EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

.NET Framework Nedir?

.NET Framework Nedir?

.NET Framework, Microsoft tarafından geliştirilen, açık internet protokolleri ve standartları üzerine kurulmuş komple bir uygulama geliştirme platformudur.

Buradaki uygulama kavramının kapsamı çok geniştir. Bir masaüstü uygulamasından bir web tarayıcı uygulamasına kadar her şey bu platform içinde düşünülmüştür ve desteklenmiştir. Bu uygulamaların birbirleriyle ve geliştirildiği ortam farketmeksizin dünyadaki tüm uygulamalarla iletişimi için kolayca web servisleri oluşturulmasına imkân verilmiştir. Bu platform, işletim sisteminden ve donanımdan daha üst seviyede taşınabilir olarak tasarlanmıştır.

.Net mimarisi, ortak bir yürütme ortamı (runtime environment), ortak bir değişken tür sistemi, ve birbirleriyle bağlantılı kütüphanelerden oluşur.

EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

.NET Vizyonunun Oluşumu

.NET Vizyonunun Oluşumu

.NET platformu, Microsoft tarafından geliştirilmiş ve platformdan bağımsız bir şekilde uygulama geliştirilmesini sağlayan bir ortamdır. Sağladığı çoklu dil desteği sayesinde programcıların tek bir dile bağımlı kalmadan (hatta farklı dilleri bir arada kullanmasını sağlayarak) değişik tipte uygulamalar geliştirmelerine olanak sağlar. Masaüstü (Windows, konsol), web, mobil, web servisi, windows servisi, remoting söz konusu uygulama çeşitlerinden bazılarıdır.

EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

.NET'in Getirdiği Çözümler

.NET'in Getirdiği Çözümler

- **Varolan kodlarla tam çalışabilirlik desteği:** Varolan COM binary'leri ile yeni .NET binary'leri bir arada uyumlu olarak çalışabilirler, ayrıca tam tersi de geçerlidir. Aynı zamanda .NET kodundan C-tabanlı kütüphanelerin çağrılmasına izin verilir.

- **Tüm .NET dilleri tarafından paylaşılan ortak bir çalışma zamanı:** .NET ortamında program geliştirirken (kullanılan dilden ve uygulama tipinden -web,masaüstü…- bağımsız olmak üzere) çalışma zamanı prensiplerini belirleyen ve temellerini sağlayan Ortak Çalışma Zamanı (Common Language Runtime), daha önce uygulama geliştiricinin düşünmek zorunda olduğu birçok işin üstesinden gelir (Bellek yönetimi (Memory management), tip güvenliği (Type safety), istisna yönetimi (Exception handling) vb…).

- **Çoklu dil desteği:** Microsoft radikal bir karar alarak CLR ile uyumlu her .NET dilinin kullanılmasına olanak sağlıyor. Visual Studio 2010 ile gelen yazılım geliştirme kitinde C#, VB.NET, J#.NET ve C++.NET kullanarak program geliştirilebiliyor. Öte yandan .NET ortamına entegrasyonu tamamlanmış 50'den fazla programlama diliyle de uygulama geliştirilebilir. (Örnek: Delphi.NET, Perl for .NET…)

- **Tüm .NET dilleri tarafından paylaşılan ortak temel sınıf kütüphanesi:** Artık karmaşık API çağrıları sona erdi. .NET ile birlikte uygulama geliştiricinin hizmetine sunulan 3500'den fazla sınıftan oluşan zengin kütüphane, daha hızlı program geliştirme imkanı ve bütün .NET dilleri tarafından kullanılan tutarlı bir nesne modeli sunuyor.

- **Programlama modelinden bağımsız uygulama geliştirme ortamı:** Tek bir uygulama geliştirme ortamı (Visual Studio 2010) kullanarak ASP.NET, masaüstü form (windows), mobil, web servisi ve remoting uygulamaları geliştirilebilir.

