Designing and synthesizing the synchronization of concurrent object-oriented systems

Doktori értekezés tézisei

Hajdara Szabolcs

Témavezető:
Kozma László, C.Sc.

Az Informatika alapjai és módszerei doktori program
Informatika Doktori Iskola
Prof. Demetrovics János

Eötvös Loránd Tudományegyetem
Informatika Kar
Programozáselmélet és Szoftvertechnológiai Tanszék

Budapest, 2007

Támogatta: GVOP-3.2.2-2004-07-005/3.0.

1. Célkitűzések

Az objektum-elvű módon megtervezett és elkészített programok ([21], [23], stb.) sok előnnyel rendelkeznek, hatékonyan karbantarthatók és könnyen újrafelhasználhatók. A hardvereszközök gyors fejlődésének következtében programjainkat egyre többször futtatjuk több processzorral rendelkező hardverkörnyezetben, vagy akár több összekapcsolt számítógépen. Az objektum-elvű szoftvertervezésben így nagy jelentőséget kapott az objektumok párhuzamos – esetleg elosztott [19], [25], [26] – működésének, illetve szinkronizációjának megtervezése, implementálása [16], [18], [20], stb.

Általában a programok szinkronizációs kódja és kiszámítási kódja szétválasztható [14], ezért ezt feltételezve csak programok szinkronizációs kódjával foglalkozunk. A konkurens módon futtatott, egymással kommunikáló objektumok használata nagy mértékben megnöveli a programok bonyolultságát, áttekinthetetlenségét, megnehezíti a szinkronizációs kód elkészítését.

Az értekezés célja egy olyan módszer létrehozása, mellyel adott specifikáció szerint helyes szinkronizációs kóddal rendelkező objektum-orientált programok állíthatók elő a szinkronizációs kód automatikus generálásával, valamint megfelelő eszközöket szolgáltat a szinkronizációs kód, és kiszámítási kód helyes összekapcsolására a biztonságosabb és gyorsabb szoftveralkotás érdekében. A dolgozatban törekszünk a megoldható feladatok körének tágítására és a módszerek használhatóságának növelésére. Szem előtt tartjuk továbbá, hogy az előállított programkód használható, átlátható és hatékony legyen, valamint módszert adunk a hatékonyság javítására. Megvizsgáljuk, milyen módon választható külön a szinkronizációs kód strukturális szinten, és milyen körülmények között válik lehetővé a szinkronizációs kód újrafelhasználása. Az értekezésben csak az osztály alapú objektum-orientált nyelvek körét vizsgáljuk.

Kutatásaimat Ugron Balázzsal közösen végeztem, eredményeinket együtt publikáltuk. Az alapok kidolgozása után Ugron Balázs a pipeline rendszerek irányába haladt kutatásaival [29], míg jómagam az objektum-orientált rendszerek területén folytattam kutatásaimat.

2. Módszerek

Bizonyítottan helyes programok előállításának egyik módszere a szintézis módszer [14], [17], mely garantálja, hogy az előállított programkód megfelel a specifikációnak. A szintézis módszeréhez szükség van egy specifikációs nyelvre, valamint egy algoritmusra, mely a specifikáció alapján előállítja a programkódot. Az MPCTL* (Many Processes Computational Tree Logic*) [14] nyelvet választottuk a szinkronizáció specifikációjához, mely egy elágazó idejű temporális logikai [17] nyelv.

A specifikáció alapján történő programkód előállítására Attie és Emerson által bevezetett „Many Processes" szintézis módszerre [14] alapozott eljárást használtunk, mely Emerson és Clarke tabló alapú módszerének [17] egy kiterjesztése. Attie és Emerson módszere az állapotrobbanás problémáját a hasonlóság feltételének megkövetelésére alapozva oldotta meg.

A szinkronizációs kód automatikus előállításának problémáját objektumorientált környezetben kívánjuk megoldani. Ehhez szükséges a rendszer bizonyos strukturális, logikai tulajdonságainak megtervezése, mely feladatra az UML-t (Unified Modeling Language) [27], [28] választottuk.

