Gert-Jan Messchendorp is senior consultant bij Capgemini

Overstappen van Visual Basic 6.0 naar Visual Basic .NET

WAT DOEN WE MET DE OUDE APPLICATIES?

Drie jaar na de introductie van .NET is Visual Studio .NET uitgegroeid tot de standaard ontwikkeltool voor de Microsoft-omgeving. Steeds meer mensen wagen de overstap naar .NET. Een overstap die gezien de ontwikkelingen op het Windows-platform, met de releases van Whidbey en Longhorn in zicht, onvermijdelijk is.

Naast de vraag op welke taal iemand zou kunnen overstappen, rijst in de automatiseringsmarkt steeds vaker de vraag wat te doen met de 'oude' Visual Basic 6.0-applicaties. Dit artikel gaat in op de keuzemogelijkheden die er zijn hoe met de oude Visual Basic 6.0-code om te gaan. Daarnaast komt het geautomatiseerd migreren van de bestaande Visual Basic 6.0-code aan de orde en het verhogen van het rendement van de geautomatiseerde migratie. Ook zien we hoe een migratietraject kan worden aangepakt. Globaal gezien zijn er drie opties om met de oude programmatuur om te gaan. De eerste optie is helemaal niets doen en de oude code in Visual Basic 6.0 blijven onderhouden. De tweede, voor ontwikkelaars meest attractieve, mogelijkheid bestaat uit het helemaal opnieuw beginnen. De laatste mogelijkheid is het migreren van de bestaande Visual Basic 6.0-code naar Visual Basic .NET.

Optie 1: Helemaal niets doen

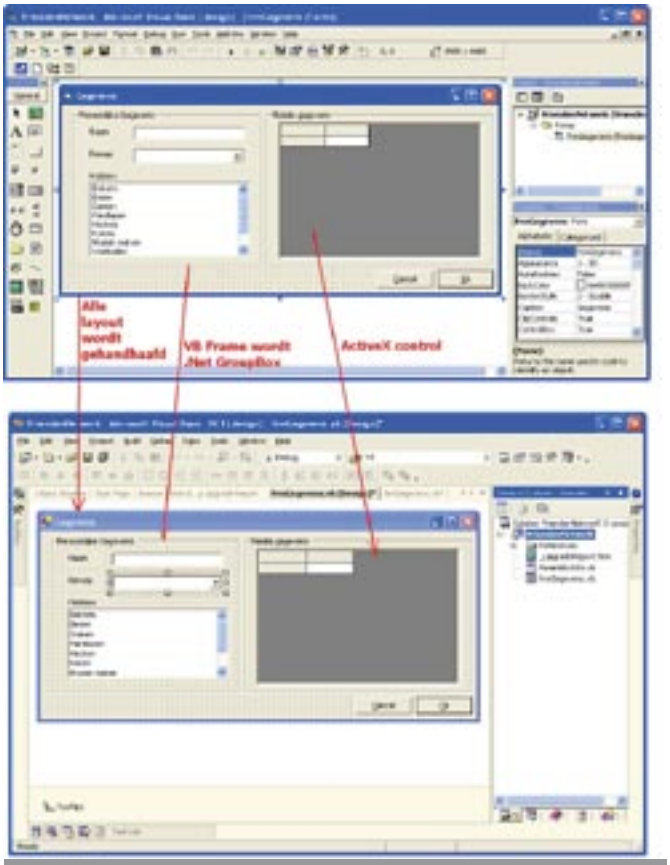
Hoewel de optie op stilstand lijkt, is het soms het beste om helemaal niets te doen. Vooral voor applicaties waarin heel veel testcapaciteit is gestoken, waarvan weinig of geen onderhoud wordt verwacht of die naar verwachting snel zullen worden vervangen, loont het vaak niet te migreren. Een codemigratie moet zich terug kunnen verdienen. Er moeten voor de applicatie voldoende voordelen van .NET worden onderkend om de investering in de migratie te kunnen rechtvaardigen (return of investment). Eigenlijk zou je code slechts in twee gevallen willen migreren. Ten eerste als je van plan bent de applicatie uit te gaan breiden met functionaliteit die alleen .NET biedt, ten tweede als je van plan bent de code nog heel lang te onderhouden. Houd ook rekening met de lifecycle policy van Microsoft voor Visual Basic 6.0 en eventueel componenten waar de applicatie gebruik van maakt. Zie hiervoor www.microsoft.com/lifecycle.

