Komponens rendszerek modell alapú tesztelése

BÁTORI GÁBOR, THEISZ ZOLTÁN

Ericsson Magyarország, Software Engineering Group
{gabor.batori, zoltan.theisz}@ericsson.com

Lektorált

Kulcsszavak: szakterület specifikus modellezés, komponens alapú rendszerek, Deployment Tool

A távközlési szoftverek területén egyre növekvő igény tapasztalható az összetett rendszerek építése iránt, amely nehezen teljesíthető új fejlesztési paradigmák bevezetése nélkül. A szakterület specifikus modellezés újszerű megoldási módot kínál ahhoz, hogy megfelelő komplexitás menedzsmentet lehessen alkalmazni, valamint a modellezés során keletkező modelleknek a cikkben bemutatandó komponens alapú platformra (ErlCOM) történő átalakítása révén a végső rendszer összes képessége megvizsgálhatóvá válik. A módszer elterjedésével szükségessé válik a modell alapon készült szoftverek tesztelésére, de eddig még nem áll rendelkezésre egy általánosan elfogadott módszer erre nézve. A cikkben összefoglaljuk a szakterületen alkalmazható különböző módszereket, valamint bemutatjuk az általunk kifejlesztett Deployment Tool-t.

1. Bevezetés

Napjainkban egyre több új programtervezési módszertan lát napvilágot. Ezeknek az új módszereknek a legfontosabb célja, hogy csökkentsék az egyre komplexebbé váló szoftverek előállításának költségeit, valamint újrafelhasználhatóságot lehetővé téve a korábban befektetett munkát hasznosítsák. Az egyik legdinamikusabban fejlődő, és legtöbbet kutatott, módszertan a modell alapú programtervezés. Több módszer is létezik a modell alapú tervezés megvalósítására, de a módszerek alapvető céljai a következőek:

- *Befektetések megőrzése,* hogy a szellemi és anyagi befektetésekre ne legyenek hatással a technológiákban bekövetkező változások.
- *Automatikus implementáció,* hogy növelni tudjuk a fejlesztés hatékonyságát azzal, hogy a modellből automatikusan előállítható a végső forráskód.
- *Tesztelés,* hogy a fejlesztés korai szakaszában is lehetőség nyíljon a modell ellenőrzésére, akár annak közvetlen futtatásával, továbbá, hogy a fordító programok által létrehozott alkalmazások minősége jobb és egyenletesebb legyen, mint a más módszerrel létrehozottaké.

Az Object Management Group (OMG) által támogatott Modell Vezérelt Architektúra (Model Driven Architecture, MDA) az egyike ezeknek a modell alapú módszereknek. Az MDA egyre elterjedtebbé válik komplex szoftverek fejlesztése során. Az újrafelhasználást úgy érhetjük el az MDA filozófia szerint, hogy szétválasztjuk az alkalmazást leíró funkcionális részeket egy adott platformhoz kötődő implementációs részletektől. Platform alatt nem csak egy adott hardware környezetet értünk, hanem ide tartoznak olyan technológiai kérdések is, mint például az adatábrázolás. Az MDA-ban alapvetően magas szintű modellekkel ábrázoljuk a különböző részeket. Az MDA az UML-t (Unified Modeling Language) használja a modellek ábrázolására, mely *de facto* szabvány az objektum-orientált tervezésben. Az UML egy univerzális modellező nyelv, így igen sokféle módot enged meg a tervezőnek az adott probléma ábrázolására.

