Microsoft Shared Source CLI

Artur Popławski

Seminarium magisterskie
“Zagadnienia Programowania Obiektowego”

Co to jest SSCLI?

- wirtualna maszyna, implementacja na 3 systemy
  - "data-driven architecture, in which language-agnostic blobs of data are brought to life as self-assembling typesafe software systems"
- Shared Source, dystrybucja wraz ze źródłami
- przenośność kodu, oprogramowanie komponentowe
- standardy ECMA:
  - Common Language Infrastructure (334)
  - Język programowania C# (335)
  - Jscript (242)
- nazwa kodowa "ROTOR"
- SSCLI a .NET

Zasady licencji “Shared Source”

- tylko do użytku niekomercyjnego
- można modyfikować i rozpowszechniać kod
- modyfikacje:
  - nie wolno wykorzystywać w celach komercyjnych
  - należy zaznaczyć, że nie jest to oryginalny kod
  - rozpowszechniane z “kompatybilną” licencją
- należy zachować informację o prawach autorskich
- nie ma żadnych gwarancji na kod

Cele powstania i status projektu

- przedstawienie przykładowej implementacji standardów CLI i C#
- zachęta dla powstawania innych implementacji
- do nauczania dla środowisk naukowych
- dla wszystkich zainteresowanych VM
- próba zbudowania wspólnoty
- pierwsze wydanie ma stadium “beta”
- czyżby zastój?

Składniki SSCLI

- około 1.9 miliona linii kodu, z czego 60% w C/C++, 30% w C# i trochę kodu w CIL i ASM
- w pełni funkcjonalny kompilator C# i Jscript
- clix, czyli loader programów
- narzędzia programisty:
  - linker, assembler, disassembler
  - narzędzia do przeglądania zasobów i składników VM
  - debuggery
  - duży zbiór bibliotek standardowych

Przegląd architektury

- Typy
  - typy danych i metadane
  - zorientowane obiektowo
  - “properties” i “events”
  - symboliczne referencje
- Common Intermediate Language
- “assemblies” -- pudełka dla typów
- kompilacja Just In Time (JIT)
- nadzorowane wykonywania programów
- izolacja komponentów, konwencje nazywania
- Platform Adaptation Layer

Prosty przykład kompilacji

- clix ładuje silnik VM
- ładowanie i weryfikacja kodu
- ładowanie metadanych
- weryfikacja i ładowanie typów
- kompilacja JIT
- wykonanie kodu

System typów

- zapewnia możliwość współpracy z SO i innym kodem
- CLI jest środowiskiem silnie typowanym
- znaczenie systemu typów
  - kontrola typów
  - bezpieczeństwo
  - automatyczne wykrywanie błędów
  - umowa co do zasad działania
  - zapewnia prawidłowe działanie systemu
- “types as contracts”, typy zapewniają szczegółowy opis zasad, które dany element zobowiązuje się dotrzymać

System typów cd.

- typ
  - najprostsza jednostka, zawiera opis zachowania chroniący przed niewłaściwym użyciem
- obiekt
  - element System.Object, ma tożsamość i pamięć
- komponent
  - pojęcie szersze od obiektu, moduł, niezależna jednostka

System typów cd

- Value Types – abstrakcja najprostszych typów
- zdefiniowane dwa rodzaje rzutowania
- boxing/unboxing – przekazywanie VT jako wskaźnik
- typy proste:
  - int8, unsigned int8, float, enum
- referencje
  - silnie typowane, kontrola miejsca zapisu, nie wskazują na nic
  - interfejsy
  - delegaci (wskaźniki na funkcje, zawiera wskaźnik na obiekt i kod metody)
  - wskaźniki zarządzalne (niskopoziomowy wskaźnik wzbogacony o instrukcje dostępu do niego)

Metadane

- nieformalnie: dane opisujące inne dane
- bardziej formalnie: system opisujący elementy deklarowane i wskazywane w module
- metadane opisują strukturę modułu
- zaimplementowane jako baza relacyjna
- tokeny metadanych znajdują się w kodzie
- optymalizacje metadanych (read-only,“upychanie”)
- możliwość modyfikacji, atrybuty własne
- weryfikacja metadanych
- zawierają informacje o położeniu typu w pamięci

Assemblies

- moduł (assembly może zawierać wiele modułów), jednostka programu
- nacisk na możliwość wielokrotnego użycia
- niezależny od systemu i w pełni samoopisujący się
  - zawiera metadane (manifest) i zasoby
  - zapisany w formacie PE/COFF
- kontrola wersji i zabezpieczenia
- assembly może mieć atrybut private lub public
- Global Assembly Cache
- izolacja i domena aplikacji
  - analogia do izolacji procesów w SO

Manifest

- metadane zawarte w assembly
- manifest składa się z:
  - określenia “osobowości” assembly (nazwa, klucz publiczny)
  - wersja (numer, locale, hasz, nazwa)
  - opisu zawartości (typy, zasoby)
  - zależności
  - praw dostępu i własne atrybuty
- dane przechowywane najczęściej w postaci tablicy
- weryfikacja podczas działania