Optie 2: Opnieuw beginnen

Visual Basic .NET is een krachtige taal die veel meer handgrepen biedt om goede applicaties te schrijven dan Visual Basic 6.0. Voorbeelden hiervan zijn overerving, remoting, security en het objectmodel dat het .NET Framework biedt. Als er toch al aanleiding is om een applicatie eens grondig te herstructureren, dan is opnieuw beginnen een goede optie. Een herontwerp is een goed moment om op .NET over te gaan. De originele applicatie kan daarbij als een goed werkend prototype worden gezien. Bovendien is er een aantal instructies uit Visual Basic 6.0, zoals DAO en RDO databindings en OLE-containers, waar geen geldig .NET-equivalent voor bestaat (zie ook het onderdeel Compatibiliteit). Als je applicatie hiermee doorspekt is, dan is geautomatiseerd migreren geen optie. De enige geldige keuzes zijn dan herschrijven of niets doen. Een nadeel van herschrijven is dat een volledig ontwikkeltraject in moet worden gegaan en dat er veel tijd en geld in testen moet worden geïnvesteerd. Het blijft moeilijk keuze, ook als we alle voor- en nadelen op een rij zetten. Sommige ontwikkelaars hebben een uitgesproken voorkeur om opnieuw te beginnen. De stelling is dat het betere code oplevert.

Optie 3: (Geautomatiseerd) migreren

Een derde optie is het migreren van de bestaande programmacode naar .NET. Op deze manier wordt zo veel mogelijk van de reeds geproduceerde code hergebruikt en is het minder inspannend om over te gaan naar .NET. Er bestaan verschillende tools waarmee zo'n migratie kan worden uitgevoerd en een aantal daarvan doet dat best goed.



Afbeelding 1. Bij automatische migratie wordt de lay-out zeer nauwkeurig overgenomen

Een nadeel van een rechtstreekse migratie is dat niet optimaal van alle mogelijkheden (zoals overerving) van .NET gebruik wordt gemaakt en dat de opbouw en structuur van de applicatie dezelfde blijven. Bovendien moet alle programmatuur opnieuw worden getest alsof het nieuwbouw betreft, omdat code niet altijd hetzelfde reageert als voorheen. Hier kom ik bij compatibiliteit op terug met een voorbeeld. Ondanks de genoemde nadelen heeft migreren de potentie om voor een behoorlijke tijdwinst te zorgen, mits de applicatie er zich voor leent en de migratie goed overwogen wordt uitgevoerd.

