Proiectarea Sistemelor Software Complexe

Curs 10 –Managed Extensibility Framework

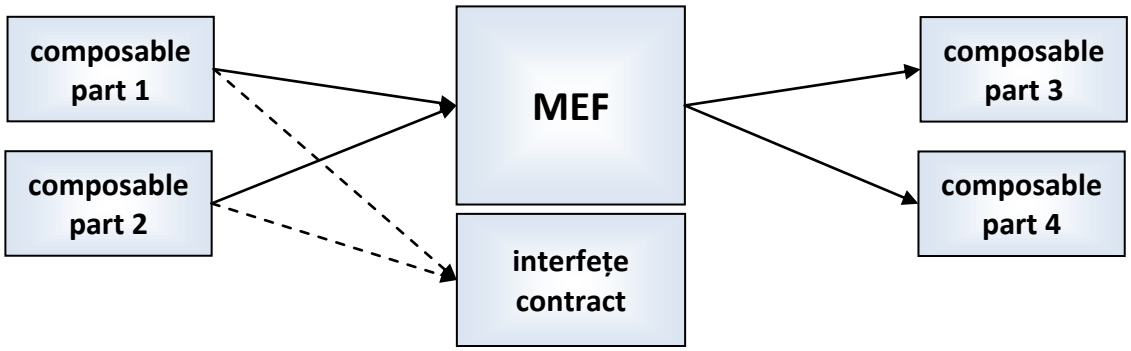
14.1 Introducere

Multe din sistemele software complex au nevoie să poată fi extinse cu noi funcționalități, respectiv cu noi module. Proiectarea unui sistem software extensibil nu este ușor de realizat, de aceea au fost dezvoltate o serie de platforme care facilitează proiectarea acestor sisteme software. Una din cele mai noi astfel de platforme este Managed Extensibility Framework (MEF). Platforma MEF permite dezvoltarea de aplicații .Net extensibile.

14.2 Prezentare Managed Extensibility Framework(MEF)

Din punctul de vedere al platformei MEF un sistem software este compus din așa numitele composable parts (părți ce pot fi compuse). O astfel de parte oferă servicii altor părți și la rândul ei consumă servicii oferite de alte părți. Din punctul de vedere al MEF faptul că aceste părți sunt dezvoltate odată cu sistemul software sau ulterior ca și componente terțe, nu reprezintă nici o diferență. Principalele concepte care trebuie înțelese pentru a putea folosi platforma MEF sunt:

- export – reprezintă un serviciu oferit de o parte; aceeași parte poate exporta unul sau mai multe servicii, în mod obișnuit o parte exportă un singur serviciu;
- import – reprezintă un serviciu consumat de o parte; o parte poate consuma unul sau mai multe servicii;
- contract – un contract este un identificator pentru un export sau un import; o parte care exportă un serviciu trebuie sa precizeze ca și contract un identificator de tip text care reprezintă contractul prin intermediul căruia o alta componentă care dorește să consume serviciul va putea importa acel serviciu; dacă la export nu este precizat un contract explicit, platforma MEF va folosi ca și contract numele tipului care este exportat;
- compunere – părțile sunt compuse de către MEF care le instanțiază și apoi face legătura între importuri și exporturi.



Figură1: Exemplu de arhitectură bazată pe MEF

În Figură1 este prezentat un exemplu de arhitectură care folosește MEF pentru a interconecta părțile componente ale sistemului software. Astfel sistemul software este compus din patru componente (composable parts), două componente exporta servicii (composable part 1 și composable part 2) care sunt importate de alte două componente (composable part 3 și composable part 4). În acest exemplu platforma MEF are rolul de-a face disponibile serviciile exportate de componentele composable part 1 și composable part 2 către componentele composable part 3 și composable part 4. Astfel componentele care exporta servicii implementează interfețele contract, iar componentele care importa serviciile vor obține referințe la servicii prin intermediul platformei MEF și vor putea apela funcționalitatea publică a acelor servicii definită prin intermediul interfețelor contract. Rolul platformei MEF este acela de a face transparent legătura între componente asigurând o decuplare totală a componentelor sistemului. Astfel platforma se ocupă de crearea instanțelor, iar singurul element de legătură între componentele sistemului software este reprezentat de interfețele contract care reprezintă o abstractizare a serviciilor. Cu alte cuvinte dacă se dorește o implementare diferită a unui serviciu nu trebuie decât să se realizeze o nouă implementare a interfeței care definește contractul acelui serviciu și să se exporte acea implementare prin intermediul platformei MEF.