Az absztrakt programok implementációjához felhasználjuk a párhuzamos programozás egy elterjedt eszközét, a szemaforokat [30], az elosztott adatbázisok kezelésénél is használatos tokeneken alapuló lockolás módszerét [24], valamint a feltétel-kiértékeléseknél használható „add át a stafétabotot" technika [13] alapelveit.

Példáinkat Java [22], [25] nyelven mutatjuk be standard eszközök használatával.

3. Az értekezés felépítése

Tanulmányunkat az alapok tárgyalása után az objektumok közötti szinkronizációs kapcsolatok vizsgálatával kezdtük és a 3.2.1 fejezetben megterveztük a szinkronizációs kapcsolatok kezelésének strukturális felépítését objektumorientált környezetben, mely *új eredményünk*. A 3.2.2 fejezetben egy *általunk kidolgozott módszert* adtunk a szinkronizációs kapcsolatok specifikációjára.

A 3.2.3 fejezetben *módszert dolgoztunk ki* az összeköttetési reláció implementálására. A szinkronizációs specifikáció objektum-elvű környezetben történő megadásához az objektumok szinkronizációs részeinek osztályszerkezetét *definiáltuk* a 3.2.4 fejezetben, mely *általunk bevezetett*, automatikusan létrehozható osztályokat tartalmaz. A 3.2.5 fejezetben olyan megszorításokat vezettünk be, melyek segítségével a szinkronizációs specifikáció nagyobb fokú átláthatóságát biztosíthatjuk. A 3.2.6 fejezetben meglévő szerkezetekre építve *algoritmust dolgoztunk ki* az absztrakt szinkronizációs kód megvalósítására. A 3.3 fejezetben *újszerű módon* közelítettük meg a hasonlóság kérdését, módszert adtunk a szinkronizáció és kiszámítás strukturális szétválasztására, a nagyszámú állapot hatékonyabb kezelésére, valamint a szintézis algoritmus több osztályos környezetben történő alkalmazására. A 3.4 fejezetben módszert adtunk az állapot particionálási öröklődési anomália részleges kezelésére, mely *új eredményünk*, továbbá *újszerű módon alkalmazzuk* az öröklődés lehetőségeit a dinamikus szinkronizációs viselkedés váltásainak definiálására. A 3.5 fejezetben *módszert dolgoztunk ki* osztott osztályok szinkronizációjának specifikálására és implementálására, melyet visszavezettünk a párhuzamos objektum-orientált rendszerekre. A 3.6 fejezet a szinkronizációs kód és kiszámítási kód automatikus összekapcsolásának lehetőségeit tárgyalja kiterjesztett állapotdiagramokra építve, mely *új eredményünk*. A 4.1 fejezetben téroperátorok kiterjesztésével és a kapcsolatrendszer egyirányúsításával megnöveltük a módszer alkalmazhatóságát, mely *új eredményünk*. A 4.2 fejezet a módszer használhatóságának növelését tűzi ki célul grafikus elemek *bevezetésével*. A 4.3 fejezet az előállított program *hatékonyságának növelésével* foglalkozik. Az értekezést a kapcsolódó témakörök vizsgálata és egy rövid összefoglaló fejezet után a nyitott kérdéseket tárgyaló rész zárja.

4. Tézisek

4.1. Objektumok szinkronizációs kapcsolatai

Attie és Emerson módszerére [14] alapozva az egymással kapcsolatban lévő hasonló folyamatok szinkronizációját adhatjuk meg. Objektum-elvű rendsze-

rek esetén figyelnünk kell az objektumok dinamikus keletkezésének és megszűnésének jelenségére, ezáltal az új objektumok kapcsolatainak kialakítására, valamint a megszűnő egyedek kapcsolatainak biztonságos felszámolására.