- **Basitleştirilmiş masaüstü uygulama geliştirme ve yayınlama modeli:** .NET ortamında geliştirilen bir masaüstü uygulaması, herhangi bir windows işletim sisteminin kurulu olduğu makinede çalıştırılabilir, gereken tek şey .NET Framework'ünün kurulu olmasıdır. Ayrıca sistem kayıt defterine (registry) yazılmasına gerek yoktur. Bunun yanında .NET aynı makinede bir .dll'in farklı versiyonları ile çalışılmasına izin verdiği için ".dll cehennemi" (".dll hell") adı verilen durum oluşmamaktadır.

EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

.NET'in Yapı Taşları

.NET'in Yapıtaşları

.NET'in sağladığı bazı avantajlar incelendi, şimdi de bu avantajları hayata geçirmek için gereken (birbirleriyle bağlantılı) üç yapıtaşı incelenecek: **CLR**, **CTS** ve **CLS**. Yazılım geliştiricinin bakış açısıyla .NET yeni bir çalışma zamanı ve çok yönlü bir temel sınıf kütüphanesi olarak görülebilir.

Ortak Çalışma Zamanı (Common Language Runtime -CLR-)

Çalışma zamanı ortamı, **Common Language Runtime (ortak çalışma zamanı)** olarak adlandırılır ve CLR kısaltmasıyla anılır. CLR'ın birincil rolü .NET tiplerinin yerini öğrenmek, bu tipleri kendi ortamına yüklemek ve yönetmektir. CLR ayrıca bellek yönetimi ve tip güvenlik kontrollerini yerine getirmek gibi birçok alt seviye ayrıntıdan da sorumludur.

Ortak Tip Sistemi (Common Type System -CTS-)

.NET platformunun bir diğer yapıtaşı **Common Type System (ortak tip sitemi)**'dir ve kısaca CTS olarak anılır. CTS spesifikasyonları, çalışma zamanı tarafından desteklenen bütün veri tipleri ve programlama yapılarını tanımlar, bu yapıların birbirleriyle nasıl etkileşeceklerini açıkça belirtir ve .NET metadata formatında nasıl temsil edileceklerinin ayrıntılarını belirler (metadata hakkında ayrıntılı bilgiye bir sonraki bölümde erişilebilir). Böylece .NET destekli tiplerin, aynı veri tiplerini kullanabilmesi sağlanabilmektedir.

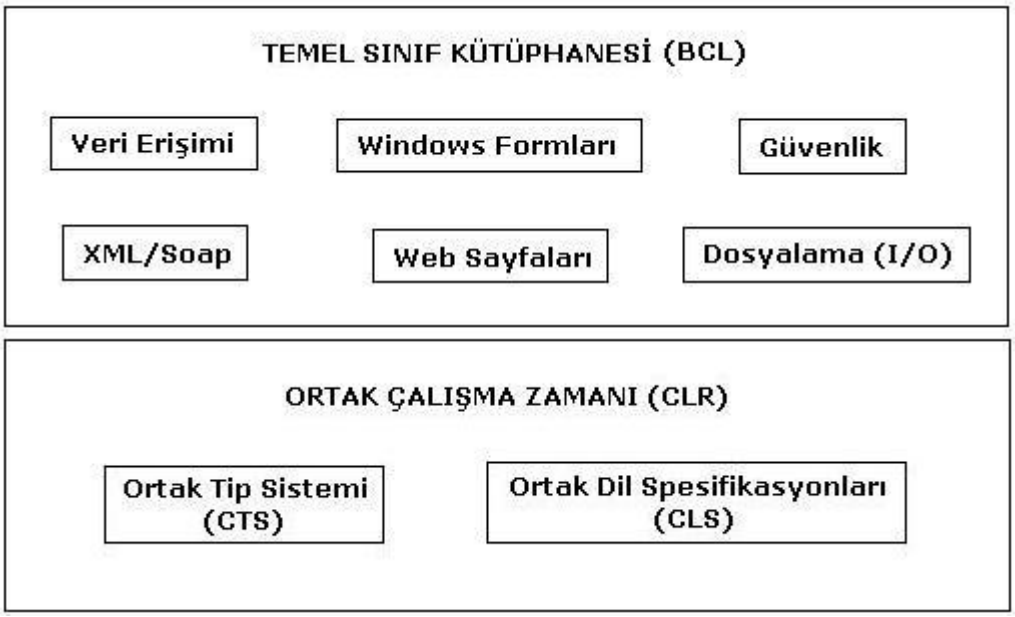
Ortak Dil Spesifikasyonları (Common Language Spesification -CLS-)