Assembly - ładowanie

- ładowane tylko kiedy są potrzebne
- Schemat ładowania:
  1. aplikacja woła metodę i w jej metadanych sprawdzana jest żądana wersja
  2. sprawdzenie GAC
  3. jeśli nie ma w GAC, to VM szuka gdzie indziej
  4. ładowanie metody
- możliwość konfiguracji sposobu ładowania

Wykonywanie kodu

- newobj tworzy nowy obiekt, a właściwie MethodTable i zmienne instancji na stosie
- ładowane są metadane oraz kod metod
  - MethodTable (wskaźniki do metod i interfejsów)
  - EEClass (informacje o klasie)
  - ClassLoader (przygotowanie klasy)
- kompilacja odkładana do ostatniego momentu
  - za kompilację odpowiedzialny jest “prestub helper”
  - weryfikacja kodu
  - obie czynności wykonane jednoprzebiegowo

Konwencje wołania

- po skompilowaniu kod jest gotowy do wykonania
  - na stosie znajduje się typowy rekord aktywacji
- konwencja JIT
  - call, calli (przez wskaźnik do metody) i tailcall
  - argumenty i wartość zwracana jest wrzucana na stos
  - na stos wrzucany jest wskaźnik this
- konwencja x86
  - konwersja wołania JIT (argumenty, ramka)

Intermediate Language

- assembler maszyny wirtualnej
- około 220 instrukcji (polimorfizm)
- Common Type System
- wykorzystuje stos, brak rejestrów
- wielkość ramki na stosie
- konstrukcje wysokopoziomowe
  - dostępu do danych
  - typowo obiektowe instrukcje
  - Structural Exception Handling

Kompilacja prostego programu

```
public class Echo {
    private string toEcho = null;
    public string EchoString {
        get { return toEcho; }
        set { toEcho = value; }
    }
    public string DoEcho() {
        if (toEcho == null)
            throw new Exception(“Nie ma co wypisać.”);
        return toEcho;
    }
}
```

```
.class public auto ansi beforefieldinit Echo extends System.Object
    {

    field private string toEcho
    .method public hidebysig specialname instance string
       get_EchoString() cil managed
    { ...}

    .property instance string EchoString() {
         .get instance string Echo::get_EchoString()
         .set instance void Echo::set_EchoString(string)
    }
}
```

```
.method public hidebysig instance string DoEcho() cil
   managed {
        .maxstack 2
        .locals init ([0] string ...)
        ldarg.0
        ldfld   string Echo::toEcho
        brtrue.s TRUE
        ldstr "Nie ma co wypisać."
        newobj instance void [mscorlib]          \
   System.Exception::.ctor(string)
        throw
        TRUE: ldarg.0; ldfld string Echo::toEcho
        stloc.0
        br.s JMP
        JMP: ldloc.0; ret
}
```

Przykłady instrukcji CLI

- Metadane:
  - .assembly extern mscorlib {} - odwołanie do innego assembly
  - .assembly OddOrEven { } - nazwa
  - .class public auto ansi Even extends [mscorlib]System.Object {...}
  - .field public static int32 val – definicja zmiennej
  - .ldstr “Enter a number” - w metadanych zapisane gdzie znajduje się łańcuch
- Inne instrukcje:
  - .entrypoint
  - .call vararg int32 sscanf(string,int8*,...,int32*)
  - .data FormatData = bytearray(25 64 00 00 00 00)
  - .method public static pinvokeimpl("msvcrt.dll" cdecl) vararg int32 sscanf(string,int8*) cil managed{}

Podstawowe instrukcje

- instrukcje mogą mieć sufiksy (np. brtrue.s)
- Instrukcje skoku:
  - bezwarunkowe (br, br.s)
  - warunkowe (brfalse, brtrue.s)
- Operacje na stosie:
  - ładowanie stałych (ldc.i4.1, ldc.i8.4)
  - ładowanie wskaźników (ldind.i1, ldind.u2)
  - zapisywanie do pamięci (stind.i1)
- Operacje arytmetyczne:
  - add, sub, mul, neg
- Operacje logiczne, na pamięci (cpblk), konwersji (conv)

Instrukcje IL cd.

- ładowanie argumentów (ldarg), zmiennych lokalnych (ldloc)
- ładowanie pól (ldfld)
- wołanie metod (call)
- tworzenie tablic (newarr), długość tablic (ldlen), ładowanie elementu tablicy (ldelem.u2)
- wyjątki:
  - .try, .catch, .finally

