TEHNOLOGIJA, INFORMATIKA I OBRAZOVANJE
ZA DRUŠTVO UČENJA I ZNANJA
6. Međunarodni Simpozijum, Tehnički fakultet Čačak, 3–5. jun 2011.
TECHNOLOGY, INFORMATICS AND EDUCATION
FOR LEARNING AND KNOWLEDGE SOCIETY
6th International Symposium, Technical Faculty Čačak, 3–5th June 2011.

UDK: 004.43 Stručni rad

KOMPAJLIRANJE IZVORNOG KODA U UPRAVLJIVE MODULE

Zdravko Ivanković [1], Branko Markoski[2], Ružica Ivković[3], Predrag Pecev[4],Višnja Istrat[5]

***Rezime**: Izvorni kod napisan u bilo kom programskom jeziku koji podržava CLR se pomoću kompajlera pretvara u upravljivi modul (managed modul). Upravljivi modul je standardni 32-bitni Microsoft Windows prenosni izvršni fajl ili 64-bitni Windows prenosni izvršni fajl koji zahteva CLR. Prirodni (native) kompajleri kreiraju kod koji je namenjen specifičnoj CPU arhitekturi, kao što su x86, x64 ili IA64. Svi CLR kompajleri kreiraju IL kod (naziva se i managed kod jer CLR upravlja njegovim izvršavanjem). Microsoft C#, Visual Basic, F# i IL Asembler uvek kreiraju module koji sadrže upravljiv kod i upravljive podatke. Krajnji korisnici moraju imati CLR (dolazi kao deo .NET Framework-a) instaliran na svojim mašinama kako bi pokrenuli upravljive module.*

***Ključne reči:** CLR, upravljivi modul, metapodaci, manifest.*

SOURCE CODE COMPILATION INTO MANAGED MODULES

***Summary**: Source code written in any programming language that supports the CLR is compiled into a manageable module by compiler. Managed modul is standard 32-bit Microsoft Windows portable executable file or 64-bit Windows portable executable file that requires CLR for execution. Native compilers create source code that is designed for specific CPU architecture, such as x86, x64 or IA64. All CLR compilers create IL code (also called managed code because the CLR manages its execution). Microsoft C#, Visual Basic, F# and IL Assembler always create modules that contain the controllable and manageable data. End users must have a CLR (comes as part of .NET Framework) installed on their machines to run the managed module.*

***Key words:** CLR, managed modul, metadata, manifest.*

[1] Zdravko Ivanković, Univerzitet u Novom Sadu, Tehnički fakultet "Mihajlo Pupin" Zrenjanin, Đure Đakovića bb, Zrenjanin, E-mail: zdravko@tfzr.uns.ac.rs
[2] Branko Markoski, Univerzitet u Novom Sadu, Tehnički fakultet "Mihajlo Pupin" Zrenjanin, Đure Đakovića bb, Zrenjanin, E-mail: markoni@uns.ac.rs
[3] Ružica Ivković, Univerzitet u Novom Sadu, Tehnički fakultet "Mihajlo Pupin" Zrenjanin, Đure Đakovića bb, Zrenjanin
[4] Predrag Pecev, Univerzitet u Novom Sadu, Prirodno-matematički Fakultet, Trg Dositeja Obradovića 6, Novi Sad
[5] Višnja Istrat, Univerzitet u Novom Sadu, Tehnički fakultet "Mihajlo Pupin" Zrenjanin, Đure Đakovića bb, Zrenjanin

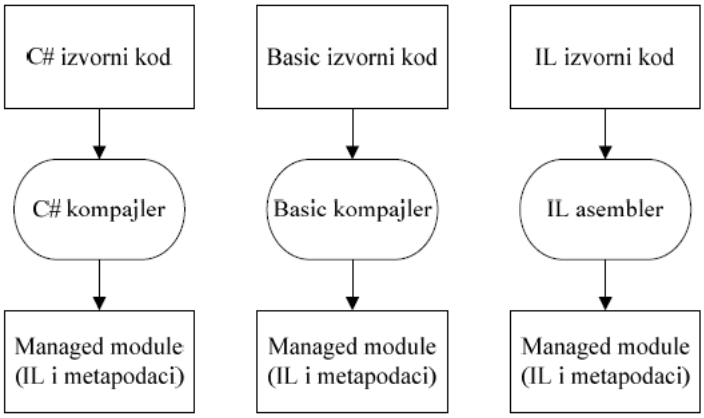
1. UVOD

Prilikom kreiranja nove aplikacije, pre svega je potrebno odabrati okruženje i programski jezik. To često zna da bude težak izbor jer različiti jezici nude različite mogućnosti. Npr. programski jezik unmanaged C/C++ omogućava kontrolu nad sistemom na dosta niskom nivou. Omogućeno je upravljanje memorijom, kreiranje niti itd. Sa druge strane Microsoft Visual Basic 6 omogućava kreiranje UI aplikacija veoma brzo kao i jednostavnu kontrolu nad COM objektima i bazama podataka.