.NET tabanlı bir programlama dili, ortak tip sistemi CTS tarafından tanımlanan her bir özelliği desteklemeyebilir. **Common Language Specification (ortak dil spesifikasyonu),** tüm .NET dillerinin orta noktada buluşabileceği ortak tip ve programlama altyapısını tanımlayan yönerge listesidir. Dolayısıyla CLS uyumlu özelliklere sahip bir .NET tipi geliştirilirse, bu tipi diğer bütün .NET dillerinin kullanabileceği garanti edilmiş olur. Tersine CLS sınırlarının dışında bir veri tipi ya da programlama yapısı oluşturulursa, bu kod kütüphanesi ile bütün .NET dillerinin sağlıklı bir şekilde etkileşebileceği garanti edilemez.

Temel Sınıf Kütüphanesi (Base Class Library)

CLR ve CTS/CLS spesifikasyonlarına ek olarak .NET platformu, tüm .NET programlama dillerinin kullanabileceği **Base Class Library'i (temel sınıf kütüphanesi)** sunar. Her biri belli bir görevi yerine getirmekle sorumlu olan sınıflardan oluşan bu kütüphane hem temel işler (thread -kanal-, dosya giriş çıkış,grafiksel görünüm…) için kullanılacak tipleri içerir hem de gerçek hayat uygulamalarının ihtiyaç duyacağı birçok servise destek sağlar. Örneğin temel sınıf kütüphanesinin bize sağladığı tipler veritabanı erişimi, xml etkileşimleri, programatik güvenlik konularını ele almayı ve web,masaüstü ya da konsol tabanlı kullanıcı ara yüzleri geliştirmeyi çok kolaylaştırır.

.NET platformunun yazılım geliştiricilere sunduğu bu koleksiyonun güzel yanlarından biri de kullanılmasının son derece kolay olmasıdır. Kullanılan isimler o kadar açıklayıcıdır ki temel ingilizce bilgisine sahip birisi için sezgisel olarak hangi üyenin kullanacağının bulunması çok fazla zaman almamaktadır.



.NET Framework temel bileşenleri : CLR,CTS,CLS ve BCL

C# : Geçmişi Olmayan Bir Dil

.NET'in önceki teknolojilere göre ne kadar radikal bir proje olduğunun göstergesi olarak Microsoft, bu yeni platforma özel yeni bir programlama dili geliştirdi. Bu dil modern nesne yönelimli dizayn metodolojisi üzerine kuruldu ve Microsoft C#'ı geliştirirken yıllardır nesne yönelimli prensiplere sahip benzer dillerden elde ettiği tecrübelerden faydalandı. Sonuç olarak ortaya sözdizimi son derece temiz, öğrenmesi ve yazması kolay, ayrıca güçlü ve esnek bir dil çıktı.