Az objektumkapcsolatokat tartalmazó I összeköttetési relációban [14] az implementált kód könnyebb kezelhetősége érdekében a folyamat-index párokat objektummutatókból alkotott párokkal helyettesítettük. Ennek segítségével az UML [27], [28], jelölésrendszerére támaszkodva megterveztük az I kezeléséhez szükséges osztályok struktúráját. Az összeköttetési reláció kezelése központilag valósul meg az ún. *SharedObject* osztályon keresztül, mely osztályszintű metódusokat támogató nyelvek esetén statikus függvényekkel, egyéb nyelvek esetén a *Singleton* minta [15] szerint példányosított globálisan elérhető objektum segítségével valósítható meg.

Az I objektumpárjainak lehetséges elemeire építve, az MPCTL* [14] nyelv elemeinek felhasználásával specifikációs módszert adtunk, melyben Attie és Emerson által definiált téroperátorok kiterjesztésének segítségével a két kapcsolódó objektum típusától függően tudjuk megfogalmazni a kapcsolat létrehozásának feltételeit. (Ezen specifikáció implementálása szintén központilag használt kódhoz vezet.)

Az összeköttetési reláció implementálása a specifikációban szereplő elsőrendű logikai kifejezések kiértékelésének implementációján alapul, figyelembe véve, hogy a téroperátorok miatt az összes lehetséges objektum-pár vizsgálata szükséges. A kvantorok kezelése performanciális problémákat okoz, ezért az I általuk indukált módosításait optimalizáltuk. Az I kezelésével kapcsolatos többi kód „konstans"-nak tekinthető, működésüket így nagyobb hatékonysággal tudtuk megvalósítani. Az I kezelésére adott kód biztonságos viselkedése az író/olvasó probléma megvalósítása során bevezetett számlálásra épülő szemaforos technika [21] segítségével adható meg. Az I kezelésére vonatkozó módszer egy kezdeti verzióját [1]-ben közöltünk.

1. tézis. *Módszert adtunk objektumok összeköttetési relációjának specifikációjára, megadtuk az I biztonságos kezeléséhez szükséges kód felépítését, implementációját, valamint kijelöltük az objektumkapcsolatok létének eldöntését végző függvény megvalósítási lehetőségeinek irányvonalát.*

4.2. Szinkronizáció specifikációja

A kapcsolódó objektumok egymáshoz viszonyított viselkedését specifikálni kell. Hogy a specifikált kód megvalósítható legyen, előbb a rendszer szinkronizációs kódjának struktúráját terveztük meg. Ahhoz, hogy egy objektum eldönthesse, lehetősége van-e munkájának folytatására, szükséges, hogy a kapcsolódó objektumoktól információt szerezzen az állapotukról.

Több osztály esetén az osztályok állapothalmazainak összevonása szükséges, hogy Attie és Emerson módszerét [14] használhassuk, mert így biztosítható az objektumok szinkronizációs hasonlósága. Ez megnöveli az állapotok számát – de csak típus szinten; nem olyan mértékben, mint egyed szinten történne. Az állapotok száma két módon csökkenthető. Az egyesített állapothalmazban állapotok összevonása lehetséges. Sajnos az állapotösszevonás nagyrészt manuálisan történik, csak néhány speciális esetben tudunk automatikusan állapotokat egyesíteni (pl. azonos szemantikájú kritikus állapotok összevonása). A másik módszer a szinkronizációs és a kiszámítási kód strukturális szétválasztása, mivel különböző típusú objektumoknak gyakran azonos a szinkronizációs viselkedése: a szinkronizációs kód külön osztályhierarchiába szervezhető. Ez jelentősen csökkenti a szinkronizációs osztályok, így az állapotok számát. Az öröklődés lehetőségeit felhasználva az objektumok dinamikusan változtathatják a szinkronizációs viselkedésüket a szinkronizációs struktúra különböző pontjaiba történő csatlakozással. A szinkronizációs specifikáció az MPCTL* nyelv használatával adható meg az összevont állapothalmazon. Objektumok szinkronizációjának specifikációs és implementációs problémáival kapcsolatos kezdeti megfontolásainkat [1], [2] és [5] tartalmazza.

