# Het .NET framework in vogelvlucht

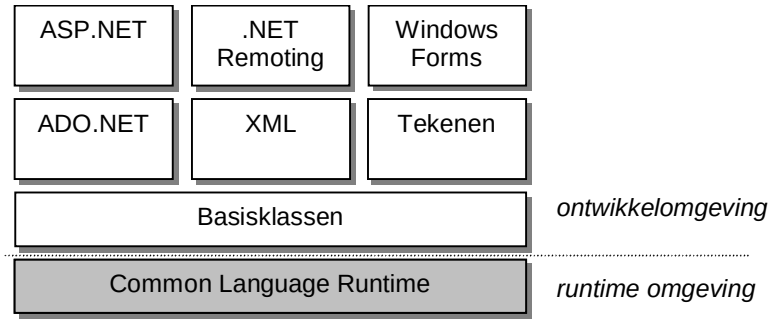
## De basis voor .NET applicaties

**Danny Greefhorst**

**Het .NET framework is de basis voor het .NET platform en bestaat zowel uit een ontwikkel- als een runtime-omgeving. De ontwikkelomgeving bevat een grote hoeveelheid standaardbibliotheken waarmee vrijwel alle soorten applicaties kunnen worden ontwikkeld; van webdienst tot Windowsapplicatie. De runtime-omgeving biedt applicaties een object-georiënteerd model waarin allerlei standaard diensten worden geboden. Dit artikel geeft een overzicht van beide kanten van het .NET framework.**

### Inleiding

Het eerste deel van het .NET framework is een uitgebreid raamwerk van standaard klassen voor het ontwikkelen van allerlei soorten .NET applicaties (zie figuur 1). Zo zijn er het nieuwe ASP.NET voor het ontwikkelen van webapplicaties en webdiensten, .NET Remoting voor gedistribueerde componenten en de Windows Forms klassen voor applicaties die gebruik willen maken van Windows-specifieke eigenschappen. Voor al deze typen van applicaties kan gebruik gemaakt worden van veel basisklassen voor onder andere databasetoegang, XML, tekenen, security, netwerktoegang, stringmanipulaties, reflectie en COM-integratie.



**Figuur 1** .NET framework

Het andere belangrijke deel van het .NET framework is de Common Language Runtime (CLR); een omgeving voor het uitvoeren van .NET applicaties. Het bijzondere van deze omgeving is dat hij gebaseerd is op een programmeertaal-onafhankelijk tussenformaat, de zogenaamde Intermediate Language (IL), waardoor integratie tussen programmeertalen eenvoudig wordt. Elke programmeertaal die .NET begrijpt, en dat zijn niet alleen de door Microsoft geleverde talen zoals Visual Basic en C#, kan direct gebruik maken van klassen geschreven in andere talen. De speciaal voor .NET ontwikkelde IL kan snel naar machine-afhankelijke code worden vertaald door bijvoorbeeld een JIT-compiler waardoor de snelheid goed blijft. Naast een executieomgeving biedt de CLR een aantal toegevoegde-waarde diensten voor bijvoorbeeld het beheren van geheugen, processen en threads, excepties, beveiliging en metadata. In het bijzonder biedt de CLR *garbage collection* waarbij objecten die niet meer worden gebruikt automatisch worden opgeruimd. .NET applicaties kunnen executeren in elke omgeving waarvoor een CLR implementatie beschikbaar is, onafhankelijk van de onderliggende hardware en software.

### De ontwikkelomgeving

Het eerste belangrijke deel van de .NET ontwikkelomgeving is ASP.NET. Microsoft heeft met ASP.NET een geheel nieuwe implementatie van het ASP webserver raamwerk gemaakt waarmee naast webapplicaties ook webdiensten kunnen worden gemaakt. Webdiensten zijn componenten die via standaard protocollen over het Internet kunnen communiceren. Een belangrijk onderdeel van deze protocolstack is SOAP (Simple Object Access Protocol), een protocol voor het versturen van XML-gebaseerde berichten. In .NET is voluit ondersteuning voor SOAP en is het heel eenvoudig om een ASP.NET pagina te definiëren met daarin een webdienst (zie figuur 2). Deze webdienst kan in een willekeurige .NET programmeertaal zijn geschreven en hoeft alleen met het speciale *WebMethod* attribuut aan te geven welke operaties via SOAP beschikbaar moeten zijn.

```
<%@ WebService Language="C#" Class="Hello" %>
using System.Web.Services;
public class Hello : WebService {
  [ WebMethod ]
  public string helloWorld() {
    return "Hello World";
  }
}
```