C#'ın Microsoft.NET platformu ile gelmesi ile ilgili olarak anlaşılması gereken önemli noktalardan biri, sadece .NET çalışma zamanında çalışacak kod üretmesidir. C# hiçbir zaman COM ya da Win32 API uygulaması geliştirirken kullanılamaz. Teknik bir ifadeyle .NET çalışma zamanında işlenecek kodu tanımlayan terim **managed code** (yönetimli kod) ; yönetimli kod içeren binary birimin .NET dünyasındaki terim karşılığı ise **assembly** dir. Aksine doğrudan .NET çalışma zamanı tarafından işlenmeyen kod ise **unmanaged code** olarak adlandırılır. Örneğin C,C++ gibi diller yardımıyla geliştirilebilirler.

EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

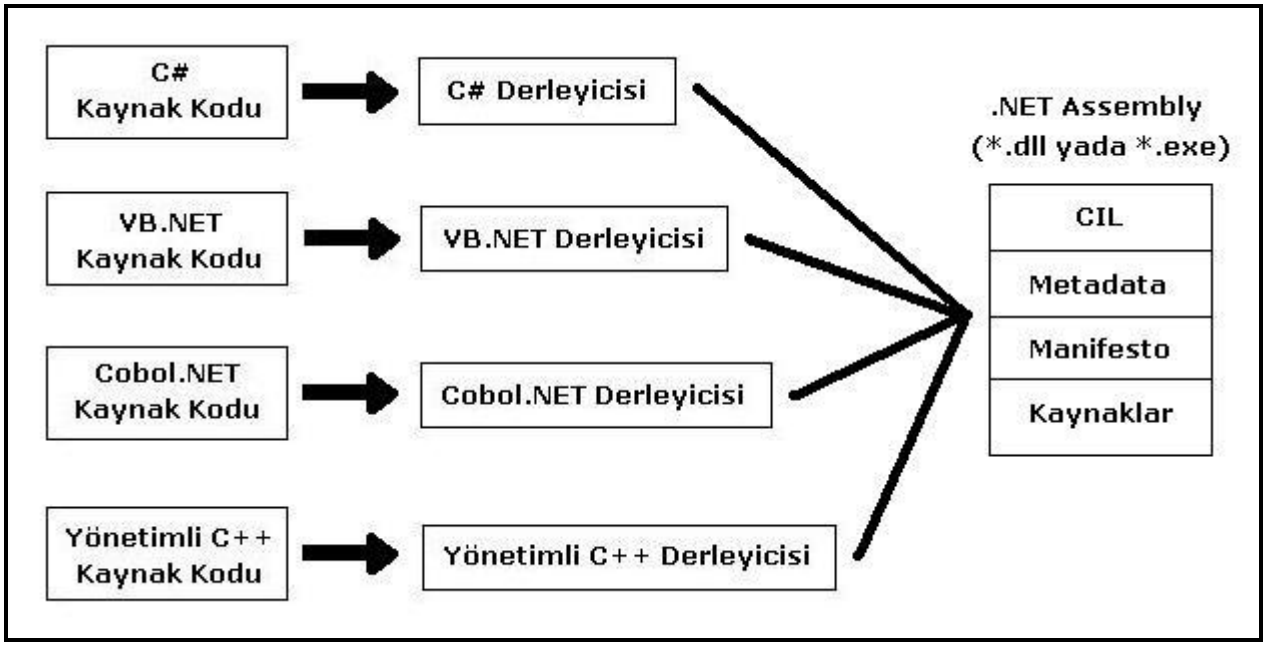
Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

Assembly Nedir?

Assembly Nedir?

Hangi .NET dilinin kullanıldığından bağımsız olarak .NET binary'leri bir dosya uzantısı alırlar (*.dll ya da *.exe). Burada dikkati çekilmesi gereken nokta bu binary'lerin işletim sistemine özel komutlar içermemesidir. Bunun yerine .NET binary'leri, platformdan bağımsız **Common Intermediate Language(CIL)** adındaki ara dili içerirler.



Tüm .NET derleyicileri kaynak kodu assembly içerisinde CIL'e derler.

