Az UML2 és a modell-vezérelt alkalmazásfejlesztés

Papp Ágnes, *agi@delfin.unideb.hu*
Debreceni Egyetem EFK

A vállalati alkalmazások fejlesztése manapság olyan megközelítést igényel, amely flexibilis módon teszi lehetővé a tervezők és fejlesztők számára a megoldások kidolgozását. Ez a megközelítés megengedi a már meglévő eredmények újrafelhasználását, az infrastruktúra változása mellett, amelyben az alkalmazást implementálták. Egy elosztott vállalati környezetben egy újonnan fejlesztett alkalmazásnak a különböző típusú meglévő rendszerekkel is kapcsolatot kell tartania. Olyan szabványokra és keretrendszerre van szükség, amely biztosítja a különböző platformokon fejlesztett alkalmazások közötti együttműködést. Az Object Management Group (OMG) [1] hozott létre egy olyan koncepcionális keretrendszert, amely elkülöníti egymástól az üzlet-orientált döntéseket a platform-függő döntésektől, nagyobb teret hagyva a tervezőknek a rendszerek kidolgozásakor.

A Modell-vezérelt Architektúra

A Modell-vezérelt Architektúra (Model Driven Architecture - MDA) [2] egy új alkalmazásfejlesztési megközelítés. Egy MDA specifikáció egy platform-független UML [4] modellből (Platform Independent Model - PIM) és egy vagy több platform specifikus modellből (Platform Specific Model - PSM) áll. Az MDA szemlélet szerint először a vizsgált rendszer funkcionalitására és viselkedésére kell koncentrálni, eltekintve a technológiai környezettől, amelyben implementálásra kerül. Ily módon egy alkalmazási rendszer teljes modelljét csak egyszer kell megalkotni.

Az MDA elkülöníti az alkalmazás-architektúrát a rendszer-architektúrától [3]. Az alkalmazás-architektúra az alkalmazás funkcionális céljait specifikáló komponenseket és kapcsolatokat tartalmazza. A rendszer-architektúra pedig az alacsonyabb szintű komponensekből és kapcsolatokból áll, amelyek az alkalmazás-architektúra végrehajtását teszik lehetővé. Ez az elkülönítés az MDA fundamentális eleme.

Egy PIM tulajdonképpen teljes alkalmazás specifikáció, amely platform-független. Egy PIM egy PMS-be képződik le, amely a rendszer-architektúra infrastruktúráját biztosítja. Ez a leképezés a PIM implementációját jelenti, azaz a végrehajtható alkalmazás generálását. A PIM lehetővé teszi számunkra a megoldás vizuális modellezését, magas absztrakciós szinten. Szükségtelen az alkalmazás újraírása, amikor egy új technológia megjelenik, csak újra kell generálni az alkalmazást, azaz elvégezni a modell leképezését az új környezetbe.

Sok népszerű technológiai platform, mint a CORBA, J2EE vagy .NET számára lehetséges PIM→PSM leképezést definiálni. Az OMG szabványai, a MOF (Meta Object Facility), XML (Extensible Markup Language), XMI (XML Metadata Interchange) és CWM (Common

Warehouse Metamodel) együttműködése biztosítja, hogy az MDA teljes szoftverfejlesztési megközelítés legyen.

Executable UML

Szükség van a műveletek specifikálására az UML modellben, amely a végrehajtható UML (Executable UML) megvalósíthatóságát támogatja.

Az UML osztály diagramokból és állapot modellekből lehet adatszerkezeteket és control flow-t generálni, de nem volt szabványos mód az objektumok alacsony szintű viselkedésének leírására. A „Precise Action Semantics for the UML" szabvány biztosítja a műveletek szemantikájának leírását, amelyek az UML modell viselkedésének specifikálásához kellenek. A modellnek olyan részletesnek kell lennie, hogy a teljes végrehajtható alkalmazás generálható legyen belőle.

A modellező egy platform-független, szöveg-alapú nyelv használatával adja meg a szükséges műveleteket. Így egy UML modell viselkedésének teljes specifikációjához a szemantikus szabványnak megfelelő műveleti nyelvre van szükség.

Modell compiler-ek

Hogy a modellezett alkalmazás vagy PIM végrehajtható legyen, a PSM architekturális szolgáltatásaitól függ. A leképezés folyamán minden UML elemhez a PSM valamely eleme lesz megfeleltetve, amely aztán a kiválasztott végrehajtási környezet része lesz.

Az UML 1.x

Az UML (Unified Modeling Language) [4] Egységesített modellező nyelvet jelent. Grafikus nyelv, mely alapvetően szoftverrendszer elemek vizualizálására, specifikálására, létrehozására és dokumentálására szolgál. Az objektum-orientált szemléletre épülő elemzés és tervezés eszközeként indult. Több, korábbi módszertan és jelölésrendszer egyesítésével jött létre, majd vált szabvánnyá. Az 1997-es szabvánnyá válást követően több módosítást végeztek rajta.