Az MDA-ban történő szoftverfejlesztés során kétféle modell típus jelenik meg. Az egyik a platform független modell (Platform Independent Model, PIM), míg a másik a platform függő modell (Platform Specific Model, PSM). A PIM egy funkcionálisan teljes modellje a rendszernek, így alkalmas szimulációra valamint tesztelésre. A platformfüggő modell ezzel szemben azokat az információkat ábrázolja, ahogy egy általános probléma oldható meg egy adott platformon. A két modellből transzformáció segítségével megkaphatjuk az adott problémát megoldó, az adott platformhoz jól illeszkedő forráskódot. Sajnos az egyes szakterületek speciális problémáinak megoldásához nem mindig jól illeszkedik az UML nagyon általános struktúrája, ezért egy szakterületet átfogó rendszer specifikussága és az UML alapú fejlesztés általánossága közötti ellentmondás meta-programozási architektúra alkalmazásával oldható fel. Metamodellezés alkalmazásával szakterület-specifikus nyelveket készíthetünk, melyek meta-generátorok által támogatott fordító programokkal szakterület-specifikus platformokra képezhetőek le.

Egy ipari környezetben is többször alkalmazott metaprogramozható eszköz a GME (Generic Modeling Environment) [1], melyet a RUNES IST projekt [2] kapcsán elosztott hibatűrő komponens alapú rendszerek tervezésére használunk. A RUNES komponens rendszer reflexivitását és futási idejű újra konfigurálhatóságát kihasználva garantálható, hogy az aktuális rendszerkonfiguráció mindig eleget tegyen a metamodell szabályainak. A komponens rendszer és a modell alapú tervezés nyújtotta előnyöket felhasználva különböző tesztelési módszerek dolgozhatóak ki. A cikk összefoglalást nyújt a modellezés során alkalmazható ellenőrzési eljárásokról, és az általunk kifejlesztett új módszerről, mely segíti a komponens rendszerek helyességének ellenőrzését.

2. Modell alapú szoftverfejlesztés (szakterület-specifikus modellezés)

Mint a bevezetésben említettük, a modell-alapú szoftverfejlesztés újszerű módon közelíti meg a programfejlesztés kérdését, hiszen míg hagyományosan a legfontosabb területek a különböző programozási nyelvek és operációs környezetek és a segítségükkel előállított forráskód voltak, addig a modell-alapú programfejlesztés inkább a szoftverrendszer logikai működésének precíz leírására törekszik. A modellkészítés folyamata kap nagyobb szerepet, mivel az MDA az elkészült modellt végrehajthatónak tekinti, tehát a modell vagy interpretált környezetben vagy generatív technikát alkalmazó modell-interpreterek segítségével automatikusan futtathatóvá alakítható.

Mielőtt a folyamatot részleteiben is megvizsgálnánk, előbb egy-két alapfogalom bevezetése elengedhetetlen. A legfontosabb fogalom maga a modell és a hozzá tartozó metamodell. A modell írja le mindazt, amely a szoftverrendszer működését specifikálja, mind funkcionális kapcsolatrendszerében, mind ezek dinamikus viselkedésében. Az összes fontos tényezője a rendszernek meg kell, hogy jelenjen a modellben, hisz csak ily módon biztosítható, hogy később a modell automatikusan futtathatóvá váljék. Természetesen a modellben megjelenő elemek osztályozhatóak, és a közöttük fennálló kapcsolatok szabályok formájába önthetőek, más szóval élve a modell egy metamodellen alapszik, amely megadja a modellek absztrakt szintaktikáját és statikus szemantikáját. Az UML nyelv esetében az OMG szabványosította a megengedhető modellezési módszert, amely tetszőleges modellezési problémákat hivatott megoldani. Az általánosság egyben jó dolog, hiszen univerzális hatókörű, másrészt rossz, mert nem optimális egyik alkalmazási területen sem.

