.NET Assembly

Introducere

Un asamblaj reprezinta un program executabil .NET (sau o parte a unui program executabil) care se prezinta ca o unitate de sine statatoare. Asamblajele reprezinta mijlocul utilizat de a impacheta programele C# pentru executie.

Atunci cand este realizat un program, *fisierul cu extensia exe obtinut in urma compilarii este un asamblaj*. Daca este contruita o biblioteca de clase (class library*), fisierul DLL (Dynamic Link Library) este deasemenea un asamblaj*. Asamblajul permite claselor, proprietatilor si metodelor publice sa fie vizibile si in alte programe. In schimb, toti membrii privati sunt mentinuti in interiorul asamblajului.

In cele ce urmeaza vom prezenta cateva aspecte legate de asamblaje. In particular, vom aborda urmatoarele:

- **O scurta trecere in revista a componentelor;**
- **Proprietatile asamblajelor, inclusiv abilitatea de a se autodescrie;**
- **Structura unui asamblaj si cum poate fi vizualizat continutul sau;**
- **Compatibilitatea versiunilor;**
- **Apelarea asamblajelor;**

Atunci cand un program C# este impachetat intr-un asamblaj, multe dintre proprietatile asamblajelor sunt destinate crearii si furnizarii unor clase speciale de programe numite componente.

O scurta trecere in revista a componentelor

O componenta este un subprogram sau o parte a unui program destinat pentru a fi utilizat de alte programe. In plus, o componenta este o unitate binara (cod executabil si nu cod sursa) care poate fi utilizata de alte programe fara a fi necesara o recompilare a codului sursa corespunzator componentei.

In sensul cel mai larg, o componenta include orice subprogram binar. Astfel orice DLL este o componenta deoarece este un subprogram continand cod executabil.

O definitie mai stricta prevede ca aceasta (componeta) sa aiba si capacitatea de a informa celelalte programe asupra continutului ei. Asamblajele au aceasta abilitate "de a-si face reclama" in .NET.

Avantajele utilizarii componentelor

Componentele ofera *posibilitatea reutilizarii subprogramelor intr-un mod flexibil*. In plus, reutilizarea unitatilor binare *salveaza timp si creste siguranta*.

Spre exemplu, sa consideram o clasa numita Shapes care contine obiecte pentru reprezentarea cercurilor, triunghiurilor sau pentru alte forme geometrice. Ar putea contine metode pentru calcularea ariei sau ar putea realiza alte operatii. Multe programe ar putea utiliza clasa Shapes: in arhitectura, inginerie, jocuri, design cu ajutorul calculatorului si altele. Astfel, nu ar fi grozav daca *routinele* pentru desenare si manipularea formelor *geometrice ar fi definite o singura data si reutilizate in toate aceste programe ?* Acesta este benficiul reutilizarii. Dar daca aceasta reutilizare ar fi *realizata fara a recompila si referi* clasa Shapes de fiecare data cand utilizam aceasta clasa ? Aceasta salveaza timp si creste siguranta deoarece elimina posibilitatea introducerii de probleme de fiecare data cand clasa este compliata si referita (compile and link). Chiar mai mult, daca o companie sau o persoana a scris aceasta componenta pe care o dorim, atunci o putem utiliza descarcand-o de pe internet sau cumparand-o, fara a fi necesar sa o scriem. *La nivel binar nu este necesar sa ne punem probleme legate de limbajul de programare utilizat pentru realizarea componentei.*

Arhitectura .NET si asamblajele furnizeaza toate aceste beneficii.

Un scurt istoric al componentelor

Pentru ca diverse programe sa utilizeze componente la nivel binar, trebuie sa existe ceva standard pentru implementarea modului cum sunt numite si utilizate clasele si obiectele la nivel binar. Metoda standard care realizeaza acest lucru in ceea ce priveste produsele Microsoft a evoluat de-a lungul timpului.

Microsoft Windows a introdus DLL (Dynamic Link Library) unde unul sau mai multe programe pot utiliza o parte din codul stocat intr-un fisier separat. Aceasta a functionat la nivelul de jos daca programele erau scrise in acelasi limbaj (uzual C). Totusi, programele trebuiau sa cunoasca in avans o multime de informatii despre DLL-ul pe care il utilizau si DLL-urile nu permiteau ca programele sa isi poata schimba date intre ele.