Az UML specifikáció tartalmazza többek között:
- Jelölés (Notation). Definiálja a szintaxist, a jelölésrendszer elemeit.
- Szemantika (Semantics). Leírja az egyes jelölések jelentését.

Az UML 1.4 szerint kilenc különböző típusú diagram áll rendelkezésre valamely rendszer szerkezetének és viselkedésének leírásához. A használati eset diagram a rendszerrel szemben támasztott követelményeket írja le, az osztály diagramok és objektum diagramok a rendszer statikus struktúráját, a komponens diagramok és a telepítési diagramok pedig az implementációs architektúrát ábrázolják. Az együttműködési diagramok, szekvencia diagramok, állapot-átmeneti diagramok és aktivitás diagramok a rendszer viselkedésének különböző aspektusait specifikálják.

Az UML-t sok kritika éri, mind általános, mind domain-specifikus hibákat illetve hiányosságokat említve [5].

Az UML általános gyengeségei

Ami a specifikációt illeti, nem formális, a nyelv szintaktikai elemeinek szemantikája gyakran nincs precízen definiálva. Így a vendorok különböző módon implementálják az UML egyes részeit az eszközeikben. Kommunikációs problémához vezethet egy fejlesztő csoport tagja között is, ha nem világos, hogyan kell egy diagramot interpretálni, és egy modellen belüli UML diagramok konzisztenciáját is bonyolult ellenőrizni.

Az UML az OMG négy rétegű metamodellezési megközelítésén alapszik, amelyben a MOF szolgál metamodellként. Sajnos ez a megközelítés nem lett szigorúan végig követve, ami azt jelentené, hogy egy réteg minden eleme kizárólag csak a fölötte lévő rétegtől függ.

Az UML 1.4 egyik leggyakrabban emlegetett hibája a használhatósága, mivel túl sok különböző típusú diagramot és elemet tartalmaz. Nehéz kitalálni, hogy melyik kombináció a legalkalmasabb egy feladat megoldására. Továbbá, a diagramok a rendszerek különböző nézeteit (tervezési, implementációs, stb.) és különböző aspektusait (szerkezet, viselkedés, stb.) reprezentálják, de nincs mechanizmus, amely a rendszert leíró diagramok közötti kapcsolatot definiálná.

A jelen verzióban nincs lehetőség a modellek hierarchikus strukturálására, minden elem ugyanazon a szinten jelenik meg. Ez nem okoz gondot kisebb rendszereknél, de a nagyobb rendszereket így nehezebb megérteni. Az a modell, amely hierarchikus, kevésbé bonyolult, mert el lehet rejteni a részleteket, amikor nincs rájuk szükség. A komponensek és az alrendszerek nem lettek egyértelműen implementálva, és a szintenkénti kompozíció egyáltalán nem lehetséges.

És végül, meg kell említeni, hogy a szoftverrendszerek egy része nem a megszokott használatot, hanem a előforduló hibákat kezeli. Ez a tény vezetett az explicit hibakezelés támogatásának igényéhez az UML-ben.

Az UML2

Az UML továbbfejlesztett változata, az UML 2.0 az OMG által végzett véglegesítés folyamatában van.

Négy különböző dokumentum létezik az UML 2.0 specifikálására:

- UML2 Infrastructure
- UML2 Superstructure
- UML2 Object Control Language (OCL)
- UML2 Diagram Interchange (XMI)

A Superstructure RFP tart igényt a legnagyobb érdeklődésre, mert ez tartalmazza a felhasználók számára a leglátványosabb részt, a modellkészítéshez szükséges elemeket. Két fő szempont vezérelte a követelmények megfogalmazását: a skálázhatóság és az architektúra [6]. Az architektúrára vonatkozó változások első sorban a szerkezet modellezésében érvényesülnek, míg a skálázhatóságra vonatkozó változások a továbbfejlesztett szekvencia diagramok esetében láthatók leginkább.

A továbbfejlesztett UML2 jellemzői:

- A komponens-alapú szoftverfejlesztés támogatása
- Szoftver-architektúra modellezésének támogatása
- Több lehetőség a szimulációval és kódgenerálással történő fejlesztésre
- Végrehajtható modellek és a dinamikus viselkedés támogatása
- Diagramok cseréjének specifikálása az eszközök között
- Skálázhatóság
- Kiterjesztési mechanizmus – UML Profile-ok

A szerkezet modellezése

Talán a legfontosabb fejlesztés az UML2-ben, a komplex rendszerekre való tekintettel, az architektúra-modellezési koncepció támogatása. E szerint egy osztály (Class, Component, Collaboration) más osztályok egyedeinek kompozíciójaként állhat elő, azaz belső szerkezettel rendelkezhet. A belső szerkezetet leíró elemek : Part, Connector, Port [5].