**Figuur 2** Voorbeeld SOAP-server, geschreven in C# met ASP.NET

Natuurlijk is het ook nog gewoon mogelijk om webapplicaties in ASP.NET te definiëren; de daarvoor beschikbare gereedschapskist is alleen verder uitgebreid. Naast standaard *HTML controls* zijn er bijvoorbeeld speciale *web-* en *validatiecontrols*. Webcontrols zijn visuele componenten die zich automatisch aanpassen aan de gebruikte webbrowser. Een webpagina is hierdoor automatisch geschikt voor zowel desktop als handheld apparaten, zoals mobiele telefoons. Validatiecontrols geven de mogelijkheid om door de gebruiker

ingevoerde gegevens snel en eenvoudig te kunnen valideren. Zo zijn er controls om te bepalen of een waarde in een bepaald bereik ligt, ingevuld is, aan een reguliere expressie voldoet of op een bepaalde manier is gerelateerd aan een andere control. Ook is er een speciaal control waarmee een samenvatting van de validatie wordt gegeven.

Een verbetering ten opzichte van het oude ASP is dat web-pagina's nu worden gecompileerd waardoor ze een stuk sneller zijn geworden. Verder is er in ASP.NET beter nagedacht over caching, sessiebeheer en beveiliging. Zo wordt de toestand van een sessie bewaard als de server crashed en kan toestandsinformatie door meerdere machines worden gedeeld, waarvoor speciale synchronisatiemechanismen beschikbaar zijn.

### Windows Forms

Als applicaties voluit gebruik willen maken van de grafische kracht van Windows dan is *Windows Forms* de aangewezen keuze. Windows Forms is een verzameling van klassen voor het maken van grafische gebruikersinterfaces. De geheel object-georiënteerde verzameling van formulieren en *controls* kunnen gebruik maken van specifieke GDI+ eigenschappen zoals die aanwezig zijn in Windows 2000. Handig is dat formulieren nu ook kunnen erven van andere formulieren waarbij basis-eigenschappen in algemene formulierklassen gedefinieerd kunnen worden. Controls zijn flexibel in formulieren te positioneren middels *anchoring* en *docking* waarbij ze zich automatisch aanpassen aan de grootte van het formulier.

### .NET Remoting

Voor het ontwikkelen van gedistribueerde applicaties is *.NET Remoting* het aangewezen ontwikkelgereedschap. Met .NET Remoting is het mogelijk om op een volwaardige manier met andere objecten in het netwerk te communiceren en kunnen zelfs hele objecten worden verstuurd. Objecten kunnen zowel synchroon als asynchroon communiceren en gebruik maken van verschillende transportkanalen en berichtformatteringen. Zo kan er gekozen worden voor een HTTP-kanaal met SOAP als formattering of het TCP-kanaal met een binaire formattering als snelheid echt van belang is. Objecten kunnen door de client of door de server worden geactiveerd. De levensduur van het eerste soort objecten wordt beheerd in de vorm van een *lease* die bij beëindiging zorgt voor het opruimen van het object. Server-geactiveerde objecten kunnen eenmalig of per aanroep worden geactiveerd. Bij het creëren van een object wordt er een lokaal proxy object gecreëerd dat zo slim is dat het voor lokale communicatie een directe aanroep doet en voor server-geactiveerde objecten pas over het netwerk communiceert bij het aanroepen van de eerste operatie. Communicatie is tenslotte in verre mate te beïnvloeden door het definiëren van eigen formatteerders, transportkanalen en proxies. In figuur 3 is een voorbeeld .NET Remoting client weergegeven die communiceert via het TCP-kanaal.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels.TCP;