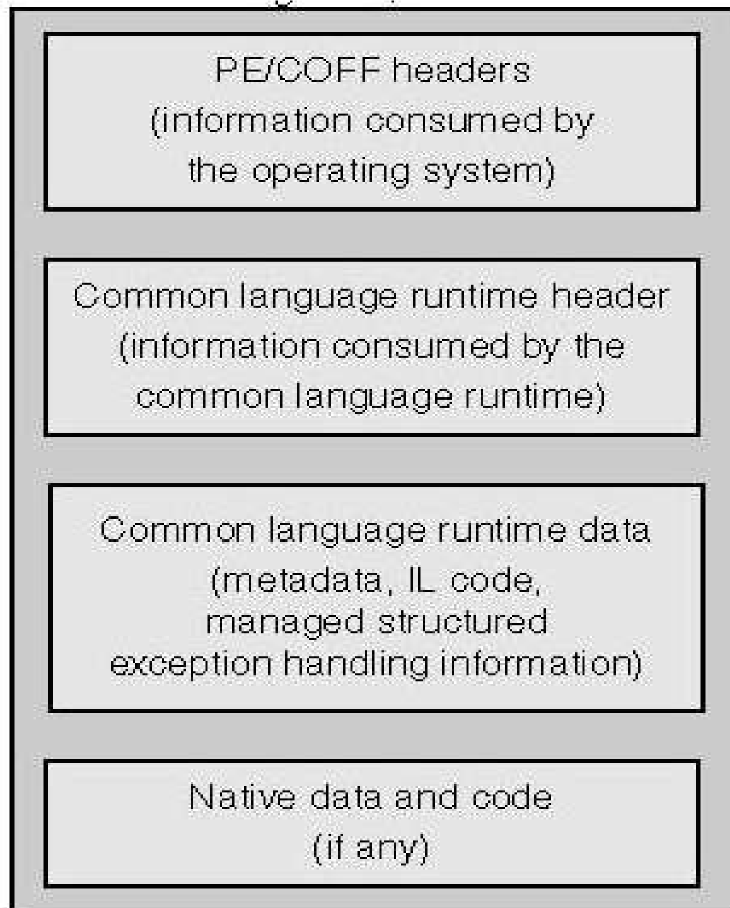
Garbage collector

- metoda finalize()
- zarządzanie stertą:
  - odnajdywanie martwych obiektów metodą “tracing the roots”
  - zwalnianie pamięci - “mark and sweep collection”
  - zapobieganie fragmentacji metodą kopiowania kolekcji
  - podział sterty (pokolenia obiektów i wielkie obiekty)
  - wskaźniki między pokoleniami obiektów i “write barrier” (kopiowanie obiektów), rola JIT

Garbage collector cd.

- Odzyskiwanie pamięci:
  - wywołanie GC (kosztowne i wywołanie w odpowiednim miejscu)
  - zawieszenie wszystkich wątków
  - “tracing the roots”
  - skopiowanie promowanych obiektów
  - poprawienie wskaźników
  - uaktualnienie listy wolnej pamięci

Struktura pliku wykonywalnego



- format PE/COFF
- nagłówki SO i CLR
- Sekcje:
  - .reloc
  - .text (metadane,IL,zas)
  - .sdata
  - .rsrc (zasoby zarządzalne)

Tworzenie pliku PE

- inicjalizacja buforów i szablonu pliku w pamięci
- kod IL jest kopiowany do sekcji .text
- dane są kopiowane do .sdata, metadane do buforów
- metadane, zasoby zarządzalne kopiowane do .text
- zasoby niezarządzalne do .rsrc
- zapis na dysku

Platform Adaptation Layer

- jednym z głównych celów CLI jest przenośność
- biblioteka zapewniająca interfejs pomiędzy VM a systemem operacyjnym
- wyższe warstwy komunikują się tylko z PAL
- podzbiór Win32 API
- prosta implementacja dla systemu Windows
- implementacja na systemy Uniksowe

Implementacja uniksowa

- dzielenie pamięci pomiędzy procesami przy użyciu pamięci dzielonej (mmap)
- pamięć dzielona jest zorganizowana w listę segmentów
- do identyfikacji zasobów służy uchwyt (handle)
- zarządca zasobów
  - pamięta ile razy zasób był zablokowany
  - zawiera informację o typie zasobu
  - uchwyty trzymane w formie listy ze wsk. na koniec

Procesy i wątki

- każdy proces (wątek) w VM odpowiada procesowi (wątkowi) w SO
- pthread i wykorzystanie syscalli, np. fork(), execve()
- Thread Local Storage
- Synchronizacja i implementacja (syscall pipe()):
  - sekcje krytyczne
  - muteksy
  - wydarzenia (events)
  - semafory
  - czekanie na zakończenie procesu/wątku

Problemy z przenośnością

- czekanie na proces w Windowsie i Uniksach
- zawieszanie wątków (pipe'y i zakleszczenia)
- bloki __try, __except i __finally i Structured Exception Handling (she)
  - w Uniksach implementacja przy użyciu sygnałów
- zarządzanie pamięcią i problemy z mmap
- blokowanie fragmentów plików
- asynchroniczne operacje na gniazdach

Więcej o SSCLI

- David Stutz, Ted Neward, Geoff Shilling, “Shared Source CLI Essentials”
- Serge Lidin,“Inside Microsoft .NET IL Assembler”
- http://msdn.microsoft.com/net/sscli/
- http://msdn.microsoft.com/net/ecma/
- http://www.sscli.net
- http://msdn.microsoft.com/msdnmag/issues/02/07/sharedsourcecli/default.aspx
- http://msdn.microsoft.com/msdnmag/issues/1100/gci/default.aspx