Párhuzamos, osztály alapú programok esetén gyakran fordul elő, hogy egy osztály kiterjesztése során az ős szinkronizációs kódját is módosítani kell. Azokban az esetekben, mikor az új osztály új állapotai a régi állapotokhoz hasonlóan viselkedhetnek az ős osztályra nézve, az állapotok leképezésével megoldást adhatunk a problémára. Ebben az esetben a leszármazott objektumai osztály szinten beállítják az ősben az új állapotok lokális leképezését, mely hatására az ős szinkronizációs kódja megtartható. A leképezést UML

szinten, az osztálydiagramban adhatjuk meg.

Az osztott módon használható erőforrások speciális esetei az osztott osztályok. Osztott osztályok objektumainak belső viselkedésének tervezését visszavezettük az osztott objektumokból álló rendszerek esetére.

A szinkronizációs és kiszámítási osztályhierarchia közötti kapcsolatot a kiszámítási kódhoz tartozó osztályok állapotdiagramjával adhatjuk meg, melynek kiterjesztettük a szerkezetét, hogy megfelelő módon és átláthatóan lehessen definiálni a kapcsolatot. A módszer első változatát [11]-ben írtuk le.

A szinkronizációs kód generálásához Attie és Emerson módszerét használjuk, mely által generált köztes kód (ami egy automata) implementációjához a tokengyűjtés [24] technikáját használtuk fel, hogy az automata állapotátmeneteinek feltételeit a kapcsolódó objektumok körében kölcsönös kizárásos módon értékelhessük ki. Mivel a tokengyűjtés az I kezeléséhez kapcsolódik, ezért a *SharedObject* osztályban került megvalósításra (konstans kódként). A változó programkód az osztályok azonos felületen keresztül elérhető *setState* függvényében van implementálva az absztrakt automata alapján. A szinkronizációs kód megvalósításának egy kezdeti verziója található [1]-ben.

2. tézis. *Megoldást nyújtunk több osztályból álló objektum-orientált rendszerek szinkronizációs felépítésének megtervezésére, leválasztva a szinkronizációs struktúrát, melyhez kapcsolódóan az objektumok szinkronizációja hatékonyan specifikálható, biztonságos szinkronizációs kód generálható automatikusan, valamint eszközt adunk a kiszámítási kód szinkronizációs kódhoz történő kapcsolására, illetve az öröklődés hatékonyabb kezelésére. Megadtuk továbbá az osztott osztályok kezelésének egy lehetséges módját, visszavezetve az osztott rendszerek kezelésére.*

4.3. Kiterjesztések

A módszer használhatóságának növelésére bevezettük az erős térbeli operátorokat. Az erős térbeli operátorokkal specifikált viselkedés implementációja akkor is felfüggeszti az objektum futását, ha nincs kapcsolódó objektum, melyen a feltételeit kiértékelheti. Ennek segítségével le tudjuk írni a több más objektumot is kiszolgáló egyedek, valamint az egymásnak adatokat továbbító

objektumok működési követelményeit. Ezek mellett az összeköttetési reláció antiszimmetrikus, ezáltal sorrendiségek kezelése is lehetővé válik, valamint egyes objektumok nagyobb szabadságot élvezhetnek, miközben más hozzájuk szinkronizálja a működését. Az erős téroperátorok egy másik megközelítés szerinti bevezetése található [3]-ban.

A használhatóság növelésének egy másik eszközeként az MPCTL* specifikáció jelöléseinek leírására egy, az objektum-elvű tervezésben megszokott, diagramokra épülő módszert vezettünk be az UML egy kiterjesztéseként, szinkronizációs diagramok formájában. Ezen diagram típust úgy terveztük meg, hogy az MPCTL* segítségével leírható fontos tulajdonságokat továbbra is ki lehessen fejezni. Azon felül, hogy a szinkronizációs diagramokkal átláthatóbb formában fogalmazhatók meg a szinkronizációra vonatkozó elvárások, sok sablont is definiál, mellyel a tervezés menete gyorsítható, a specifikáció átláthatósága pedig tovább nő. A szinkronizációs diagramokat (sablonok nélkül) [11]-ben vezettük be.