Hybride oplossingen

Vaak zal het zo zijn dat niet de tijd en noodzaak bestaat om in één keer alle programmatuur naar .NET om te zetten. Zeker als de gehele applicatie al in meer lagen of modules is opgebouwd, kan de COM-interoperabiliteit een uitkomst bieden om in verschillende fasen op .NET over te stappen. COM-interop is een mechanisme waarmee COM-componenten vanuit .NET kunnen worden aangeroepen en andersom. Het werkt met RCW's en CCW's (Runtime/COM Callable Wrappers). RCW's zijn .NET-assemblies die een COM-component wrappen en functionaliteit bevatten om alle calls naar COM-calls te vertalen. RCW's zijn eenvoudig te creëren door in een COM-component naar een .NET-project te refereren of door tlbimp.exe te gebruiken. CCW's worden gebruikt om .NET-assemblies als COM-component in de registry te plaatsen. Onder de naam van de .NET-component wordt een COM-component (mscoree.dll) geregistreerd die in staat is de COM-calls naar de juiste .NET-component te routen.
Gebruikmakend van COM-interop kunnen die dll's, die je naar .NET om wil zetten, worden vervangen terwijl de rest in COM blijft werken. Houd er rekening mee dat COM-interop overhead met zich meebrengt, zie ook http://msnd.microsoft.com/library/en-us/cpguide/html/cpconInteroperatingWithUnManagedCode.asp. Ook in het geval dat gekozen wordt een applicatie volledig te herschrijven, is met een deelconversie nog winst te behalen. Denk daarbij aan het geautomatiseerd migreren van de Forms zodat je een heleboel design-, schuif-, resize- en uitlijnwerk uitspaart. Zie daarvoor ook het voorbeeld in afbeelding 1.

Beschikbare tools voor geautomatiseerde migraties

Naast Microsoft zijn er verschillende andere aanbieders van tools om Visual Basic 6.0 naar .NET te vertalen. Toen ik eind vorig jaar een onderzoek deed naar het geautomatiseerd converteren van Visual Basic 6.0-code, waren er voornamelijk tools te vinden die Visual Basic 6.0 naar C# converteren en was de upgrade wizard van Artinsoft (wordt standaard met Visual Studio .NET meegeleverd) de enige die Visual Basic .NET als doeltaal had. Veel tools die ik toen heb bekeken, waren nog niet erg bruikbaar. Maar de tijd heeft niet stilgestaan en gedurende het afgelopen jaar zijn er heel wat nieuwe versies op de markt gekomen die zeker het proberen waard zijn. De kosten van de tools variëren nogal, maar op veel sites zijn trialversies te downloaden. In tabel 1 heb ik twee van die tools opgenomen. Afgaande op de documentatie hebben de meeste tools gelijkwaardige beperkingen als de upgrade wizard die standaard met Visual Studio .NET wordt meegeleverd. Voor het migratieproces maakt het niet uit welke tooling precies wordt gebruikt. Ik beperk me daarom tot de wizard.

