerek nagyban hozzájárulnak a hatékonyabb szoftverfejlesztéshez.

A szerzőkről



ASZTALOS MÁRK harmadéves PhD hallgató a Budapesti Műszaki és Gazdaságtudományi Egyetemen Automatizálási és Alkalmazott Informatikai Tanszékén. Kutatási területe a szakterület-specifikus modellfejlesztés matematikai alapjai, illetve a gráfújraírás alapú modellfeldolgozó programok automatizált, formális verifikációja.

MADARI ISTVÁN 2007-ben szerzett mérnök informatikus diplomát a Budapesti Műszaki és Gazdaságtudományi Egyetemen. Jelenleg a BME Automatizálási és Alkalmazott Informatikai Tanszékén doktorandusz hallgató. Kutatási területe a gráf újraírás alapú modelltranszformáció, modell alapú fejlesztés, modelltranszformációval támogatott modellszinkronizáció.

MÉSZÁROS TAMÁS a Budapesti Műszaki és Gazdaságtudományi Egyetemen szerzett műszaki informatika diplomát 2007-ben. Jelenleg az Automatizálási és Alkalmazott Informatika tanszék doktorandusz hallgatója. Kutatási területe modellszimuláció, modelltranszformációs rendszerek optimalizálása és modellezési minták használata szakterületi modellezésban.

MEZEI GERGELY a Budapest Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai tanszékén szerezte meg doktori címét 2008.-ban. A fokozat megszerzése óta a tanszéken tanít, jelenleg adjunktusként. Kutatásait a metamodellező és szakterületi vizuális nyelvek, valamint feldolgozásuk témakörében végezte és végzi ma is.

VAJK TAMÁS a Budapesti Műszaki és Gazdaságtudományi Egyetem doktorandusz hallgatója. Műszaki informatika diplomáját 2007-ben kapta meg. Ph.D. kutatása során metamodellekhez csatolt kényszerek optimalizálási lehetőségeit vizsgálja.

Irodalom

[1] Craig Larman,
Applying UML and Patterns:
An Introduction to Object-Oriented Analysis and Design and Iterative Development.
Prentice Hall, October 2004.

[2] J. Warmer, A. Kleppe,
The Object Constraint Language:
Getting Your Models Ready for MDA, 2nd edition.
Addison Wesley, 2003.

[3] Steven Kelly, Juha-Pekka Tolvanen,
Domain-Specific Modeling:
Enabling Full Code Generation.
Wiley-IEEE Computer Society Press, March 2008.

[4] Visual Modeling and Transformation System:
http://vmts.aut.bme.hu

[5] The MathWorks: Matlab and Simulink,
http://www.mathworks.com/

[6] Gergely Mezei, László Lengyel,
Tihamér Levendovszky, Hassan Charaf,
A Model Transformation for Automated Concrete Syntax Definitions of Metamodeled Visual Languages.
ECEASST, Vol. 4, GraMoT, 2006.

[7] Visual Studio Text Template Transformation Toolkit (T4):
http://www.microsoft.com/

[8] Google Web Toolkit::
http://code.google.com/webtoolkit/