Azáltal, hogy a szinkronizációs diagramban a tervező egy ismert problémakört jelölhet meg és tárolhat egy sztereotípiával, melyhez saját szinkronizációs kódot adhat, növelhető a generált program hatékonysága. Mivel a szinkronizáció nagyban függ az összeköttetési relációtól, a sztereotípia mellett az összeköttetési reláció formuláit is el kell menteni.

Az antiszimmetrikus kapcsolatok és az erős téroperátorok használata hatékony eszközt nyújt a kommunikációs lehetőségek növelésében. Ez azonban gyakran felesleges, esetenként költséges vizsgálatokhoz vezethet I kezelésében. Ezért a felhasználónak lehetősége van a kivételes esetek megnevezésére, azaz azon kifejezések megadására, amikor az ilyen vizsgálatok elvégzése nem szükséges (hiszen I-t nem változtathatja meg). Tovább javul a kód hatékonysága, ha a tokengyűjtések számát csökkentjük, melyet a több paraméterű státuszállító függvénnyel értünk el. Az optimalizálási sztereotípiákról és a több paraméterű státuszváltással történő optimalizálásról [10]-ben van szó.

3. tézis. *A módszer felhasználhatóságát nagyban megnöveltük az összeköttetések antiszimmetrikus kezelésével és a hozzá kapcsolódó erős téroperátorok bevezetésével, mely nagyobb mozgásteret nyújt az objektumok kapcsolatainak*

dinamikus alakításában, valamint a struktúrák kezelésének felhasználó által lehetséges optimalizálásával. A módszer átláthatóságát a grafikus jelölésrendszer, azon belül az általánosan használható, bonyolult struktúrák leírására szolgáló összetett elemek bevezetése segíti, és biztosított annak a lehetősége, hogy a felhasználó befolyásolja a generált kód hatékonyságát.

Irodalomjegyzék

Saját közlemények

Referált folyóiratokban megjelent dolgozatok

[1] Sz. Hajdara, L. Kozme and B. Ugron: *Synthesis of a system composed by many similar objects*, Annales Univ. Sci. Budapest., Sect. Comp. 22 (2003) 127-150

[2] B. Ugron, L. Kozma, Sz. Hajdara and L. Blum: *Implementations of synchronization of concurrent objects*, Annales Univ. Sci. Budapest., Sect. Comp. 23 (2004) 79-102

[3] Szabolcs Hajdara and Balázs Ugron: *An extension of synthesis of the synchronization of object-oriented pipeline systems*, Pure Mathematics and Applications (PU.M.A.), Vol. 15, No. 2-3 (2004), pp. 157-169

[b] 5th Joint Conference on Mathematics and Computer Science, Debrecen (2004)

[4] Balázs Ugron, Szabolcs Hajdara, and László Kozma: *Synthesis of the synchronization of general pipeline systems*, Acta Cybernetica 17 (2005) 123-151

Referált konferencia kiadványok, összefoglalók

[5] Szabolcs Hajdara, Balázs Ugron: *An Example of generating the synchronization code of a system composed by many similar objects*, 17th European Conference on Object-Oriented Programming (ECOOP), The 13th Workshop for PhD Students in Object-Oriented Systems, Darmstadt (2003), ECOOP 2003 Workshop Reader, LNCS 3013, Springer-Verlag, Germany (2004) 55

[6] Szabolcs Hajdara, Balázs Ugron: *Describing Semantics of Data Types in XML*, Proceedings of the 6th International Conference on Applied Informatics, Vol. I, B. V. B. Nyomda és Kiadó Kft., Eger (2004) pp. 173-180

[7] Balázs Ugron, Szabolcs Hajdara: *Synthesis of the synchronization of pipeline systems*, Proceedings of the 6th International Conference on Applied Informatics, Vol II, B. V. B. Nyomda és Kiadó Kft., Eger (2004) pp. 351-359

Konferencia előadások

[8] Hajdara, Sz., Kozma, L., Ugron, B.: *Párhuzamos programok szintézise és objektum elvű kiterjesztése*, V. Országos Objektum-orientált Konferencia, Dobogókő (2002)