A szakterület-specifikus modellezés (Domain Specific Modeling, DSM) ezzel szemben azon az ötleten alapszik, hogy minden alkalmazási terület számára készít egy olyan specifikus metamodellt, amely az adott terület problematikáinak a legjobban megfelel, ezáltal lehetővé téve alkalmazás-családok gyors előállíthatóságát a metamodell és/vagy modell újrafelhasználása által. A létrehozott modellek absztrakciós szintjük szerint megfeleltethetőek az OMG PIM és PSM kategóriáinak, de lényükből fakadóan ezen besorolásnál rugalmasabbak. A szakterület-specifikus modellezés technikája megköveteli, hogy a metamodellek előállítása könnyű legyen, és hasonlóan az OMG által szabványosított MOF (Meta Object Facility) metamodellező nyelvhez, a DSM környezetnek is rendelkeznie kell egy hasonlóval. Továbbá a hatékony fejlesztés elengedhetetlenné teszi egy metamodell alapján modellek építését támogató szerkesztő program meglétét is. A bevezetésben megemlített GME egy mind kutatási, mind ipari környezetben jól alkalmazható DSM környezetet szolgáltat.

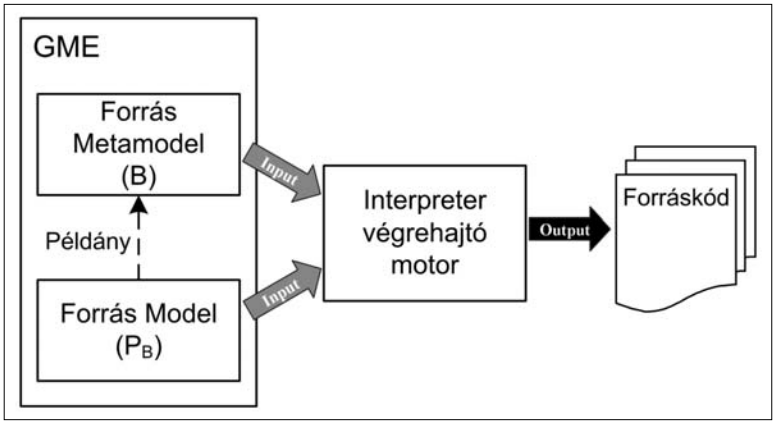
Miután az alapfogalmak bevezetésre kerültek, bemutatjuk a fejlesztés két alaplépését. Először azt vizsgáljuk meg, amikor a modellből generált kód valamely már meglévő programnyelv forrásszövegeként jelenik meg. Az *1. ábrán* látható, hogy a forrás metamodellt és modellt felhasználva a modell-interpreter metamodell mintákat felhasználva bejárja a modellt és a benne foglalt információk alapján előállítja a célnyelvű szöveget.

Fontos kiemelni, hogy míg a statikus szemantikát a metamodellen alapuló modell-szerkesztő ellenőrizte, addig a dinamikus viselkedés szemantikáját a modell-interpreter helyezi el a kódban.

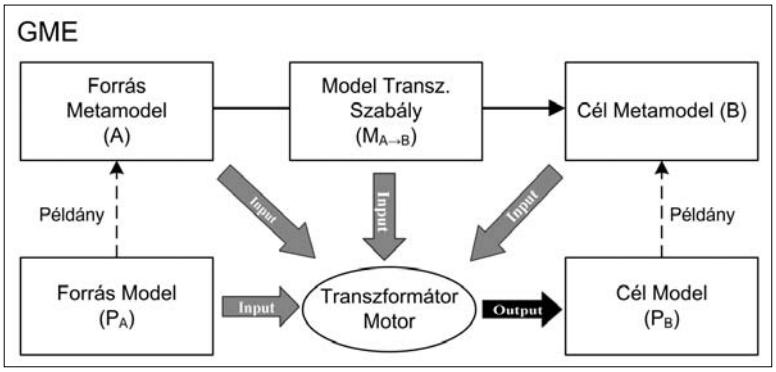
Természetesen maga a modell-interpreter is előállítható rekurzívan modell alapon és ily módon egy-két általános programozási mintát megvalósító véginterpretertől eltekintve a teljes folyamat modell-alapúvá alakítható. A második alapeset a *2. ábrán* látható. A különbség az előbbi esethez képest az, hogy itt most nemcsak a transzláció forrása, hanem a célja is metamodell, modell párosként reprezentálódik. A fordítás folyamatában a forrás-metamodell, a forrásmodell valamint a cél-metamodell ismeretében, továbbá a fordítási szabályok leírásának megadásával – amely az esetek többségében gráftranszformációs algebrát használ – a cél-modell automatikusan generálódik.