Pentru a permite schimbul de date, a fost creat DDE (Dynamic Data Exchange). Acesta a definit un format si un mecanism pentru a trimite date dintr-un program in altul, insa nu a fost suficient de flexibil. A urmat OLE 1.0 (Object Linking and Embedding) care a permis unui document precum fisierul Word sa contina un document dintr-un alt program (precum Excel). OLE 1.0 functiona ca o componenta, insa nu era o componenta standard.

Microsoft a definit prima sa componenta standard odata cu COM (Component Object Model), la mijlocul anilor 90. OLE 2.0 si alte tehnologii succesoare au fost construite pe baza lui COM. Distributed COM (DCOM) a introdus capacitatea componentelor COM de a interactiona in retea. COM functioneaza bine, insa este dificil de invatat (mai ales cand se utilizeaza din C++) si utilizat. COM necesita informatii despre componente, pentru inserarea lor in registri, facand instalarea complexa iar dezinstalarea dificila. COM a fost initial destinata utilizarii cu C si C++, fiind apoi extinsa la Visual Basic. De-a lungul timpului au fost semnalate mai multe probleme legate de DLL-uri. Intrucat utilizatorii puteau instala versiuni multiple ale DLL-urilor si componentelor COM furnizate de Microsoft sau alte companii, era foarte usor ca un program sa instaleze o versiune diferita a unui DLL deja utilizat de un alt program iar aceasta putea sa cauzeze caderea programului initial. Povara urmaririi tuturor informatiilor despre diferite DLL-uri instalate pe un system a facut dificila upgradarea si mentinerea componentelor.

Programarea .NET aduce un nou standard care se adreaseaza acestor probleme, asamblajul .NET (.NET assembly).

Proprietatile asamblajelor, inclusiv abilitatea de a se autodescrie

Inainte de a analiza structura unui asamblaj, sa discutam mai intai cateva proprietati ale asamblajelor .NET.

Autodescrierea

Cel mai important aspect al asamblajelor .NET, care le diferentiaza fata de predecesoare, este acela de a se autodescrie. Descrierea este continuta in asamblaj si astfel sistemul sau programul care apeleaza asamblajul nu trebuie sa caute informatii in registri sau in alta parte despre obiectele continute intr-un asamblaj.

Autodescrierea asamblajelor .NET trece dincolo de numele obiectelor, metodelor sau tipul de date al parametrilor. Un asamblaj .NET contine informatii *despre versiunea obiectelor* (spre exemplu Shapes 1.0 urmat de Shapes 1.1 sau Shape 2.0) *si controleaza securitatea obiectelor* continute. Toate aceste informatii sunt continute in asamblaj si nu este necesar ca programele care utilizeaza acest asamblaj sa caute informatii in alta parte. Aceasta face instalarea unei componente .NET mult mai usoara si mai directa decat tehnologiile Windows precedente. In fapt, trebuie literalmente copiat asamblajul pe discul dorit.

Asamblajele .NET si biblioteca de clase .NET

Fiecare program .NET, inclusiv programele C#, utilizeaza biblioteca de clase .NET. Aceste clase sunt apelate de fiecare data cand este apelata o metoda dintr-un spatiu de nume (spre exemplu spatiul de nume System). Fiecare clasa a acestei biblioteci este parte a propriului sau asamblaj. Spre exemplu, clasele care deseneaza sunt continute in asamblajul System.Drawing.dll. Daca atunci cand editati un program adaugati o referinta catre System.Drawing.dll, compilatorul va include o referinta catre acest asamblaj cand este construit asamblajul programului dumneavoastra. *La executie, CLR (Common Language Runtime) citeste metadatele din asamblajul programului dumneavoastra pentru a determina care sunt celelalte asamblaje necesare, iar apoi le localizeaza si le incarca pentru ca programul sa le utilizeze. Asamblajele programului dumneavoastra pot sa refere alte asamblaje. Proprietatea asamblejelor de a se autodescrie face posibila urmarirea tuturor referinelor fara a fi necesar ca programatorul sa le cunoasca*.