IL kısaltması ile ilgili bir noktaya dikkat çekmek gerekiyor. .NET'in geliştirilme aşamasında IL'in resmi adı "Microsoft intermediate language" (MSIL) iken final sürümüyle birlikte bu terim "Common intermediate language" (CIL) olarak değiştirildi. Dolayısıyla kaynaklarda geçen MSIL ve CIL aynı kavramı işaret etmektedir.

.NET ortamındaki bir programlama dilinin derleyicisi kullanılarak bir .dll ya da .exe oluşturulduğunda bu bir **assembly** içerisine koyulur. Daha önce bahsedildiği gibi assembly, CIL kodu içerir ve bu kod, ihtiyaç duyulana kadar platforma özel bilgilere derlenmez. Burada ihtiyaç duyulan nokta ile kastedilen, .NET çalışma zamanı tarafından kullanılmak üzere başvurulan IL kod bloğudur (Metot çağrısı gibi)

Şunu da eklemek gerekir ki binary dosya (.exe ya da .dll uzantılı) ile assembly tektir ve aynı kavramlardır; alt seviye bir programlama dili olan Assembly ile sadece isim benzerliği vardır. Ayrıca çalıştırılabilir kod (.exe) ve kütüphane kodu (.dll) aynı assembly yapısına sahiptir ve terim olarak her ikisi de assembly olarak çağrılır. Aralarındaki tek gerçek fark çalıştırılabilir assembly (.exe) program için ana giriş noktası (Main metodu) içerirken, kütüphane assembly'si (.dll) bunu içermez.

Metadata ve Manifesto

Assembly, CIL dışında bir de **metadata** içerir. **Tip metadata**, binary içerisindeki her bir tipin ayırt edici özelliklerini tanımlar. Daha aydınlatıcı olması açısından şöyle bir örnek verelim: Mevcut evimizden başka bir eve taşınırken eşyaları kutulara koyarız. Eğer kutuların üzerine tek tek içerisinde neler olduğunu bir kağıda yazıp yapıştırırsak yeni evimizde açarken aradığımızı kolayca buluruz. Metadata, kutunun üzerindeki bu listedir ve assembly içerisindeki .dll ya da .exe'nin sahip olduğu tüm tipler hakkında bilgi içerir. Örneğin X adında bir sınıf varsa, tip metadası X sınıfının türediği sınıf, varsa hangi arayüzleri uyguladığı gibi ayrıntıları taşırken, X tipinin her bir üyesinin tüm tanımlamalarını da içerir.

Bir assembly'de aynı zamanda kendisini tanımlayan bilgileri içeren **assembly metadata** bölümü bulunur. Assembly adı, versiyonu, kültür bilgisi, kısa bir açıklama, başka assembly'lere olan referanslar gibi bilgilerin tamamına assembly metadata denir ve **manifesto** adıyla assembly içerisinde yer alır. Bir assembly'nin assembly metadatası taşımasının arkasındaki gerçek, içerisindeki kodu çağıran uygulama ya da diğer assembly'lerin o assembly'yi nasıl kullanacaklarını öğrenmek için registry'ye ya da başka bir veri kaynağına başvurmalarına gerek kalmamasıdır.

Öyleyse bir assembly içerisindeki kaynak kodun CIL karşılığı, tip metadatası, manifesto bilgisi ve kaynaklar yer alır.

Dört bileşeni ile bir .NET assembly'si

EĞİTİM :

.NET UYGULAMA GELİŞTİRME PLATFORMU

Bölüm :
.NET Uygulama Geliştirme Platformu

Konu :