A tartalmazó osztály belső szerkezetét alkotó elem (Part), egy másik osztály előfordulása vagy előfordulásainak halmaza. A belső elem az őt tartalmazó elemhez tartozik és nem is létezhet nélküle.

A portok (Port) kapcsolódási pontokat írnak le az osztályok határán. Arra szolgálnak, hogy a belső részek kapcsolatba kerülhessenek a külvilággal, és a tartalmazó osztályok egymással. Egy port címezhető, ami azt jelenti, hogy jelek küldhetők rá. Egy portnak kétféle interfésze lehet, az egyik az osztály által szolgáltatott, a másik a környezettől igényelt műveleteket és jeleket definiálja.

Az úgynevezett konnektorok (Connector) létesítenek kapcsolatot (asszociáció egy előfordulása) a részek között, a kommunikációt lehetővé téve, azaz üzeneteket lehet küldeni és fogadni. Az asszociációkkal ellentétben, amelyek osztályok közötti kapcsolatot jelentenek, a konnektorok csak a belső szerkezet részei közötti kapcsolatot specifikálják. Egy konnektor egy porthoz vagy közvetlenül az osztály valamely részéhez csatlakozhat.

A fent részletezett módon egy modell különböző absztrakciós szinteken írható le, mivel az elemek egymásba ágyazhatók. A belső részek elrejtésével a rendszer jobban áttekinthető, míg megjelenítve őket, részletes információt kapunk. A portok használata lehetőséget ad arra, hogy elrejtse egy osztály belső szerkezetét a környezet elől, és továbbítsa a beérkező vagy kimenő jeleket.

Mindez fontos a komponens-alapú szoftverfejlesztés támogatásának szempontjából. A komponensek így már szoftver-komponensként kezelhetők, ami azt jelenti, hogy egy komponens egy helyettesíthető, telepíthető része a szoftvernek, amely elérhető a specifikáció alatt, telepítéskor és futásidőben. Egy komponensnek megvan a maga belső szerkezete, és a környezetével kizárólag interfészeken vagy portokon keresztül lép kapcsolatba.

Szerkezeti diagramok áttekintése:

- Osztály diagram (Class Diagram), Objektum diagram (Object Diagram): a fent leírt új modellelemeket tartalmazzák.
- Telepítési diagram (Deployment Diagram): ez az egyetlen diagram, amely az implementációs környezet elemeivel foglalkozik. Némileg különbözik a korábbitól, mivel új modellelemek jelentek meg a diagramban.

- Csomag diagram (Package Diagram), Komponens diagram (Component Diagram): hasonlóan az osztálydiagramhoz, csomagok és komponensek ábrázolására szolgálnak.
- Composite Structure Diagram: a belső szerkezet ábrázolására szolgáló új diagramtípus, az osztályok, komponensek hiearchikus kompozícióját mutatja be.

A viselkedés modellezése

A viselkedés modellezésének lehetőségei is kiterjedtek az UML 2.0-ában, amennyiben sokkal részletesebb modellek alkothatók [5].

A szekvencia diagramok változásai leginkább a skálázhatóságot célozzák meg. Az objektumok közötti együttműködés ábrázolása ugyanúgy életvonalakkal, üzenetekkel történik, de van néhány eltérés. A diagramok ún. interakció fragment-ekre (Iteraction Fragment) bonthatók szét, amelyek vagy ugyanabban, vagy más diagramban reprezentáltak. A diagram egy másik, beágyazott diagramot azaz fragment-et tartalmazhat. A különböző operátorok jelzik, hogy a beágyazott fragment alternatívát (if/then/else), másikra való hivatkozást, stb. jelent. Az életvonalak is dekomponálhatók (Lifeline Decomposition) egy másik diagramban. Az üzenetek belépési és kilépési pontjai biztosítják, hogy a diagramok konzisztensek legyenek.

Az új akciómodell részben a korábbi Action Semantics Profile for UML 1.4-hez lett igazítva. 20 különböző akciótípus van már a korábbi 7 helyett a finomított modellben, amelyek 3 csoportba lettek sorolva: invocation, read/write, computational actions. Az akciók a viselkedés atomi, tovább nem bonható elemei. A viselkedés modellezésében azonban a tevékenységek játszanak fontos szerepet, ahogyan az a továbbra is használható aktivitás diagramokból ismert.

Interakciók folyamát lehet modellezni az új Interakció áttekintés diagrammal, amely tulajdonképpen egy speciális aktivitás diagram, ahol az aktivitások mind interakciók.