Aanbieder/Tool	Omschrijving
Delux Software / Delux#	Vertaalt Visual Basic 6.0 naar C#. De demoversie die er begin dit jaar stond voldeed nog niet (AND werd bijvoorbeeld vertaald naar ++, terwijl het && moet zijn), maar er is inmiddels een nieuwe versie die veelbelovend is. Prijsindicatie 250 tot 600 euro. http://www.deluxsoftware.com verkrijgbaar via: http://www.qbssoftware.com
NetCoole / VB6 to C#	Vertaalt ook Visual Basic 6.0 naar C#. Voor simpele applicaties doet de demoversie dat goed. Op middelgrote applicaties loopt de demoversie hopeloos vast, zonder een error-melding of logbestand te geven. In juli is een nieuwe versie uitgekomen. Prijsindicatie 350 tot 550 euro. http://www.net-coole.com/vb6tocs.htm
Aivosto/Project Analyzer 7.0	Analyseert een project, detecteert dode code et cetera. De applicatie geeft de mogelijkheid een Visual Basic .NET compatibility-report te genereren waarin staat hoeveel fixes moeten worden gedaan vóór de upgrade, hoeveel vooraf en naderhand gefixt kunnen worden en hoeveel fixes achteraf nog nodig zijn. Bovendien is aan de hand van een problem-report vooraf te zien welke zaken gefixt kunnen worden. Deze tool is niet alleen handig bij het prepareren van code voor conversies, maar bevat ook heel veel tips die bij bestaande Visual Basic 6.0-applicaties een gunstig effect op de performance en de mate van onderhoud kunnen hebben. De kosten voor deze tool zijn afhankelijk van het licentietype en pakketuitvoering. De prijzen variëren van $ 199 voor een single user-licentie tot $ 3950 voor een site-licentie. www.aivosto.com/vbcatalog.html.
Microsoft Code Advisor	Add-in voor Visual Basic 6.0. Doormiddel commentaarregels in de code wordt een suggesties gedaan om verbeteringen aan te brengen. De tool kan zelf deze commentaarregels ook weer verwijderen. De tips die de tool bevat zijn met name gericht op het converteren van Visual Basic 6.0 naar .NET. Een deel van de tips komt overeen met de tips die Aivosto geeft, maar de code advisor heeft er meer. Met een filter is aan te geven voor welke target-omgeving (Visual Studio .NET 2002 of Visual Studio .NET 2003) de code moet worden voorbereid. Ook de code advisor geeft de mogelijkheid een issue-rapport te laten maken. De code advisor doet er overigens significant langer over om tot een resultaat te komen dan de Aivosto-tool. De code advisor is gratis te downloaden bij Microsoft. http://msdn.Microsoft.com/vbasic/downloads/codeadvisor/default.aspx.
FXCop	Een tool die op GotDotNet (www.gotdotnet.com/) te downloaden is. Deze tool checkt .NET-code op de door Microsoft voor .NET gedefinieerde naming conventions. Voor veel geconverteerde Visual Basic-code zal deze tool rood uitslaan omdat de naming convention in .NET heel anders is dan in Visual Basic 6.0. Deze tool waarschuwt alleen en past de code niet aan. http://www.gotdotnet.com/
New Technology Inc. - Attila/VB	Deze tool dwingt codestandaards af. De tool is voor Visual Basic 6.0 geschreven maar er is ook een .NET-versie aangekondigd. http://www.newtechusa.com/Resources/Attila.asp

Tabel 1. Nuttige tools bij migratie

De upgrade wizard

De upgrade wizard is een standaard onderdeel van Visual Studio .NET en is gemaakt om de drempel van de overstap van Visual Basic 6.0 naar Visual Basic .NET te verlagen. De upgrade wizard nodigt uit om uitgeprobeerd te worden. De wizard start heel eenvoudig door een Visual Basic 6.0-project te openen. Daarna bestaat de vervolgstap uit een aantal malen op next te drukken, waarna het Visual Basic-project wordt geconverteerd. De wizard plaatst het resultaat in een nieuwe directory en opent het geconverteerde project. In de tasklist van het project staat wat verder nog moet worden gedaan. Bovendien wordt in het project een logbestand gezet (_UpgradeReport.htm) waarin alle conversie-issues staan.
Ik denk dat een groot aantal mensen de upgrade wizard al een keer heeft uitgeprobeerd. De euforie dat het zo eenvoudig is een .NET-project te creëren, wordt vaak gevolgd door een desillusie wanneer de grote lijst met punten in de tasklist volgt. Het is ook

```
Private Sub Command1_Click(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Command1.Click
  Dim colItems As New Collection
  Dim address As Integer

  'UPGRADE_ISSUE: ObjPtr function is not supported. Click for more:
  'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1040"'
  address = ObjPtr(colItems)

  MsgBox("Het adres is: " & address)
End Sub
```

Codevoorbeeld 1.

Commentaar met de upgrade-issues en een link naar de MSDN-pagina waar het probleem nader wordt beschreven.

```
Public Sub ToggleTimer(bOn As Boolean)
  If bOn Then
    Timer1.Interval = m_iTimerInterval 'Turn On
  Else
    Timer1.Interval = 0                'Turn Off
  End If
End Sub
```

Codevoorbeeld 2a.