Zajedničko izvršno okruženje - CLR (Common Language Runtime) predstavlja izvršni sloj koji koriste različiti programski jezici. Osnovne funkcionalnosti CLR-a (kao što su upravljanje memorijom, bezbednost, upravljanje izuzecima i sinhronizacija niti) su dostupne svim jezicima koji se oslanjaju na njega. Npr. CLR koristi izuzetke kako bi prijavljivao greške, pa će svi jezici koji pozivaju CLR dobijati greške prikazane pomoću izuzetaka. U vreme izvršavanja, CLR ne zna koji programski jezik se koristi za kreiranje izvornog koda. Ovo znači da se može koristiti bilo koji jezik za kreiranje aplikacije, sve dok kompajler za dati jezik poziva CLR.

2. KOMPAJLIRANJE KODA

Microsoft je kreirao više kompajlera koji se naslanjaju na CLR: C++/CLI, C#, Visual Basic, F#, Iron Python, Iron Ruby i Intermediate Language (IL) Asembler. Pored Microsoft-a, više drugih kompanija i univerziteta takode je kreiralo kompajlere koji koriste CLR. Medu njima su kompajleri za Ada, COBOL, Fortran, LISP, ML, Pascal, Perl, Php, Prolog, RPG,... Slika 1. prikazuje proces kompajliranja izvornih fajlova.



Slika 1: *Kompajliranje izvornog koda u upravljive module*

Kao što se na slici vidi, može se kreirati izvorni kod napisan u bilo kom programskom jeziku koji podržava CLR. Kompajler dati kod pretvara u upravljivi modul (managed modul). Upravljivi modul je standardni 32-bitni Microsoft Windows prenosni izvršni fajl (PE32 - Portable Executable) ili standardni 64-bitni Windows prenosni izvršni fajl (PE32+) koji zahteva CLR da bi se izvršio. Opis delova managed modula se nalazi u tabeli 1.

Tabela 1: Delovi upravljivih modula

Deo	Opis
PE32 ili PE32+ zaglavlje	Ukoliko zaglavlje fajla koristi PE32 format, fajl se može izvršavati na 32-bitnoj ili 64-bitnoj verziji Windowsa. Ukoliko zaglavlje koristi PE32+ format, fajl zahteva 64-bitnu verziju Windowsa. Ovo zaglavlje ukazuje i na tip fajla: GUI, CUI ili DLL, i sadrži vremenski podatak kada je fajl kreiran. Za module koji sadrže samo IL kod, masa informacija koje se nalaze u PE32(+) zaglavlju su izostavljene. Za module koji sadrže izvorni CPU kod, zaglavlje sadrži informacije o datom kodu.
CLR zaglavlje	Sadrži informacije koje fajl čine managed modulom. Zaglavlje uključuje zahtevanu verziju CLR-a, određene flegove, *MethodDef* token ulazne metode (main metoda), kao i lokaciju/veličinu metapodataka datog modula, njegovih resursa, jakih imena, određenih flegova, i drugih manje značajnih stavki.
Metapodaci	Svaki managed modul sadrži tabelu sa metapodacima. Postoje dva glavna tipa tabela: tabele koje opisuju tipove i članove definisane u kodu i tabele koje opisuju tipove i članove na koje kod upućuje a koje se nalaze u drugim modulima.
IL kod	Kod koji kompajler kreira iz izvornog koda. U vreme izvršavanja, CLR kompajlira IL u CPU instrukcije.

Prirodni (native) kompajleri kreiraju kod koji je namenjen specificnoj CPU arhitekturi, kao što su x86, x64 ili IA64. Svi CLR kompajleri kreiraju IL kod (naziva se i managed kod jer CLR upravlja njegovim izvršavanjem).