[9] Ugron, B., Kozma, L., Hajdara, Sz.: *A strukturált programozástól az objektum elvű technológiáig*, V. Országos Objektum-orientált Konferencia, Dobogókő (2002)

[10] Szabolcs Hajdara, Balázs Ugron: *Analyzing the effectiveness of synthesized synchronization codes*, 6th Joint Conference on Mathematics and Computer Science, Pécs (2006)

Egyéb tudományos eredmények

[11] Szabolcs Hajdara, Balázs Ugron, László Kozma: *A graphical interface for specifying the synchronization of a concurrent system*, Periodica Politechnica, Budapest(2005) – közlésre benyújtva

[12] Szabolcs Hajdara: *Objektum-elvű rendszerek szintézisproblémái*, Professional conference of BME, Department of Measurement and Information Systems (2007)

Az értekezés szempontjából fontos további közlemények

[13] G. R. Andrews: *A Method for Solving Synchronization Problems*, Science of Computer Programming 13 (1989/90), pp.1-21.

[14] P. C. Attie, E. A. Emerson: *Synthesis of Concurrent Systems with Many Similar Processes*, ACM TOPLAS Vol. 20, No. 1 (1998), pp. 51-115.

[15] J. W. Cooper: *Java Design Patterns: A Tutorial*, Addison-Wesley (2000), ISBN 020 148 539 7.

[16] M. Dwyer, J. Hatcliff, M. Mizuno, M. Neilsen, G. Singh: *Automatic Derivation, Integration and Verification of Synchronization Aspects in Object-Oriented*

Design Methods Report for DARPA Order K203/AFRL Contract F33615-00-C-3044 (2001).

[17] E. A. Emerson, E. M. Clarke: *Using branching time temporal logic to synthesize synchronization skeletons*, Science of Computer Programming, 2 (1982), pp. 241-266.

[18] C. Girault, R. Valk: *Petri Nets for Systems Engineering*, Springer (2007), ISBN 354 041 217 4.

[19] D. Harkey, R. Orfali: *Client/Server Programming with Java and CORBA, 2nd Edition*, John Wiley & Sons (1998), ISBN 047 124 578 X.

[20] S. Jagadish, R. K. Shyamasundar: *UML based approach to specify secured, fine-grained concurrent acces to shared variables*, Journal of Object Technology, Vol. 6, No. 1 (2007), pp. 107-119

[21] L. Kozma, L. Varga: *A szoftvertechnológia elméleti kérdései*, ELTE Eötvös Kiadó (2003).

[22] D. Liang: *Introduction to Java Programming-Comprehensive Version (6th Edition)*, Prentice Hall (2006), ISBN 013 222 158 6.

[23] B. Meyer: *Object-Oriented Software Construction, Second Edition*, Prentice Hall (1997), ISBN 013 629 155 4.

[24] T. Minoura, G. Wiederhold: *Resilient Extended True-Copy Token Scheme for a Distributed Database System*, IEEE Trans. Software Eng. 8(3) (1982), pp. 173-189.

[25] G. J. Nyékyné et al. ed.: *Java 2 útikalauz programozóknak 1.3 (Programmers' Guide to Java 2, version 1.3)*, ELTE TTK Hallgatói Alapítvány, Budapest, Hungary (2001).

[26] G. J. Nyékyné et al. ed.: *J2EE útikalauz programozóknak (Programmers' guide to J2EE)*, Kiskapu Kft., Budapest, Hungary (2002).

[27] T. Quatrani: *Visual Modeling with Rational Rose 2002 and UML*, Addison-Wesley (2003), ISBN 020 172 932 6.

[28] S. Sike, L. Varga: *Objektum elvű modellalkotás UML-ben – Példatár definíciókkal* –, ELTE, TTK, Informatikai Tanszékcsoport, Budapest (2001).

[29] B. Ugron: *Synthesis of the synchronization code of pipeline systems*, PhD Thesis, ELTE IK, Budapest (2007).

[30] A. S. Woodhull, A. S. Tanenbaum: *Operating Systems. Design and Implementation.*, Prentice Hall (1997)