Ortak Ara Dilin Rolü

Ortak Ara Dilin (Common Intermediate Language) Rolü

.NET assembly'leri hakkında elde edilen bilgiler ışığında platformun ortak ara dilinin (CIL) rolü biraz daha ayrıntılı incelenebilir. CIL, herhangi bir platform-özel direktif setinin yerine geçmiştir. Seçilen .NET tabanlı dilin hangisi olduğundan bağımsız, ilgili derleyici CIL direktifleri üretir. Örnek olarak aşağıdaki C# kodu basit bir hesap makinesini modelliyor. Şimdilik sözdizimi ile çok ilgilenmeye gerek yok; ancak HesapMak sınıfının içerisindeki Topla() metodunun formatına dikkat:

```
//Hesap.cs
using System;
namespace HesapMakinesiOrnek
{
  //C# hesap makinesi
  public class HesapMak
  {
    public int Topla(int x, int y)
    {
      return x + y;
    }
  }
  //Bu sınıf, programın giriş noktasını içerir.
  class HesapMakUygulamasi
  {
    static void Main(string[] args)
    {
      HesapMak hesap = new HesapMak();
      int cevap = hesap.Topla(23, 41);
      Console.WriteLine("23 + 41 = {0}", cevap);
      Console.ReadLine();
    }
  }
}
```

C# derleyicisi (csc.exe) ile bu kaynak kod dosyası derlendiğinde;

- CIL direktifleri
- Manifest
- HesapMak ile HesapMakUygulamasi sınıflarının her ayrıntısını tanımlayan metadata bilgisi

içeren tek dosyalı (single-file) bir *.exe assembly elde ederiz. Bu assembly ildasm.exe ile açılırsa Topla() metodunun CIL kullanılarak aşağıdaki gibi temsil edildiği görülür. (Visual Studio 2010 Tools altındaki Visual Studio 2010 Command Prompt'a ildasm.exe yazılarak bu araç açılabilir ve ilgili .dll ya da .exe'nin yolu gösterilir) :

```
.method public hidebysig instance int32  Topla(int32 x,
```

```
                                int32 y) cil managed
{
  // Code size      9 (0x9)
  .maxstack  2
  .locals init ([0] int32 CS$1$0000)
  IL_0000:  nop
  IL_0001:  ldarg.1
  IL_0002:  ldarg.2
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s       IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
} // end of method HesapMak::Topla
```

Bu metot için üretilen CIL kodundan pek bir şey anlaşılmaması çok önemli değildir. Burada üzerinde durulması gereken nokta şu ki; C# derleyicisi işletim sistemine özel kodlar değil, CIL üretir.

Tekrarlamakta fayda var, bu özellik bütün .NET tabanlı derleyiciler için geçerlidir. Bunu gösterebilmek için aynı uygulamanın C# yerine Visual Basic .NET (VB.NET) kullanılarak yazıldığını varsayalım :

```
'Hesap.cs
Imports System

Namespace HesapMakinesiOrnek
   'Bir VB.NET 'Modülü'sadece static üyeler içeren bir sınıftır.
   Module HesapMakUygulamasi
      Sub Main()
         Dim hesap As New HesapMak
         Dim sonuc As Integer
         sonuc = hesap.Topla(23, 41)
         Console.WriteLine("23 + 41 = {0}", sonuc)
         Console.ReadLine()
      End Sub
   End Module

   //C# hesap makinesi
   Class HesapMak
      Public Function Topla(ByVal x As Integer, ByVal y As Integer) As Integer
         Return x + y
      End Function
   End Class
End Namespace
```

Eğer Topla() metodunun CIL direktiflerine bakacak olursak C# kaynak kodu için üretilenlerin VB.NET için üretilenlerle aynı olduğunu görürüz.

```
.method public instance int32  Topla(int32 x,
                                     int32 y) cil managed
{
 // Code size      9 (0x9)
 .maxstack  2
 .locals init ([0] int32 Topla)
 IL_0000:  nop
 IL_0001:  ldarg.1
 IL_0002:  ldarg.2
 IL_0003:  add.ovf
 IL_0004:  stloc.0
 IL_0005:  br.s       IL_0007
 IL_0007:  ldloc.0
 IL_0008:  ret
} // end of method HesapMak::Topla
```

