Új programozási paradigmák a láthatáron
(aspektus-orientált és intencionális programozás)

Csontos Péter

AITIA Informatikai Rt.

Bevezetés

A 70-es években az ún. Szoftver Krízis alapjaiban változtatta meg a számítógép-programozás megközelítését és módszereit, elterjesztve a struktúrált, moduláris, majd objektum-orientált programozást.

Ugyanígy napjainkban is egyre inkább érezhető az a hatás, mely szerint a szoftverek fejlesztői nem képesek a megfelelő tempóban és minőségben felvenni a versenyt egyrészt a Moore-törvénnyel, másrészt az informatikai eszközök egyre szélesebb körű elterjedésének sebességével.

Az OOP-n túl

Az objektum-orientált programozás annak idején mondhatjuk, hogy a lehető legjobb választ adta az áttekinthető és ily módon biztonságoságosabb programok írására vonatkozó igényre, valamint a kódújrafelhasználhatóság követelményére, ez okozta széleskörű elterjedését és de-facto egyeduralmát. Mindezt tovább fokozta a modellezésben végbement szabványosodási folyamat az UML révén.

Általános módszerként és programozási nyelv-orientált megközelítésként az OOP és az UML nagyrészt azonban nem oldott meg két problémát.

Egyrészt, mivel csupán a szoftver-modellezést absztrahálja a megfelelő módon, eredeti formájában csak csekély mértékben alkalmas arra, hogy az egyes alkalmazási területek és szoftverfejlesztési aspektusok (az üzleti intelligenciától a 3 dimenziós grafikai modellezésig és a távközlési protokollok vezérlésétől a banki számlavezetésig, a loggolástól a biztonsági mechanizmusok kezeléséig, stb.) absztrakcióját megoldja, szabványosításukra módot adjon.

Másrészt, az UML alapú round-trip engineering megoldások csak részben alkalmasak az öröklött rendszerek integrációjára és a különböző platformokra és nyelvekre épülő összetett szoftverrendszerek egységes tervezésére és generálására.

Új irányok

Az első problémára igyekszik megoldást találni az aspektus-orientált programozás, általánosabban fogalmazva a vonatkozások szétválasztása (angolul separation of concerns), valamint a meta-metamodellező megközelítések.

Ezen túl a második problémára is igyekszenek gyógyírt adni az intencionális (szándék alapú, szándék-orientált stb.) programozás és társai (van belőlük néhány).

Aspektus-orientált programozás

Az aspektus-orientált programozás egy olyan programozási paradigma, amely a kilencvenes évek közepén született meg. Manapság a programozási nyelvekkel kapcsolatos kuatások középpontjában áll, és minden bizonnyal a közeljövõben széleskörûen el fog terjedni.

Az aspektusok olyan funkcionálisan összefüggõ programrészletek gyûjteménye, amelyek egy, az AOP lehetõségeket nélkülözõ nyelven írt programban a forráskód különbözõ részein szétszórtan találhatók meg. A paradigma célja, hogy ezen funkcionálisan összefüggõ részek egy helyre gyûjtésével a programtermék (kód és/vagy terv) bonyolultságát csökkentse, a karbantartható7ságát, olvashatóságát javítsa.

A lényeg, ami minden aspektus-orientált megközelítésben közös, hogy az osztályok mellett új modularizációs elemeket vezetnek be (aspect, subject stb.), melynek feladata, hogy az osztályoktól külön, azokból kivonva valósítsa meg az egyes aspektusok funkcionalitását.

Számos aspektus-orientált módszer és eszköz létezik, ezek közül a legismertebbek az AspectJ és HyperJ, melyekre külön kitérünk.

AspectJ

Az AspectJ a Java nyelv természetes kiterjesztése: minden Java program egyben AspectJ program is, az AspectJ által elõállított programoknak minden JVM-en képesek futni, a Java fejlesztõeszközök természetes módon kiterjeszthetõk AspectJ fejlesztõeszközökké, valamint az AspectJ szintaxisa olyan módon terjeszti ki a Java-t, hogy a Java programozók a nyelv természetes részének tekintsék a kiterjesztéseket.

Az AspectJ-t [6] a XEROX PARC (Palo-Alto Research Center) fejlesztette és fejleszti, a Gregor Kiczales professzor vezetésével.

Teljes leírást nem célom e helyt adni, csupán érzékeltetni szeretném az AspectJ megközelítését, sajátos aspektus-orientált felfogását.

Az AspectJ elemei és fõbb tulajdonságai

A nyelv fõ elemei: dinamikus kapcsolódási pontok (dynamic join point), eseményleíró kifejezések (pointcut) és tanácsok (advice), amelyek aspektusokká (aspect) kombinálhatók.