Timercode in Visual Basic 6.0

```
Public Sub ToggleTimer(ByRef bOn As Boolean)
  If bOn Then
    'UPGRADE_WARNING: Timer property Timer1.Interval cannot have
    ' a value of 0. Click for more:
    'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup2020"'
    Timer1.Interval = m_iTimerInterval 'Turn On
  Else
    'UPGRADE_WARNING: Timer property Timer1.Interval cannot have
    ' a value of 0. Click for more:
    'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup2020"'
    Timer1.Interval = 0 'Turn Off
  End If
End Sub
```

Codevoorbeeld 2b.

Gemigreerde Timercode in Visual Basic .NET. De code lijkt hetzelfde als de code in voorbeeld 2a. De code compileert, maar de code werkt niet meer.

even schrikken als uit een aantal testen blijkt dat gemiddeld 1 taak per 7 regels geconverteerde code wordt aangemaakt. Toch verdient het aanbeveling eens goed naar alle taken te kijken aangezien een groot deel ervan eenvoudig is op te lossen. In principe zijn er vier niveaus waarop de migratie taken aanmaakt.

1. *Issue*. Instructies die niet rechtstreeks kunnen worden vertaald.
2. *ToDo*. Er is iets vertaald, maar het moet nog wel handmatig worden rechtgezet.
3. *Warning*. Instructies waarvan de werking niet altijd identiek is aan de Visual Basic 6.0-situatie.
4. *Note*. Notes worden vaak in de code gezet als de wizard de code zodanig heeft aangepast dat het origineel er niet meer in is te herkennen.

Over het algemeen krijgt een groot deel van de correct vertaalde code dus toch een note of een warning. Bij de warning staat altijd een link naar MSDN genoemd, waarin heel duidelijk wordt omschreven waarom en hoe de ontwikkelaar de code moet wijzigen. (Ctrl+muisklik op de link brengt je er heen). Zie ook codevoorbeeld 1.

Na het uitvoeren van de upgrade wizard ben je er nog niet. Er moet nog heel wat gebeuren om de gemigreerde code goed te laten functioneren. Een goede aanpak is de output van de wizard te gebruiken om de originele code mee aan te passen. Heel veel zaken kunnen vooraf worden verbeterd. Na het verbeteren van de originele code kan de migratie opnieuw worden uitgevoerd en blijven alleen fouten over die achteraf rechtgezet moeten worden.

Compatibiliteit

Op MSDN heeft Microsoft een grote compatibiliteitslijst gepubliceerd met instructies waarvoor geen rechtstreeks alternatief in .NET bestaat. Dit is code die dus niet te migreren is. Denk hierbij aan harde referenties (ObjPtr, StrPtr, VarPtr et cetera.), DAO/RDO databindings, OLE-containers, On x Goto/Gosub, property-pages enzovoort. In de compatibiliteitslijst staat ook hoe 'obsolete' code kan worden vervangen. Het is zaak vooraf goed te beoordelen hoeveel van dit soort code in de Visual Basic 6.0-applicatie aanwezig is. Als het echt veel is (denk bijvoorbeeld aan RDO databinding door de hele applicatie heen), dan kan het voordeliger zijn om de betreffende delen helemaal opnieuw te schrijven of om voor die delen af te zien van migratie.

Een aantal problemen is met voor- of nawerk goed te voorkomen. Kijk daarbij altijd goed wat de beoogde instructie is en vervang die voor of na de conversie door een goed equivalent. De ObjPtr-functie kan bijvoorbeeld gebruikt zijn om circulaire COM-referenties te doorbreken (object A houdt object B vast en andersom). Tussen .NET-objecten bestaat het probleem van circulaire object-referenties niet meer wegens de werking van de Garbage Collector. Zie ook http://microsoft.com/msdnmag/issues1100/gci