A fenti két alapeset természetesen kombinálható is oly módon, hogy végeredményében a szoftverfejlesztés gráftranszformációk sorozatokat lezáró modell-interpretációk rekurzív rendszerévé váljék.

1. ábra Modellfordítás interpreterrel



2. ábra Modellfordítás gráftranszformációval



3. Komponens alapú rendszerek

A komponens alapú technikák jelentős szerepet játszanak komplex elosztott rendszerek tervezésekor, mivel a komponensek létrehozásával a rendszer funkcionalitása könnyen „csomagolhatóvá” válik. A komponens rendszerek osztályozása is csomagolási tulajdonságuk alapján történhet.

Az általunk fejlesztett ErlCOM [3] és a RUNES [2] komponens rendszere is reflektív kauzális hierarchikus port alapú csomagolást biztosít a telepített rendszer működése során. A hierarchikusság arra utal, hogy a komponensek egymásba csomagolhatóak, a port alapúság azt jelenti, hogy a komponensek közötti kommunikáció kizárólag összekapcsolt kompatibilis interfészek segítségével történhet, reflektivitás a komponensek kapcsolatrendszerének futás közbeni lekérdezhetőségét foglalja magában, míg a kauzalitás fejezi ki azt, hogy ha a komponens kapcsolat-gráfon változtatunk, akkor a változtatás ténylegesen végre is hajtódik a futó komponensrendszeren. A tulajdonságok megvalósításáért a komponens futtató mag (Component Runtime Kernel, CRTK) felel.

A fent definiált komponensrendszer felfogható platform metamodellként, és egy adott telepített rendszer egy modelljeként. A modell-interpreter szerepét a CRTK veszi át a 4. fejezetben részletezendő semantic anchoring-nak megfelelően. Így gyakorlatilag a futtató komponensrendszert beleintegráltuk a modell alapú szoftverfejlesztés folyamatába, mint PSM-et. A különböző komponens rendszerbeli megvalósítást a megfelelő modell-interpreter alkalmazásával válthatjuk ki az alkalmazási területnek megfelelően. A különbség például az ErlCOM és a RUNES rendszer között abban jelenik meg, hogy míg az előbbi magas rendelkezésre állású, hibatűrő rendszerek építésére szolgál, addig az utóbbi főleg szenzorhálózatok esetén kerül majd alkalmazásra.