CIL'in Yararları

Bu noktada kaynak kodun platforma özel komut setine (makine diline) değil de CIL'e derlenmesinden tam olarak ne elde edildiği merak edilebilir. Bunun cevabı öncelikle dil entegrasyonudur. Yukarıda da gördüğümüz gibi bütün .NET tabanlı derleyiciler neredeyse aynı CIL kodunu üretiyorlar. Dolayısıyla bütün diller iyi-tanımlanmış bu binary ortamda aralarında anlaşabileceklerdir. Terimler şu an için yabancı gelebilir; ancak aşağıdaki örnekler verilebilir

- Bir dilde yazılmış sınıf, başka dilde yazılmış bir sınıfın üyelerini kalıtım yoluyla kullanabilir.
- Bir sınıf hangi dilde geliştirildiğinden bağımsız olarak başka bir sınıf örneğini içerebilir.
- Nesneler ya da nesne referansları metotlar arasında parametre olarak geçirilebilir.
- Farklı dillerde yazılmış metotları çağırırken hata ayıklayıcı ile metot çağrımları arasında gezilebilir; yani farklı dillerdeki kaynak kodları arasında adım adım ilerlenebilir.

Ayrıca CIL'in platformdan bağımsız olmasından yola çıkarak .NET Framework'ün de platformdan bağımsız olduğunu söyleyebiliriz. (Tek bir kod bloğunun sayısız işletim sistemi üzerinde çalışabilmesi) Bu platform-bağımsızlık Windows işletim sistemlerinde geçerli olmakla beraber bir-çok Windows olmayan platformda deneme aşamasında da olsa uygulamaları mevcuttur. (Mono ve Portable.NET projeleri) Özet olarak CIL'in getirdiklerine bakılınca işin en güzel yanı .NET'in yazılım geliştiriciye hangi dil ile bunu yapmak istediği seçeneğini sunmasıdır.

CIL'in Platforma Özel Koda Derlenmesi

Herhangi bir dilde yazılmış uygulamanın bir bilgisayarda çalışması için mutlaka bilgisayarın anlayacağı komutlara dönüştürülmesi gereklidir. .NET assembly'leri platforma özel kod yerine CIL içerdiğinden dolayı kullanılmadan önce makine koduna derlenmelidirler. CIL'i uygulamanın çalıştığı makine için anlamlı talimatlara dönüştüren derleyiciye **just-in-time compiler (tam zamanında derleyici)** adı verilir ve aynı zamanda **jitter** olarak da anılır. .NET çalışma zamanı ortamı, her biri üzerinde çalıştığı işletim sistemi için optimize edilmiş, her CPU için bir JIT derleyicisi kullanır ve bu optimizasyonu otomatik yapar.

Örneğin cep bilgisayarı gibi küçük akıllı cihazlarda yayınlanmak üzere bir .NET uygulaması geliştiriliyorsa, düşük bellek ortamında çalışması için uygun jitter kullanılır. Diğer yandan geliştirilen assembly bir yedek

sunucuda dağıtılacaksa (bellek çoğu zaman sorun olmayacaktır) jitter yüksek bellekli ortamda işlevini yerine getirmesi için optimize edilecektir. Bu yolla yazılım geliştiriciler, etkin bir şekilde "tam-zamanında-derlenecek" ve farklı mimariler kullanılarak çalıştırılacak kod blokları yazabilirler.

Bütün bunlara ek olarak kullanılan jitter ile CIL komutları uygun makine kodlarına derlendiğinde, sonuçlar uygun bir yolla hedef işletim sisteminin belleğinde saklanacaktır. Buna göre *DosyayaYaz()* isimli metoda bir çağrı yapılırsa, ilk çağrıda CIL komutları platforma özel kodlara derlenecek ve sonraki kullanımlar için hafızada tutulacaktır. Dolayısıyla DosyayaYaz() metodunun ikinci çağrılışında CIL'i yeniden derlemeye gerek kalmayacaktır.