A dinamikus kapcsolódási pontok a programfutás olyan eseményeihez kötõdnek, mint metódus hívás, metódus hívás fogadása, metódus futása, attribútum lekérdezés, kivétel dobása, osztály inicializáció, objektum inicializáció, stb.

Az eseményleíró kifejezések a dinamikus kapcsolódási pontok halmazai, illetve ezekbõl meghatározott mûveletekkel képzett halmazok.

A tanácsok olyan metódus-szerû mûveletek, melyek egy eseményleíró kifejezéshez rendelt (elõtte, utána vagy esemény közben végrehajtott) tevékenységeket írják le.

Az aspektus olyan moduláris egység, amely a fenti három elem, valamint bármely osztályon belül elképzelhetõ tag tetszõleges kombinációját tartalmazhatja.

Annak érdekében, hogy futás közben egyértelmûen eldönthetõ legyen, hogy egy belépési pontra kapcsolódó több aspektus elemei milyen sorrendben futhatnak, az aspektusok között és az aspektusokon belül jól definiált precedencia relációk állnak fenn.

Az aspektusok között öröklõdési reláció állhat fenn.

Fejlesztőeszközök és más nyelvek

Az AspectJ-hez létezik fejlesztőeszköz-integráció a Jbuilder-hez és az emacs-hez is, készítői folyamatosan dolgoznak a többi eszközhöz való integráción.

Az AspectJ-re alapozva más nyelvekhez is születtek kiterjesztések, pl. C, C++, C#, Perl, Python, Ruby, Smalltalk.

HyperJ

A HyperJ [1] [6] az IBM Research terméke, az aspektus-orientált programozásnak az AspectJ-től jelentősen eltérő megközelítését adja. Történetileg a szintén az IBM által kifejlesztett szubjektum-orientált programozás továbbfejlesztése.

Szubjektum-orientált programozás (SOP)

Az SOP lényege, hogy az osztályokban található metódusokat (alapvetően C++-ra fejlesztették) szétválogathatjuk, és egy specifikációs nyelv segítségével szubjektumokat (aspektusokat) állíthatunk össze belőlük.

Vonatkozások többdimenziós szétválasztása (MDSOC)

Az MDSOC, más néven hiperterek (hyperspaces) vagy vonatkozás terek (concern spaces) lényege, hogy az aspektusokat egyszerre több dimenzió mentén választhassuk szét.

A HyperJ az MDSOC Java nyelvű megvalósítása. Az SOP-hez hasonlóan az aspektusok nem hozzáadják a funkcionalitást az osztályokhoz, hanem a class-ként megírt összefüggő kódból kiszeletelik az aspektusokat, majd ezeket komponáltan jelenítik meg.

Egyéb megközelítések

Számos egyéb eszköz és módszer [1] született az aspektus-orientált paradigma részeként, a teljesség igénye nélkül: DemeterJ, Mozart, ComposeJ, ConcernJ, JADE.

Intencionális programozás

Az intencionális programozás [2] [11] [12] a Microsoft Research kutatási projektjeként indult a 90-es évek elején Charles Simonyi vezetésével, aki 1981 óta vezető fejlesztőként dolgozott a cégnél.

Az új paradigma célja egyrészt a programterv és a kód közötti éles határok eltüntetése, valamint az, hogy tetszőleges szak- és részterület absztrakcióit hatékonyan le lehessen vele írni, és az absztrakciókat rugalmasan és hatékonyan kombinálni lehessen.

Áttekintés

A módszer lényege röviden a következő: van egy szintaxisfa-szerű ún. Intencionális program-fa, amelynek csúcsai generikusak, egymással tetszőleges módon példányosíthatók (paraméterezhetők). Emiatt az intencionális programozást szokás a generatív programozás [5] egyik alfajának tekinteni.

Ezzel a fával – a megfelelő szerkesztőprogram segítségével – tetszőleges absztrakció és implementáció leírható, majd a megfelelő generátor transzformációk segítségével (amelyek szintén a fa részét képezik)

előáll az úgynevezett csökkentett fa (R-code), amely már atomi utasításokat tartalmaz (talán hasonlítható a Java byte-code-jához). Ebből aztán a megfelelő platformfüggő fordítóprogram vagy interpreter lefordíthatja és végrehajthatóvá teheti a programot.

Az IP alkalmas arra, hogy öröklött programokat kielemezve intencionális programfává alakítsa, és ilymódon a továbbfejlesztés viszonylag zökkenőmentesen oldható meg, nincs szükség arra, hogy mindent újraírjunk.