Een ander voorbeeld om vooraf de code te bewerken is DAO te vervangen door ADO. Met het converteren van ADO heeft de wizard geen problemen. Verder heeft .NET objecten die gelijk lijken te zijn aan de VB-objecten, maar toch ander gedrag vertonen. Een voorbeeld hiervan is de Timer. Voorheen (VB6) werd een timer aan of uit gezet door respectievelijk een intervaltijd of 0 aan de Interval-property toe te kennen. Deze code migreert foutloos, hoewel met twee warnings; zie codevoorbeelden 2a en 2b. De gegenereerde code compileert meteen, maar hij blijkt zich niet meer te gedragen als voorheen. De timer wordt in .NET namelijk aan en uit gezet met de Enabled-property en bovendien is het verboden om de Interval op 0 te zetten. Het feit dat er heel af en toe een fout kan voorkomen, die pas bij het uitvoeren van de code kan worden ontdekt, bepaalt dat alle code opnieuw zorgvuldig moet worden getest.

Geautomatiseerde voorbewerking en nabewerking

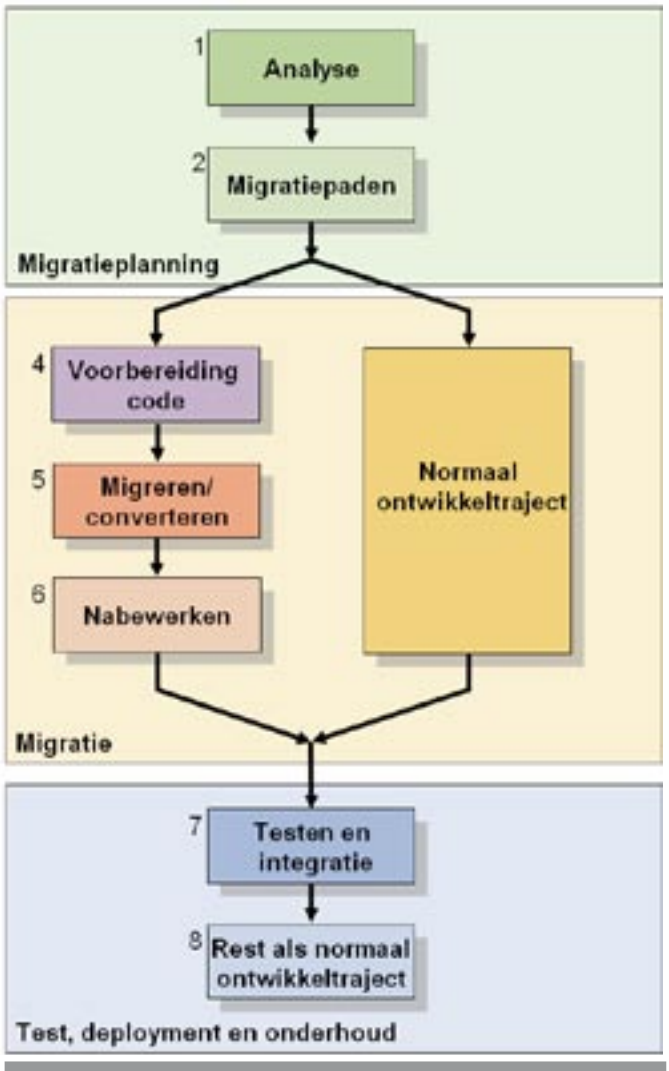
Net als voor elk ander conversietraject geldt ook voor de codemigratie: Garbage In - Garbage Out. Het is dus zaak om de code vóór de conversie zo schoon mogelijk te krijgen. Dit kan door de code (geautomatiseerd) voor en na te bewerken. In een test blijkt 85% van de code in één keer goed te converteren. Het resultaat van de conversie blijkt zelfs 90% te zijn, als de code met een tool is voorbereid. Van de 10% die daarna nog moet worden geconverteerd, zou de helft een standaard invulling kunnen krijgen, wat betekent dat uiteindelijk 5% code overblijft die echt handmatig moet worden omgezet. De applicatie waar het hier om gaat is een standaard 2-tier client/server-applicatie die via ADO een SQL Server database benadert. Ik denk dat deze applicatie representatief is voor het gros van andere, middelgrote applicaties die in Visual Basic 6.0 zijn geschreven.Verder zeggen bovenstaande getallen natuurlijk niet veel over andere projecten. Wel geven ze aan hoeveel rendementsverbetering je krijgt bij een migratie door de code een goede voor- en nabehandeling te geven.