Pored toga što kreira IL, svaki CLR kompajler kreira skup metapodataka koji ugrađuje u managed modul. Najkraće rečeno, metapodaci su skup tabela koje opisuju šta je definisano u modulu, kao što su tipovi i njihovi članovi. Pored toga, metapodaci imaju i tabele koje ukazuju na reference upravljivih modula, kao što su uvezeni tipovi i njihovi članovi. Metapodaci su nadskup starijih tehnologija kao što su COM Type Libraries i Interface Definition Language (IDL). Za razliku od ovih tehnologija, metapodaci predstavljaju kompletniji skup podataka koji se uvek nalazi ugraden u EXE/DLL fajl u kom se nalazi i IL kod.

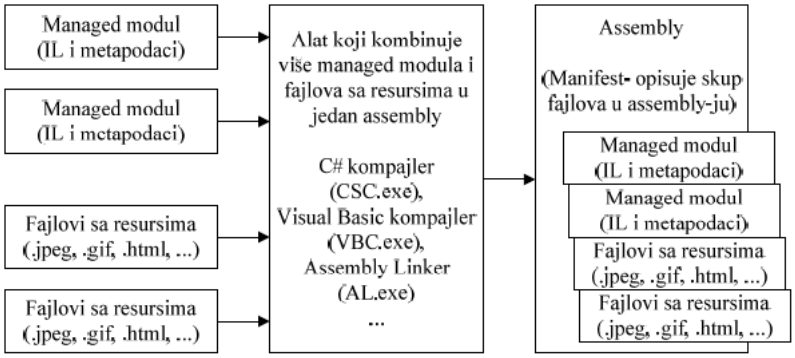
Microsoft C#, Visual Basic, F# i IL Asembler uvek kreiraju module koji sadrže upravljiv kod i upravljive podatke (tipovi podataka koji se mogu ukloniti pomocu garbage collector-a). Krajnji korisnici moraju imati CLR (dolazi kao deo .NET Framework-a) instaliran na svojim mašinama kako bi pokrenuli upravljive module.

3. KOMBINOVANJE UPRAVLJIVIH MODULA U ASSEMBLY

CLR zapravo ne radi sa modulima nego sa assembly-ima. Assembly je apstraktan koncept. On predstavlja logicko grupisanje jednog ili više modula ili fajlova sa resursima. To je najmanja jedinica sigurnosti, verziranja i višestruke upotrebljivosti. U zavisnosti od izbora, u assembly može uci jedan ili više fajlova. U CLR svetu, assembly predstavlja

komponentu. On omogucava da se grupa fajlova tretira kao jedan entitet.

Slika 2. objašnjava suštinu vezanu za assembly-je. Na ovoj slici se managed moduli i fajlovi sa resursima procesiraju pomocu odgovarajucih alata. Ovi alati kreiraju jedan PE32(+) fajl koji predstavlja logicko grupisane fajlove. Kreirani PE32(+) fajl sadrži blok podataka koji se naziva manifest. Manifest je u suštini još jedan blok metapodataka koji opisuje fajlove koji čine assembly, javno pristupne tipove koje implementiraju fajlovi u assembly-ju i resurse ili fajlove sa podacima koji su pridruženi assembly-ju.



Slika 2: *Kompajliranje upravljivih modula u assembly-je*

U podrazumevanom slučaju, kompajleri pretvaraju upravljive module u assembly-je; odnosno, C# kompajler kreira upravljiv modul koji sadrži manifest. Manifest ukazuje da se assembly sastoji od samo jednog fajla. Za projekte koji imaju samo jedan upravljivi modul bez fajlova sa resursima (ili podacima), assembly je upravljiv modul pa nisu potrebni bilo kakvi dodatni koraci u procesu kreiranja. Ukoliko se želi grupisanje više fajlova u assembly, potrebno je još alata (kao što je assembly linker, AL.exe).

Način podele koda i resursa u razlicite fajlove je u potpunosti prepušten programeru. Npr. mogu se retko korišceni tipovi i resursi staviti u posebne fajlove koji su deo jednog assembly-ja.

Moduli sa assembly-ima sadrže i informacije o referenciranim assembly-ima (ukljucujuci i broj njihove verzije). Ova informacija omogucava da assembly bude samoopisan. Drugim recima, CLR može odrediti zavisnosti datog assembly-ja kako bi pribavio neophodan kod za njegovo izvršavanje. Zbog toga, nikakve dodatne informacije nije potrebno upisati u Windows registry ili Active Directory Domain Services (AD DS).