public class Client {
  public static int Main(string [] args) {
    TCPChannel chan = new TCPChannel(8086);
    ChannelServices.RegisterChannel(chan);
    Hello hello = (Hello)Activator.GetObject(
      typeof(Hello), "tcp://server:8085/Hello");
    System.Console.WriteLine(hello.helloWorld());
  }
}
```

**Figuur 3** Voorbeeld .NET Remoting client

### ADO.NET

Voor toegang tot databases kan de ontwikkelaar gebruik maken van het nieuwe ADO.NET. Deze databasetoegangsinterface lijkt nog wel op het oude ADO maar is nu gebaseerd op XML. Naast directe databasetoegang is het mogelijk een gehele gegevensverzameling uit een database op te halen en lokaal als database te benaderen, wat de schaalbaarheid van applicaties vergroot. De gegevensverzameling wordt daarvoor in XML overgezonden en kan op verzoek ook in XML vertaald worden (en vice versa) waardoor gebruik gemaakt kan worden van de grote hoeveelheid beschikbare XML-tools op de markt. Er is lokaal een rijke toegang tot de gegevens mogelijk waarbij door de relaties tussen tabellen kan worden genavigeerd, tabellen kunnen worden gesorteerd en gefilterd, en events worden gegenereerd bij het wijzigen van de gegevens. Nadat lokaal wijzigingen in de gegevensverzameling zijn doorgevoerd kan weer worden gesynchroniseerd met de centrale database.

### XML

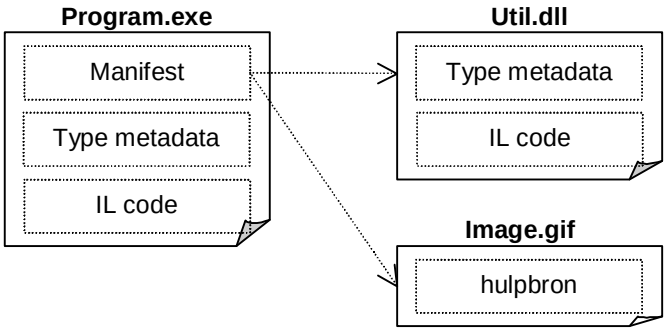
Het laatste belangrijke deel van de .NET ontwikkelomgeving is de grote hoeveelheid standaard klassen voor ondersteuning van XML. Hierbij zijn alle belangrijke XML-standaarden zoals XML Namespaces, XML Schema, XPath, XSL en DOM vertegenwoordigd. Ook het op XML gebaseerde SOAP, en het de daarbij behorende WSDL (Web Service Description Language) is in het .NET framework geïntegreerd. Het creëren, lezen en manipuleren van XML-gebaseerde gegevens is dan ook geen probleem.

## De runtime-omgeving

Het tweede deel van het .NET framework is de runtime-omgeving die bekend staat als de Common Language Runtime (CLR). De CLR bestaat uit een aantal belangrijke onderdelen waaronder het IL dat wordt gebruik als algemene voertaal en het Common Type System (CTS) waarmee deze taal betekenis krijgt. Verder is er de Common Language Specification (CLS) voor het uitwisselen van IL, metadata voor het beschrijven van IL en een Virtual Execution System (VES) voor het uitvoeren van IL. Assemblies zijn

de eenheid van installatie, security, versionering en zichtbaarheid voor .NET applicaties. Een assembly is niet zozeer een fysieke eenheid maar meer een logische eenheid die kan bestaan uit meerdere portable executables (PE) waarin de IL is verpakt, naast eventuele andere benodigde bestanden. Een voorbeeld van een assembly is weergegeven in figuur 4.




**Figuur 4** Assembly bestaande uit drie bestanden

### Intermediate Language

IL is de tussencode die door alle .NET compilers wordt gegenereerd. Alhoewel het mogelijk is IL direct te interpreteren is het sneller als het eerst door een compiler of JIT compiler naar machine-afhankelijke code is vertaald. Optimaliserende compilers kunnen een speciale subset van IL genaamd *OptIL* genereren die extra annotaties bevat waardoor snel machine-afhankelijke code kan worden gegenereerd. Er is een tekstueel formaat voor IL gespecificeerd dat qua syntax het midden houdt tussen assembly-language en C# (zie figuur 5). Dit tekstformaat kan direct door de IL-assembler worden vertaald naar een assembly en zou gebruikt kunnen worden door een compiler als uitvoerformaat. Omgekeerd is er ook een disassembler die een assembly kan omzetten naar een tekstuele IL-representatie, waardoor roundtrip engineering mogelijk wordt. Daarnaast kan IL ook direct binair via de reflectieklassen in het .NET raamwerk worden gegenereerd.