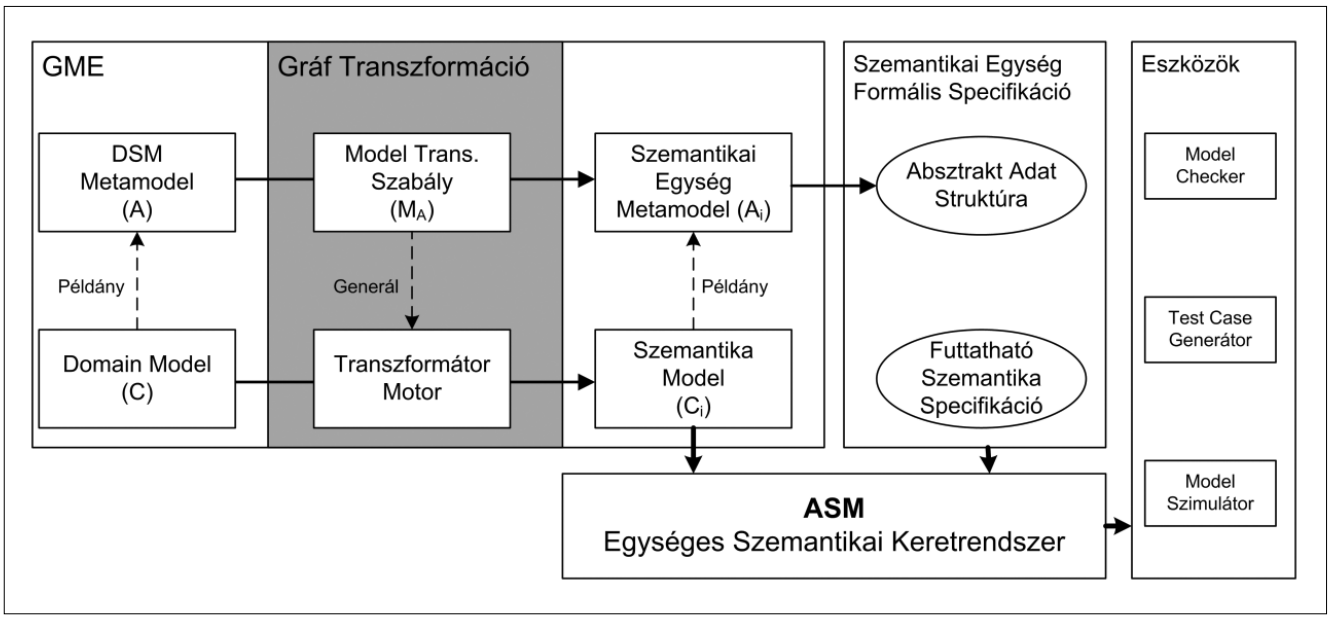
4. Tesztelés a modellalapú szoftverfejlesztésben

Metamodellezés során a megoldandó problémáknak először megpróbáljuk a statikus szerkezetét ábrázolni, azaz megtalálni azokat a strukturális építőköveket, melyekkel az adott problémák leírhatóak. A statikus elemek attribútumainak beállításával lehet az adott problémára jellemzőre alakítani egy adott strukturális elemet. Ám a statikus szerkezet leírása nem elegendő ahhoz, hogy a probléma dinamikáját is megfogalmazzuk. Ahogy az egyes szakterületek bevezetnek bizonyos fogalmakat, amiket az adott szakterület-specifikus modelleknek tükröznie kell, úgy az egyes szakterületek különböző dinamikát leíró modelleket is használnak. Erre azért van szükség, mert eltérő dinamika-leíró nyelvek alkalmasabbak különböző szakterületek problémáinak hatékonyabb, tömörebb leírására.

Azonban a szakterület specifikus dinamika-leíró nyelvek széles kategóriája kifejezhető az alapműködést leírók segítségével. Ilyen alapleírók például a következők lehetnek: véges állapotgép (Finite State Machine, FSM), időzített automata (Timed Automaton) vagy hibrid automata. Természetesen, hogy ezeket a nyelveket használhassuk a modellezés során, léteznie kell a nyelveknek megfelelő metamodelleknek a modellező eszközben.

A szakterület-specifikus modellek tesztelésénél komoly gondot okoz a modellező nyelvek diverzitása, ezért ha vizsgálatokat szeretnénk végezni a modelleken szükséges, hogy átalakítsuk egy olyan formára, mely megkönnyíti az ellenőrzések elvégzését. Ilyen megoldás lehet az úgynevezett semantic anchoring [4]. A semantic anchoring lényege *(3. ábra)*, hogy az egyes dinamikát leíró nyelveket megpróbáljuk leképezni egy matematikailag jól kezelhető közös nyelvre. Ez a közös nyelv a matematikailag bizonyítható Absztrakt Állapotgép (Abstract State Machine, ASM) [5], mely egy formális keret-

3. ábra Eszköz-architektúra a szakterület-specifikus fejlesztéshez Semantic Anchoring-on keresztül



rendszer szemantikai egységek specifikálásához. Az ASM-et sikeresen használták sok nyelv szemantikájának a leírására, így a C, a Java vagy az SDL (Specification and Description Language) specifikálására is.