Corespondenta dintre spatiile de nume si asamblaje nu este bijectiva. Un asamblaj poate contine informatii din mai multe spatii de nume si reciproc un spatiu de nume poate fi impartit in mai multe asamblaje.

Programarea in limbaj mixt

Un beneficiu al asamblajelor .NET il reprezinta *programarea in limbaj mixt*, deoarece componentele pot fi apelate din oricare din limbajele .NET (C++, C#, Visual Basic) neavand importanta in care din limbaje au fost scrise acele componente.

Pentru a permite programarea in limbaj mixt, .NET asigura urmatoarele:

- CLR (Common Language Runtime) care gestioneaza executia tuturor asamblajelor .NET necesare programului;

-MSIL (Microsoft Intermediate Language) este un pseudocod sau limbaj intermediar generat de compilatoarele limbajelor .NET. Acest pseudocod standard este executat de CLR. De asemenea, CLR defineste formatul pentru pastrarea metadatelor asamblajelor. Aceasta inseamna ca asamblajele, oricare ar fi limbajul in care au fost scrise, au un format comun pentru a pastra metadatele;

- Common Language Specification (CLS) defineste trasaturile pe care limbajele trebuie sa le indeplineasca pentru a permite interoperabilitatea cu alte limbaje .NET.

- Common Type System (CTS) defineste tipurile de baza ale tuturor limbajelor .NET si regulile pentru definirea propriilor noastre clase.

In baza specificatiilor CLS rezulta ca putem scrie o componenta in C#, iar asamblajul care contine componenta poate fi utilizat de un alt program scris intr-un alt limbaj .NET precum Visual Basic .NET deoarece atat C# cat si Visual Basic .NET vor fi executate de CLR. Similar, programele C# pot utiliza componente scrise in Visual C++ .NET etc. La nivel de asamblaj, toate clasele, obiectele, tipurile de date utilizate de limbajele .NET sunt puse in comun, incat utilizatorul poate mosteni clase si utiliza componente indiferent de limbajul in care acestea au fost scrise.

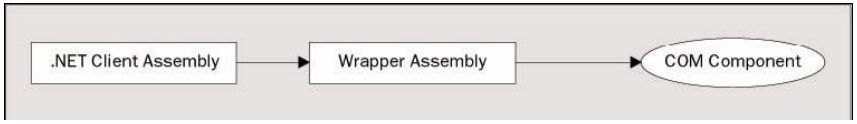
Interoperabilitatea cu COM sau cu alte tehnologii

Arhitectura .NET permite componentelor sau bibliotecilor scrise utilizand COM sau alte tehnologii sa fie utilizate cu C# sau alte limbaje .NET.

Acest mecanism functioneaza via proprietatii de autodescriere a asamblajului; Este creat un asamblaj (wrapper assembly) care “infasoara” componentele COM sau ale altor tehnologii in asa fel incat acestea sa se poata autodescrie pentru a fi executate in .NET. Asamblajul wrapper converteste tipurile de date COM la tipurile .NET si permite schimbul de apeluri de la limbajele .NET la COM si viceversa.

Visual Studio .NET creaza automat asamblajul wrapper atunci cand se adauga o referinta la o componenta COM (a se vedea dialogul Add Reference).

Diagrama de mai jos arata modul de lucru al asamblajului wrapper. Apelurile facute de asamblajul .NET client trec prin intermediul asamblajului wrapper pentru a ajunge la componenta COM. Din punct de vedere al asamblajului .NET, componenta este asamblajul wrapper si nu componenta COM.



Structura unui asamblaj si cum poate fi vizualizat continutul sau

Partile unui asamblaj mijlocesc programelor .NET aflarea existentei unora despre altele si rezolvarea problemei referintelor dintre programe si componente.

Asamblajele .NET formate dintr-un singur fisier au formatul din diagrama alaturata.