```
.assembly 'hello.exe' { … }

…
.method public instance void helloWorld() il managed {
  .maxstack 1
  ldstr "Hello World"
  call void System.Console::WriteLine(class System.String)
  ret
}
…
```

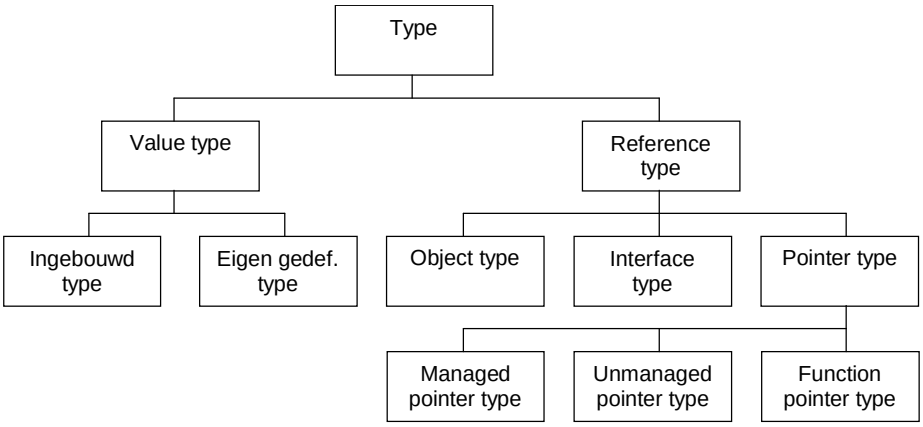
**Figuur 5** Deel van .NET programma geschreven in IL

IL bevat een grote diversiteit aan instructies voor bijvoorbeeld rekenkundige en logische operaties, programmabesturing, directe geheugen toegang en foutafhandeling. Merk op dat IL bovenal ook object-georiënteerd is met instructies voor het laden, opslaan, initialiseren en aanroepen van methoden op objecten.

### Common Type System

Het CTS is een aantal standaard afspraken over hoe wordt omgegaan met typering in IL. De basis voor het CTS is een combinatie tussen een objectgeoriënteerd en procedureel model (zie figuur 6) waarin onderscheid wordt gemaakt tussen objecten en values (waarden). Values zijn simpelweg bitpatronen voor types als integers en booleans. Elke value heeft een *value type* dat de representatie en het gedrag van de waarde beschrijft. Objecten dragen daarentegen hun *object type* in hun representatie mee, hebben een unieke identiteit en kunnen erven van andere object types. Een object type is een speciaal soort *reference type*, waarvan ook *interface types* en *pointer types* specialisaties zijn.




**Figuur 6** .NET Type hierarchie

Value types kunnen indien nodig eenvoudig worden vertaald naar een reference type middels een proces dat *boxing* heet. Naast de ingebouwde value types (zie tabel 1) is het ook mogelijk eigen value types te definieren.

| CTS type | In CLS? | Beschrijving |
|---|---|---|
| bool | ✓ | True/false waarde |
| char | ✓ | Unicode 16-bit karakter. |
| class System.Object | ✓ | Object of boxed value type |
| class System.String | ✓ | Unicode string |
| float32 | ✓ | IEEE 32-bit float |
| float64 | ✓ | IEEE 64-bit float |
| int8 | | Signed 8-bit integer |
| int16 | ✓ | Signed 16-bit integer |
| int32 | ✓ | Signed 32-bit integer |
| int64 | ✓ | Signed 64-bit integer |
| native int | ✓ | Signed integer, native size |
| native unsigned int | | Unsigned integer, native size |
| typedref | | Pointer plus runtime type |
| unsigned int8 | ✓ | Unsigned 8-bit integer |
| unsigned int16 | | Unsigned 16-bit integer |
| unsigned int32 | | Unsigned 32-bit integer |
| unsigned int64 | | Unsigned 64-bit integer |

**Tabel 1** Ingebouwde types

### Common Language Specification

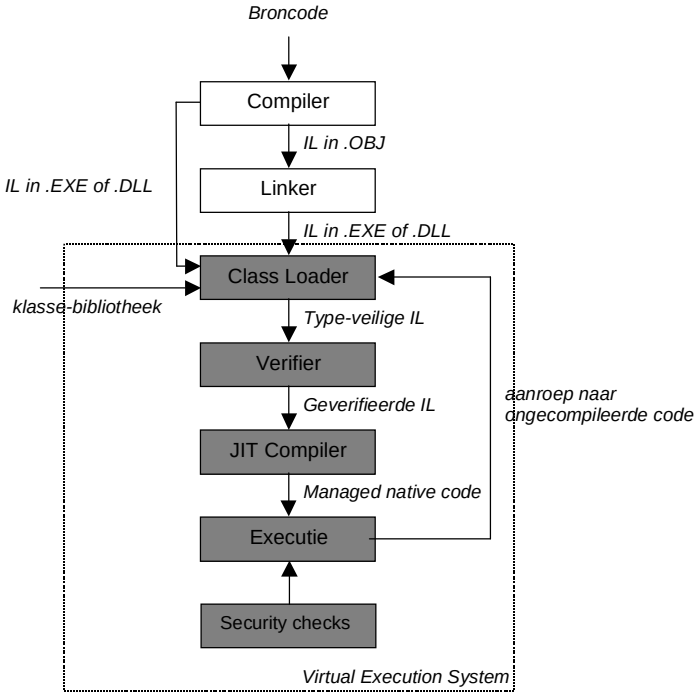