Az ASM nyelvre épülve több teszteset-előállító és -bizonyító eljárás is létezik, így erre a nyelvre leképezve a szakterület-specifikus dinamika modellünket a már meglévő eszközöket felhasználva ellenőrizni tudjuk az általunk létrehozott dinamika leírásának a helyességét. Természetesen a leképzések nem triviálisak, de [4] bemutat egy gráftranszformáción alapuló eljárást. A gráftranszformációk nagy előnye, hogy nem csak a transzformációk kiinduló és végállomása formális modell, hanem magukat a transzformációs szabályokat is formális módszerekkel tudjuk megfogalmazni, és így a 2. fejezet alapján a transzformációs szabályok is modellnek tekintendők.

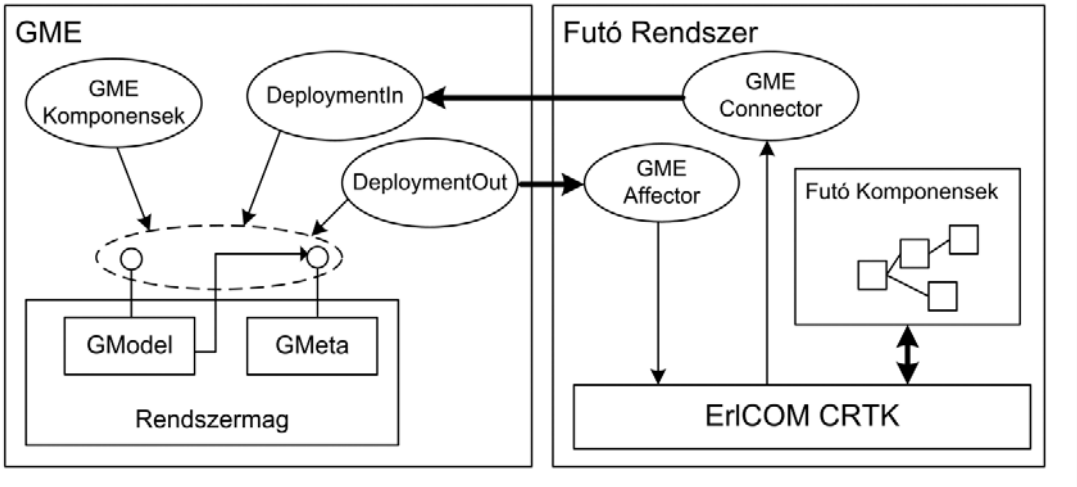
A semantic anchoring azonban nem mindig elegendő a teljes rendszer működésének a vizsgálatához, mivel a rendszer statikus szerkezete is kihatással lehet a végső implementáció működésére. Ilyen eset lehet az, amikor egy elosztott rendszer esetén a rendszer különböző elemeit rosszul helyezzük el az adott erőforrásokon. Ekkor a feleslegesen fellépő hálózati kommunikáció a különböző erőforrásokon elhelyezett elemek között lassíthatja az adatok feldolgozását, ami megnövekedett feldolgozási sorokat eredményezhet, melyek a rendszer használhatatlanságához vezethetnek. Bár az ASM nyelv alkalmas ilyen jellegű problémák leírására, azonban a leképzés igen bonyolulttá válhat, vagyis nem mindig ez a hatékony megoldás. Ilyen esetekben a megoldásnak az adódik, ha a forrásmodellünket leképezzük egy másik szakterület-specifikus nyelvre, amelyben a probléma egyszerűbben leírható, és hatékonyabban megoldható.