Un asamblaj contine:

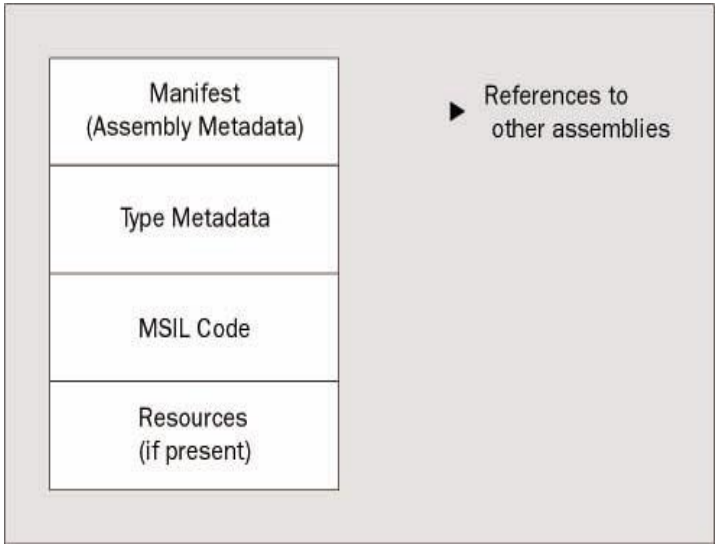
a) **metadatele** (date care descriu alte date) care permit altor programe sa caute clase, metode si proprietati ale obiectelor definite in asamblaj (**Manifest** si **Type Metadata**);

b) codul executabil al unui program sau al unei biblioteci de clase (**MSIL Code**);

c) resursele (daca exista). Resursele sunt partile neexecutabile ale unui program (precum imagini, iconite sau mesaje).

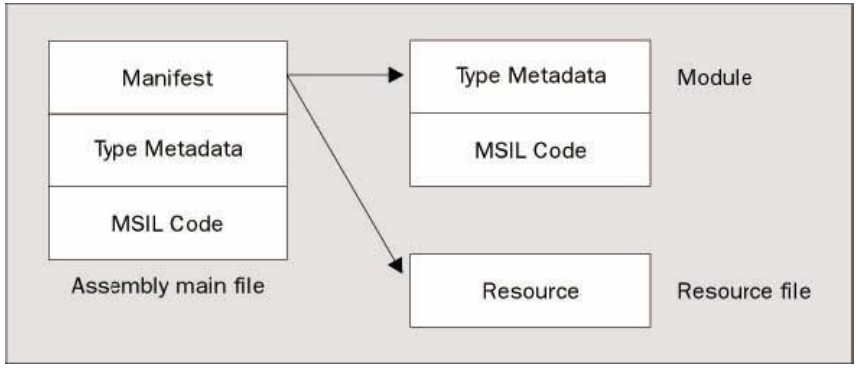
Fiecare asamblaj contine un **manifest** care descrie continutul asamblajului. El contine in esenta trei tipuri de informatii: informatii despre asamblajele externe pe care le refera, informatii despre asamblajul propriu-zis si modulele pe care le contine. El sta la baza conceptului de autodefinire.

Manifestul este urmat de **metadata type**, o descriere a claselor, proprietatilor, metodelor etc. continute de asamblaj, impreuna cu tipurile de date si valorile returnate. Apoi urmeaza **codul binar** pentru fiecare tip stocat in format MSIL.



Desi un asamblaj consta de cele mai multe ori dintr-un singur fisier dll sau exe, este posibil ca acesta sa contina mai multe fisiere (vezi diagrama).

Din punct de vedere al .NET-ului un asamblaj format din mai multe fisiere este privit ca o singura unitate logica care se intampla sa consiste din mai multe fisiere. Doar unul dintre fisiere contine manifestul. Manifestul indica celelalte fisiere ale asamblajului. Fisierele care contin cod executabil se numesc module. Modulele sunt asadar formate din tipuri metadata (metadata types) si cod MSIL. Ar putea avea si fisiere resurse care nu contin cod executabil.



Asamblajele formate din mai multe fisiere sunt utilizate doar in unele aplicatii avansate. Un modul este incarcat doar atunci cand este executat sau utilizat. Aceasta pentru a salva timp si memorie daca modulul nu este utilizat frecvent.