De CLS is een subset van de CTS, gecombineerd met een verzameling afspraken over naamgeving en structuur van elementen die buiten hun assembly zichtbaar zijn. Zo moeten er bijvoorbeeld voor events en properties operaties zijn gedefinieerd die voldoen aan standaard naamgevingsconventies. Het doel van de CLS is de uitwisseling tussen programmeertalen te vereenvoudigen door het gebruik van standaard conventies. Elementen zoals assemblies, types, methoden en attributen kunnen aangeven of ze wel of niet aan de CLS voldoen als dat afwijkt van het element waar ze in gedefinieerd zijn. In tabel 1 is aangegeven welke ingebouwde types voldoen aan de CLS. Naast beperkingen aan deze ingebouwde types zijn er ook andere beperkingen aan types, waardoor bijvoorbeeld unmanaged pointers en boxed values niet zijn toegestaan in de CLS.

### Virtual Execution System

Het Virtual Execution System (VES) vormt de virtuele machine die het CTS implementeert en verantwoordelijk is voor het executeren van IL of OptIL. Alle IL die onder toeziend oog van de VES executeert wordt *managed code* genoemd. Zoals zichtbaar is in figuur 7 verzorgt het VES naast JIT-compilatie ook mechanismen voor het laden en verifieren van IL en het afdwingen van beveiligingsmaatregelen. De ingebouwde *class loader* zorgt voor het inlezen van IL of OptIL in het geheugen, en het uitvoeren van

een aantal basiscontroles waaronder een elementaire authorisatiecontrole. Een strengere controle vindt plaats in de *verifier* waarin wordt gecontroleerd of programma's alleen gebruik maken van geheugen dat voor hun bedoeld is en of objecten alleen via hun publieke interfaces benaderd worden. Alleen betrouwbare code kan deze controleslag overslaan. Na verificatie worden de beveiligingscontroles die declaratief of programmatisch zijn gedefinieerd afgedwongen. Tenslotte biedt het VES diensten voor het *debuggen* en *profilen* van code en het aanroepen van code die niet in de CLR draait, zoals bijvoorbeeld bestaande Windows DLL's




**Figuur 7** Overzicht van de architectuur van het Virtual Execution System

## Metadata

Metadata is erg belangrijk voor de CLR aangezien het de basis vormt voor het kunnen laden, controleren en uitvoeren van code. Er zijn twee belangrijke soorten metadata te onderkennen: assembly-metadata en type-metadata. Een manifest is de representatie van de metadata van een assembly. Naast informatie over bijvoorbeeld de door een assembly geëxporteerde types, beschrijft het ook de afhankelijkheden van de assembly naar specifieke versies van andere assemblies. Deze informatie wordt gebruikt om te bepalen welke versies van andere assemblies er moeten worden ingelezen bij het uitvoeren van de assembly. Afwijkende versioneringsregels kunnen expliciet in het manifest of in een los configuratiebestand worden gedefinieerd. Dit alles is van belang omdat er in .NET is