14.3 Modelul de Programare al Platformei MEF

Modelul de programare prin care programatorii pot face uz de facilitățile MEF este unul bazat pe folosirea atributelor. În continuare vor fi prezentate principalele elemente ale platformei MEF.

Exportarea serviciilor implementate de o componentă se realizează prin decorarea unei clase sau a unei proprietăți cu atributul *Export*. În Cod 1 este prezentat un exemplu de folosire al atributului *Export*, în care atribut-ul este folosit pentru a exporta clasa *AesCryptographicAlgorithem*; clasa este exportată sub numele de contract dat de numele tipului *ICryptographicAlgorithem*.

```
[Export(typeof(ICryptographicAlgorithem))]
publicclassAesCryptographicAlgorithem:ICryptographicAlgorithem
{
    #region ICryptographicAlgorithem Members

publicstring Description
    {
get { return"AES cryptographic alorithem"; }
    }

publicbyte[] Encrypt(byte[] plainData, string password)
    {
Console.WriteLine("Encrypt using AES.");
returnnull;
    }

publicbyte[] Decrypt(byte[] encryptedData, string password)
    {
Console.WriteLine("Decrypt using AES.");
returnnull;
    }

    #endregion
}
```

Cod 1: Exemplu de folosire al atributului Export

Importarea serviciilor exportate de o componentă se realizează prin decorarea proprietăților cu atributul *Import, ImportMany* sau *ImportingConstructor*. Atributul Import se folosește atunci când există un singur serviciu exportat sub un anumit nume de contract, iar daca există mai multe servicii expirate sub

același nume de contract atunci se folosește atributul *ImportMany*. Atributul ImportConstructor permite importarea serviciilor prin intermediul constructorului. În Cod 2 este prezentat un exemplu in care se importă prin intermediul atributului *ImportMany* toate serviciile care sunt exportate sub același nume ca și numele interfeței *ICryptographicAlgorithem*.

```
[ImportMany(AllowRecomposition = true)]
publicIEnumerable<ICryptographicAlgorithem> Algorithems { get; set; }

. . .

//ulterior proprietatea poate fi folosită, ea fiind instanțiată de platforma MEF
foreach (ICryptographicAlgorithem algorithem in program.Algorithems)
{
Console.WriteLine("Use algorithem {0}.", algorithem.Description);
byte[] encData = algorithem.Encrypt(newbyte[] { 1, 2, 3 }, "password");
byte[] decData = algorithem.Encrypt(encData, "password");
}
```

Cod 2: Exemplu de folosire al atributului ImportMany

Pentru ca importurile și exporturile să poată fi satisfăcute este necesar să fie creat un CompositionContainer și să se apeleze funcția compose. În Cod 3 este prezentată o secvență de cod care compune instanța curentă cu toate librăriile .net aflate în directorul curent de lucru.

```
///<summary> Compose parts. </summary>
privatevoid Compose()
{
var directoryCatalog = newDirectoryCatalog(Environment.CurrentDirectory);
var container = newCompositionContainer(directoryCatalog);
    container.ComposeParts(this);
}
```

Cod 3: Exemplu de invocare a compunerii părților unui sistem software

Bibliografie

[1] http://mef.codeplex.com/
[2]http://msdn.microsoft.com/en-us/magazine/ee291628.aspx