Exemplu: Vom crea un proiect de tip Class Library numit Shapes avand codul de mai jos.

```
namespace Shapes{
public class Circle{
double Radius;
public Circle(){
Radius = 0;
}
public Circle(double givenRadius){
Radius = givenRadius;
}
public double Area(){
return System.Math.PI * (Radius * Radius);
}
}
public class Triangle{
double Base;
double Height;
public Triangle(){
Base = 0;
Height = 0;
}
public Triangle(double givenBase, double givenHeight){
Base = givenBase;
Height = givenHeight;
}
public double Area(){
return 0.5F * Base * Height; // area = 1/2 base * height
}
}
}
```

Odata construit asamblajul (Shapes.dll), vom utiliza Ildasm (**I**ntermediate **L**anguage **D**is**AS**se**M**bler) pentru a-i vedea continutul.

Adaugam apoi o metoda, spre exemplu

```
using System Drawing;
public void Draw()
{
Pen p = new Pen(Color.Red);
}
```

Pentru ca proiectul sa poata fi compilat trebuie adaugata o referinta catre System.Drawing.dll (in Solution Explorer\ References, click dreapta si Add References).

Se observa ca pe langa asamblajele externe, manifestul contine o multime de informatii despre el insusi. Acestea se numesc atribute ale asamblajului. Se observa ca in Solution Explorer\ Properties se afla un fisier numit AssemblyInfo.cs. Acest fisier descrie atributele asamblajului, inclusiv versiunile sale. Cu Ildasm se pot vedea valorile acestor atribute.

Unul dintre atribute il reprezinta AssemblyVersion attribute. Versiunea unui asamblaj .NET are patru parti. Valorile atributului au urmatoarea semificatie: primul numar reprezinta Major Revision, al doilea Minor Revision, apoi Build Number si Revision. Primele doua numere prezinta o mare importanta in determinarea compatibilitatii versiunilor. Ultimele doua valori pot fi setate prin default de catre Visual Studio.

Compatibilitatea versiunilor

La executia unui program, .NET verifica daca versiunile sunt compatibile.

Asa cum se poate observa, manifestul unui asamblaj contine numarul versiunii asamblajului curent si numerele versiunii asamblajelor externe referite. Atunci cand .NET incarca un asamblaj extern, ii verifica manifestul pentru a compara vesiunile. Daca asamblajele au numerele Major sau Minor diferite atunci versiunile sunt incompatibile si asamblajul referit nu va fi incarcat.

Spre exemplu Shapes 1.1 nu este compatibil cu Shapes 2.0 2.1, 1.0 sau 1.2.

Ce se intampla atunci cand sistemul contine un program A care utilizeaza Shapes 1.0 si un program B care utilizeaza Shapes 1.1? De acest lucru se ocupa .NET prin intermediul unei caracteristici intitulate “executie diferentiata” (side by side execution) care face ca atat Shapes 1.0 cat si Shapes 1.1 sa fie ambele instalate pe acelasi calculator si sa fie disponibile programelor care necesita versiunile corespunzatoare.

Doua asamblaje avand aceleasi numere Major si Minor dar avand numarul Build diferit pot fi sau nu compatibile. La rulare, se presupune ca sunt compatibile astfel ca se permite ca acestea sa fie incarcate. Depine insa de dezvoltator daca modificarile aduse nu pericliteaza executia acestuia.

Apelarea asamblajelor

Sa presupunem ca asamblajul Shapes.dll este apelat de de programul de mai jos:

```
using System;
using Shapes;
namespace ShapeUser{
public class ShapeUser{
public static void Main(){
Circle c = new Circle(1.0F);
Console.WriteLine("Area of Circle(1.0) is {0}", c.Area());
}
}
}
```

Pentru ca programul sa poata fi compilat trebuie adaugata o referinta la asamblajul Shapes.dll Acest lucru se realizeaza din Solution Explorer, se selecteaza References, iar apoi prin click dreapta se deschide fereastra Add References, de unde se alege asamblajul dorit. Astfel, o copie a asamblajului este adaugata in directorul Debug. Programul poate acum fi construit, iar continutul asamblajului obinut poate fi analizat cu Ildasm.