Új, ám igen egyszerű az UML-ben az időmodell, amelyet időtriggerek, időtartamok és időpontok reprezentálására, specifikálására és figyelésére vezettek be.

Az állapotgépek lehetnek viselkedési vagy protokol állapotgépek. Az első típus a rendszer egy részének események által vezérelt viselkedését modellezi. A második típus protokolok definiálására szolgál, absztrakt viselkedést ír le és portokkal vagy interfészekkel van kapcsolatban. A viselkedési állapotgépek vezérlő események lehetnek jelek, időtúllépések, művelethívások és értékváltozások, amelyek állapotátmeneteket okozhatnak. Az átmeneteket akciók írják le, amelyek újabb eseményeket generálhatnak. Az állapotgépekkel kapcsolatos legfontosabb fejlesztések egyike a viselkedés bezárása egy composit állapotba (composite state). A hozzáférést névvel ellátott belépési és kilépési pontok kontrolálják, amelyek az állapotgép határán helyezkednek el. A viselkedés újrafelhasználható komponensei tervezhetők így meg. A másik újítás az állapotgépek esetében a viselkedés öröklődése. Az általánosítás/pontosítás alapvető koncepció az UML-ben. Ugyanúgy, mint az osztályok esetében, új állapotgépek specifikálhatók a kiterjesztés vagy pontosítás által. A viselkedés specifikálásának új elemei: állapotok és átmenetek hozzáadása, állapotok kiterjesztése, átmenetek lecserélése, kiterjesztése, stb. A protokol állapotgépnek a viselkedést leírókhoz képest korlátjai vannak: nincsenek belépési és kilépési akciói, tevékenységei, belső átmenetei, stb. A célja, hogy szolgáltatásokat specifikáljon interfészek számára.

A viselkedést leíró diagramok áttekintése:

- Szekvencia diagram (Sequence Diagram)
- Kommunikáció diagram (Communication Diagram) – a korábbi együttműködési diag.
- Aktivitás diagram (Activity Diagram)
- Interakció áttekintés (Interaction Overview Diagram)
- Állapotgép diagram (Statemachine Diagram)
- Idő diagram (Timing Diagram)
- Használati eset diagram (Use Case Diagram)

UML Profile-ok

A profile-ok arra szolgálnak, hogy UML-t ki lehessen egészíteni domain-specifikus elemekkel, azért hogy a nyelv ne legyen túlterhelve ritkán használt lehetőségekkel. A profile-ok kiterjesztési mechanizmust biztosítanak az UML számára. Egy UML profile-nak három kulcseleme van: sztereotípiák, címke-érték párok (tulajdonságok) és megszorítások. Egy profile megadja a definícióját ezeknek az elemeknek, és elmagyarázza, hogyan bővítik ki az UML használhatóságát egy meghatározott területen. Pl. Profile for Schedulability, Performance and Time, Profile for Action Semantics. Egy profile fejlesztésére és alkalmazására a modellezésben részletes leírás található az UML 2.0-ban.

Az UML 2.0 gyengeségei

A nyelv továbbra sem rendelkezik formális definícióval, még a core elemek esetében sem, pedig ez volt az UML 1.4-ben a leginkább kritizált pont. Az UML használhatósága nem nőtt abban az értelemben, hogy több új elem lett bevezetve és újabb diagramok jelentek meg.

A diagramok között sokszor átfedés figyelhető meg. Az objektum diagram például ugyanazt ábrázolja, mint a kommunikációs diagram, kivéve az üzenetek áramlását. Az interakció áttekintés diagram pedig az aktivitás diagram egy speciális esete. A modellt alkotó diagramok közötti függőség nincs explicit módon definiálva, a felelősséget a modellezőre vagy az eszközre hagyva, hogy konzisztens modell jöjjön létre. Így viszont a modell tesztelése válik nehézkessé.

A modellvezérelt alkalmazásfejlesztés közelebb viszi a megoldás specifikálását a probléma megfogalmazásához. Az UML célja pedig az, hogy a következő lépcsőfok legyen a szoftverfejlesztési technikák között.

Irodalom

[1] Object Management Group http://www.omg.org
[2] OMG Model Driven Architecture http://www.omg.org/mda
[3] Executable UML: Diagrams for the Future http://www.devx.com/enterprise/Article/10717
[4] Unified Modeling Language http://www.omg.org/technology/documents/formal/uml.htm
[5] Using UML 2.0 in Real-Time Development by Kristen Berkenkötter
http://eai.ittoolbox.com/documents/document.asp?i=3057
[6] UML 2.0 Incrementally Improves Scalability and Architecture
http://www.elecdesign.com/Articles/Index.cfm?ArticleID=5881