4. POKRETANJE ZAJEDNIČKOG IZVRŠNOG OKRUŽENJA

Svaki assembly koji se kreira može biti izvršna aplikacija ili DLL koji sadrži skup tipova koji koriste izvršne aplikacije. CLR je odgovoran za upravljanje izvršavanjem koda koji se nalazi u okviru assembly-ja. Ovo znaci da .NET Framework mora biti instaliran na mašini na kojoj se pokrece aplikacija.

.NET Framework SDK ukljucuje alat CLRVer.exe koji prikazuje sve verzije CLR-a koje su instalirane na mašini. Ovaj alat može prikazati i koju verziju CLR-a koristi proces koji se trenutno izvršava.

Ukoliko assembly sadrži samo kod sa "bezbednim" tipovima (kod koji se može izvršavati i na 32-bitnim i na 64-bitnim verzijama Windows-a), kreirani EXE/DLL fajl se izvršava na

obe verzije Windows-a. Drugim rečima, fajl se izvršava na bilo kojoj mašini koja ima instaliranu odgovarajuću verziju .NET Framework-a. U retkim prilikama, programeri žele da kreiraju kod koji radi samo na određenoj verziji Windows-a. Pomoću raspoloživih opcija moguće je kreirati kod koji se izvršava samo na x86 mašinama i 32-bitnoj verziji Windows-a, x64 mašinama i 64-bitnoj verziji Windows-a i Intel Itanium mašinama i 64-bitnoj verziji Windows-a.

U zavisnosti od izabrane platforme, C# kompajler ce kreirati assembly koji sadrži PE32 ili PE32+ zaglavlje i podatke u željenoj CPU arhitekturi.

Prilikom pokretanja izvršnog fajla, Windows ispituje zaglavlje datog fajla kako bi odredio da li aplikacija zahteva 32-bitni ili 64-bitni adresni prostor. Fajl sa PE32 zaglavljem se može izvršavati na 32-bitnom i 64-bitnom adresnom prostoru, a fajlovi sa PE32+ zaglavljem zahtevaju 64-bitni adresni prostor. Windows takođe proverava informacije o CPU arhitekturi koje se nalaze u zaglavlju kako bi proverio da li odgovaraju tipu procesora u računaru. Tehnologija koja se zove WoW64 (Windows on Windows64) omogucava 32-bitnim Windows aplikacijama da se izvršavaju na 64-bitnim mašinama.

5. ZAKLJUČAK

Upotreba upravljivih modula i assemly-ja omogućava komforniji rad sa izvornim kodom. Metapodaci koji se u njima nalaze, su nadskup starijih tehnologija kao što su COM Type Libraries i Interface Definition Language (IDL). Za razliku od ovih tehnologija, metapodaci predstavljaju kompletniji skup podataka koji se uvek nalazi ugraden u EXE/DLL fajl u kom se nalazi i IL kod. Moduli sa assembly-ima sadrže i informacije o referenciranim assembly-ima što omogucava da assembly bude samoopisan. Drugim recima, CLR može odrediti zavisnosti datog assembly-ja kako bi pribavio neophodan kod za njegovo izvršavanje. Zbog toga, nikakve dodatne informacije nije potrebno upisati u Windows registry.

6. LITERATURA

[1] Richter J.: *CLR via C#*, Microsoft Press, Redmond, 2010.
[2] Pascal Felber, Christof Fetzer and Torvald Riegel. Dynamic Performance Tuning of Word-Based Software Transactional Memory. Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). 2008.
[3] Maurice Herlihy, Victor Luchangco and Mark Moir. A flexible framework for implementing Software Transactional Memory. Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) 2006.
[4] Maurice Herlihy. SXM: C# Software Transactional Memory. [Online] February 17, 2005
[5] Ralf Westphal. NSTM - .NET Software Transactional Memory. [Online] August 5, 2007
[6] Erik Meijer and Jim Miller- Technical Overview of the Common Language (or why the JVM is not my favorite execution environment). 2001.
[7] T. Riegel, P. Felber and C. Fetzer. A lazy snapshot algorithm with eager validation. Proceedings of the International Symposium on Distributed Computing (DISC). 2006.
[8] Joe Duffy. Concurrent Programming on Windows: Addison Wesley, 2008.