Feltűnően sok pszichológiai és biológiai párhuzamot találhatunk az IP körül. Már maga az intencionális programozás is a kognitív pszichoilógiából származik, azt hivatott megfogalmazni, hogy az absztrakció és a reprezentáció ideális szimbiózisának következtében kevésbé tűnik el a program megálmodójának eredeti szándéka a kódolás során, mint a hagyományos módszerek esetében.

Egyéb analógiák: az R-kódot előállító transzformációkat redukciós enzimeknek hívják, az intencionális programok "társadalmát" pedig az absztrakciók ökológiájának.

Hol tart most az IP?

Tavaly a Microsoft leállította az IP projektet, nemrégiben pedig Charles Simonyi és Greg Kiczales megalakították az Intentional Software Corporation-t, mely többek között az ELTE-vel való szoros együttműködésben termékké formálja az IP ötletét, várhatóan 2003 végére.

Kiczales részvétele a cégben jelzi, hogy elsősorban az AOP híveit célozzák meg az új paradigmával, mivel az IP alkalmas az aspektus-orientáltság leírására (sokminden egyéb, például tervezési minták mellett).

További ígéretes kezdeményezések

Az alábbiakban a teljesség igénye nélkül felvázolok néhány kezdeményezést, melyek a fentiekhez hasonló célokból születtek.

Eclipse Universal Modeler

Az IBM Research kutatóinak célja, hogy az Eclipse [4] nevű univerzális fejlesztőeszközre épülve (melynek fejlesztésébe nemrég bekapcsolódott a HP is) olyan eszközt adjon, mellyel tetszőleges absztrakciós mechanizmus és jelölésrendszer leírható.

Az eszköz szabványos protokollokat és alapinfrastruktúrát használ (XMI, MOF) a metaadatok leírására.

Jackpot

Ez a projekt [10] az EUM-hez hasonlóan univerzális modellezőeszköz kifejlesztését célozza, a NetBeans fejlesztőeszköz alapjain.

A projekt vezetője a Sun Microsystemsnél James Gosling, a Java egyik megalkotója, aki korábban az emacs kifejlesztésében is részt vett.

Az UML fejlődése

Az UML is fejlődik lassan de biztosan abba az irányba, hogy egyre általánosabban lehessen vele absztrahálni.

Ilyen kezdeményezések az UML akció szemantika, egy teljes UML-alapú szoftver platform kifejlesztésének terve (Ivar Jacobson szavaival élve: néhány év múlva mindenki UML-ben fog programozni, a mostani

programnyelvek nélkül) [8]. Ez kísértetiesen hasonlít az IP célkitűzésére. Szintén az IP-re rímel a The Reuse Initiative, melynek célja, hogy a meglévő programok újra felhasználhatóságát segítse.

Szintén Ivar Jacobson eredménye a JacZone [9] cég WayPointer nevű terméke [3], amely intelligens ágenseket [7] alkalmaz újszerű módon, melyek a Rational Rose és a RUP alkalmazásában állnak a fejlesztők segítségére.

Irodalomjegyzék

[1] Aspect-Oriented Software Development. http://www.aosd.net/

[2] Charles Simonyi: Intentional Programming – Innovation in the Legacy Age. IFIP WG meeting, 1996. http://web.archive.org/web/20010812092805/www.research.microsoft.com/ip/ifipwg/ifipwg.htm

[3] CyberIvar. http://213.67.11.109:80/?template=chat

[4] Eclipse. http://www.eclipse.org/

[5] Generative Programming. http://143.93.17.150/~gporg/

[6] Gregor Kiczales, Erik Hilsdale, Jim Hungunin, Mik Kersten, Jeffrey Palm and William G. Griswold: An Overview of AspectJ. Jorgen Lindskov Knudsen (Ed.): *Proceedings of the 15th ECOOP*, Budapest, 2001, pp. 327-353.

[7] Gulyás László, Tatai Gábor: Ágensek és multi-ágens rendszerek. Futó Iván (Szerk.): *Mesterséges intelligencia*, Aula kiadó, Budapest, 1999, pp. 709-754.

[8] Ivar Jacobson: *Trends in Software Engineering*. UML 2000, York, UK, http://www.cs.york.ac.uk/uml2000/jacobson.ppt

[9] JacZone. http://www.jaczone.com/

[10] Matt Berger: JavaOne: Gosling hits 'Jackpot' with futuristic tools. IDG News Service, 2002. http://www.infoworld.com/articles/hn/xml/02/03/20/020320hngosling.xml

[11] Microsoft Research (Web Archive, Wayback Machine): Intentional Programming Homepage. http://web.archive.org/web/20010801191822/http://www.research.microsoft.com/ip/

[12] OMNISCIUM – Intentional Programming. http://www.omniscium.com/nerdy/ip/