A fenti megoldás technikája természetesen nem csak matematikai módszerekkel történő kiszámítás lehet, hanem az adott feladat szimulációja is. A szimuláció során ellenőrizhetjük a rendszer működését, és az általunk legjobbnak ítélt megoldást választhatjuk ki. Egy hatékony módszer, amikor a forrásmodellt Matlab-ra képezzük le, amely köré sok, a modell különböző vetületét vizsgáló rendszer építhető. A Matlab-ra épülő szimulációs eszközök (Simulink, TrueTime) a rendszer magas szintű leírását teszik lehetővé, de a beépített matematikai modelljeik segítségével a modelleket „végre tudjuk hajtani”. A Simulink használatával lehetőségünk nyílik a modellünk korai állapotában való kipróbálására. A szimuláció eredményeit a kiindulási modellbe visszavezetve – általában manuálisan – precízebben tudjuk folytatni a modellezést. Mivel az így kapott modell jobban megfelel a valóságban elvárthoz, így a végső termék is megbízhatóbb minőségű lesz.

5. Tesztkörnyezet komponens-alapú rendszerekhez

A 3. fejezetben bemutatott komponens rendszer tesztelése során az előző fejezetben bemutatott tesztelési eljárásokon túl új módszerek is megjelennek, melyek a komponens rendszer specifikusságát használják fel. A komponens rendszer fő tulajdonsága, hogy reflektív, így működését meghatározza a metamodellje. A rendszer telepítése után csak a metamodellben adott általános szabályoknak megfelelően rendezheti át a szerkezetét.

Ezen a ponton nagyon nagy hasonlóság figyelhető meg a GME modell-szerkesztő és a futó komponens rendszer működése között. A GME-ben modellezés során csak a modellhez tartozó metamodell szintaktikai és statikus szemantikai szabályait betartva építhetünk modelleket, és ezeknek a szabályoknak a betartásáról a modellező eszköz állandóan gondoskodik. A komponens rendszer esetén a futó kód metamodell szabályainak betartására a komponens futtató mag ügyel. A fenti hasonlóságnak a következménye, hogyha a futó rendszer változásait visszavezetjük a kódot létrehozó modellező eszközbe, akkor magában a modellező eszközben nyomon lehetne követni a futó programban történő változásokat. Ez azt eredményezi, hogy ha a változásokról a modellező eszköz értesül, és megjeleníti őket, akkor egy adott pillanatban nem lehet eldönteni, hogy



4. ábra
A deployment tool felépítése

a modell-szerkesztőben megjelenő modellt a modellező személy vagy a futó program változásai hozták létre. Kihasználva ezt az ötletet, az általunk létrehozott úgynevezett deployment toollal megvalósítható az alkalmazás modell létrehozásának, transzformálásának, telepítésének és felügyeletének egyetlen eszközbe való dinamikus egyesítése.

Az előző oldali, *4. ábrán* ábrán látható, hogy a modell alapú szoftverfejlesztés teljes életciklusa egyetlen eszközbe a deployment tool-lal kiterjesztett GME-be egyesíthető. A deployment tool alkalmazásával megszűnik az az általános szemlélet, mely élesen elválasztja az alkalmazás létrehozásának és futásidejű karbantartásának a feladatát. Természetesen a két feladat egy eszközben egyesítése nem jelenti azt, hogy a két feladat során az eszközben tárolt elemeknek is ugyanúgy kell megjelennie. Sőt felhasználva GME nyújtotta lehetőséget különböző nézetek kialakítására, akár eltérő részletességgel, és grafikai megjelenítéssel is használhatjuk az eszközt az egyes feladatokra.

A deployment tool felépítése a 4. ábrán látható. A rendszer 4 fő egységen alapul. A *GME connector,* mely a futó komponens rendszerben történt változásokat a GME felé továbbítja. A *GME affector,* mely a GME-ben létrehozott változásokat a futó rendszeren érvényesíti. Valamint a GME-ben az előző két funkció fogadó egysége, a *DeploymentIn,* amely a futó rendszer változásait a modell-szerkesztőben megjeleníti, és a *DeplotmentOut,* amely a modell-szerkesztő által létrehozott változásokat a futó rendszernek elküldi.