gekozen om meerdere versies van een assembly naast elkaar te laten bestaan om de bekende "DLL hell"[1] op te lossen. In tabel 2 is weergegeven wat voor soort informatie er in een manifest kan zitten; de schuin gedrukte rubrieken zijn puur ter informatie en worden niet door de CLR gebruikt. Door de ingebouwde metadata zijn assemblies geheel zelf-beschrijvend en is het bijvoorbeeld niet meer nodig assemblies expliciet te registreren.

| | |
|---|---|
| **Naam** | Naam van de assembly |
| **Versie-informatie** | Versie-informatie bestaande uit major, minor, revisie en build nummer. |
| **Gedeelde naam informatie** | Publieke sleutel van de publiceerder van de assembly en een hash-waarde van de gesigneerde versie van het manifest-bestand. |
| **Omgevingsinformatie** | Informatie over de culturen, processoren en besturingssystemen die door het assembly worden ondersteund. |
| **Bestandslijst** | Lijst van bestanden in de assembly met hun relatieve locatie en hashwaarde. |
| **Type-referenties** | Typereferentie informatie om te bepalen in welk bestand een type is gedefinieerd. |
| **Gerefereerde assemblies** | Informatie over gerefereerde assemblies zoals de naam, omgevingsinformatie en publieke sleutel. |
| ***Titel*** | *Een gebruikersvriendelijke naam voor de assembly.* |
| ***Beschrijving*** | *Een korte beschrijving van de assembly.* |
| ***Standaard alias*** | *Een gebruikersvriendelijke standaard alias in het geval de naam niet gebruikersvriendelijk is.* |
| ***Configuratie-informatie*** | *Kan allerlei informatie bevatten* |
| ***Product-informatie*** | *Informatie zoals trademark, copyright, product, bedrijf en informationeel versienummer.* |

**Tabel 2** Inhoud manifest

Het tweede type metadata beschrijft alle informatie over individuele types en maakt deel uit van de portable executables waarin de types zijn gedefinieerd. Ook is het mogelijk zelf metadata te definiëren door eigen attributen toe te voegen. Merk op dat toegang tot de metadata altijd dient te verlopen via de daartoe gedefinieerde programma-interfaces.

## Conclusies

Het .NET framework is een belangrijk technisch component in het .NET platform waarmee .NET applicaties kunnen worden ontwikkeld en uitgevoerd. Naast een grote hoeveelheid standaard klassen voor het maken van allerlei soorten applicaties biedt het

[1] Deze DLL hell treedt op wanneer nieuwe applicaties bestaande DLL's overschrijven met nieuwe versies, waardoor oude applicaties potentieel niet meer goed functioneren.

.NET framework een rijke runtime-omgeving. In deze omgeving staan het gebruik van de Intermediate Language en de bijbehorende metadata centraal. Zij maken het mogelijk applicaties te laden, te controleren en uit te voeren. Samengevat lijkt het .NET platform veelbelovend en zal het een geduchte concurrent worden voor het Java platform.

*Danny Greefhorst is werkzaam als senior adviseur bij het Software Engineering Research Centre te Utrecht.*