Fontos megjegyezni, hogy a feladatok ezen csoportosítása által a GME-nek nem szükséges az adott rendszert futtató hardver környezetben futnia – az esetek többségében erre nem is lenne lehetőség –, hanem a rendszer változásai hálózaton keresztül is eljuthatnak a GME-hez. Így távoli felügyelő eszközként is használható a GME. Azonban, mivel a rendszer és a modell-szerkesztő egyszerre fut, így olyan változások is kiválthatóak, melyek ellentétes hatásúak, ilyenkor mindig a modell-szerkesztő által létrehozott változások a nagyobb prioritásúak.

Mivel a rendszer aktuális állapotát a modell-szerkesztő tárolja, így az adott modell elmentésével egy pillanatképet készíthetünk a rendszer konfigurációjáról, melyet különböző célokra lehet felhasználni. A pillanatkép legfontosabb felhasználási területe a rendszer tesztelése. A tesztelés során a teszter egy ilyen pillanatfelvételt kap a rendszerről, ahonnan elkezdheti a tesztelést, mivel a rendszer állapota az aktuális pillanatkép alapján visszaállítható.

A tesztelés tovább könnyíthető, ha a tesztelést nem valódi hardver platformon végezzük, hanem szimulált hardver környezetben. A szimulált platformnak előnyei, hogy nem kell drága speciális hardver erőforrással rendelkeznünk a rendszer futtatásához, hanem kiválthatjuk azt olcsóbb elemekkel, amelyeken a szimuláció azonosképp elvégezhető. Ezen kívül bizonyos szimulátorok egyéb előnyöket is nyújtanak, melyek jól párosíthatóak a modellezés előnyeivel. Például a Simics [6] szimulációs eszközt használva lehetőség nyílik a szimulált idő visszapörgetéséhez. Ezt a folyamatot szintén visszsza lehet vezetni a modell-szerkesztőbe, hogy így a modell szintjén is lehetőség nyíljon az időben előre- és hátralépésre.

6. Összefoglalás

A cikkünkben bemutattuk, hogy miképp nyújthat segítséget a szakterület specifikus modellezés összetett rendszerek fejlesztése során. A szakterület-specifikus modellek legjelentősebb hatása abban áll, hogy támogatást nyújt az adott terület szakértőinek, hogy egyértelműen és gyorsan tudják megfogalmazni az adott probléma megoldásait, és ezáltal sokkal hatékonyabbá teszi a szakterületen belüli fejlesztéseket. Több szakterületet átfogó rendszerek esetén a modellek egymásba történő automatikus átalakítása jelentősen támogatja a probléma egyre részletesebb szintű kidolgozását.

Mivel a teljes módszer formális alapokon nyugszik, így lehetőségünk nyílik különböző ellenőrzési módszerek bevezetésére. Sajnos egy általános, mindent átfogó ellenőrzési módszer nem létezik, de a rendszert különböző nézetek szerint vizsgáló módszerek jól integrálhatóak egymásba. A különböző ellenőrzési módszerek együttes alkalmazása során a végső termék sokkal megbízhatóbbá válik.

Irodalom

[1] G. Karsai, J. Sztipanovits, A. Ledeczi, T. Bapty, “Model-Integrated Development of Embedded Software”, Proceedings of the IEEE, Vol. 91, pp.145–164, January 2003.
[2] RUNES IST Project, http://www.ist-runes.org/
[3] G. Bátori, Z. Theisz, D. Asztalos, “Robust Reconfigurable Erlang Component System”, Erlang User Conference, Stockholm, Sweden, 2005.
[4] K. Chen, J. Sztipanovits, S. Abdelwalhed, E. Jackson, “Semantic Anchoring with Model Transformations”, ECMDA-FA 2005, LNCS 3748, pp.115–129.
[5] E. Boerger, R. Staerk, Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, 2003.
[6] Peter S. Magnusson et al, “Simics: A Full System Simulation Platform”, IEEE Computer, February 2002.