Federico Zoufaly van Artinsoft, het bedrijf dat de upgrade wizard heeft gemaakt, stelt zelfs dat een ontwikkelaar bij de migratie 33% van de tijd aan het voorbereiden van de code moet besteden. Er is een aantal tools op de markt waarmee je geautomatiseerd kunt worden geholpen, zoals de project-analyzer van Aivosto. Deze tool helpt je om de code gereed te maken voor .NET, het herkent dode code en geeft zelfs aan of er

performancetechnisch nog verbeteringen in de Visual Basic 6.0-code uit te voeren zijn. Ook Microsoft biedt geschikte tooling aan zoals de Code Advisor. Dit is een add-in voor Visual Basic 6.0 die voor een target-omgeving (Visual Studio 2002 of Visual Studio 2003) aangeeft waar problemen te verwachten zijn. Bovendien geeft de migratietool (of de link op MSDN) zelf ook aan hoe je het betreffende probleem vooraf kunt verhelpen. Het migreren is een iteratief proces: migreren, fouten controleren, wijzigingen in de broncode doorvoeren en opnieuw migreren. Voor het schonen van code en het forceren van codingstandaarden is geautomatiseerde hulp te krijgen met tools als Atilla/VB en FxCop. Schrik niet, een tool als FxCop genereert eerst een flink aantal meldingen, maar je kunt zelf goed instellen welke validatieregels de tool wel en niet moet gebruiken.

Verder beschikt Visual Studio .NET over een heel krachtige macro-omgeving waarin je met Visual Basic .NET als macrotaal een heleboel codegerelateerde zaken kunt uitvoeren. Als je van plan bent meer applicaties te migreren die in eenzelfde stramien zijn geschreven, is het verstandig na te denken over standaardoplossingen voor veelvoorkomende migratiefouten. Voorbeelden van te automatiseren standaardoplossingen zijn:

- Een Mousepointer wordt in VB6 vaak in een integer variabele (short in .NET) gecached. In .NET is daar een class *System.Windows.Forms.Cursor* voor gemaakt. Het cachen van de MousePointer is herkenbaar en kan geautomatiseerd worden vervangen.
- App.Revision uit VB6 bestaat in .NET niet meer. Er voor in de plaats gekomen .zijn heel veel nieuwe mogelijkheden om versienummers op te halen. Je kunt er één kiezen en de call geautomatiseerd vervangen.



Afbeelding 2. Het migratieproces

Type project	In Visual Basic 6.0 laten	Naar .NET
Code die snel wordt uitgefaseerd	X	
Code werkt perfect en hoeft functioneel niet te wijzigen	X	
Code wordt nog heel lang onderhouden	X	X
Performance van ASP moet worden verbeterd	X	X
Gebruik van extra mogelijkheden, bijvoorbeeld remoting, XML of webservices		X
Code wordt sterk uitgebreid		X
Nieuwe code		X
Architectuur wordt herzien		X
ActiveX-document, DHTML	X	
Veel grafische routines en optimalisaties (bijv. Games)	X	

Tabel 2. Visual Basic 6.0 of .NET? De keuze is niet zwart-wit. In de tabel staat alleen in welke richting de keuze wordt 'geduwd' door elk punt.

Type project	Herschrijven	Migreren
Dll zonder userinterface met weinig 'verboden' instructies		X
Userinterfaces		X
Veel RDO/DAO databinding	X	
ASP		X
StrPtr, ObjPtr et cetera en veel API-calls	X	X
Architectuur wordt herzien	X	
Veel grafische routines en optimalisaties (bijv. Games)	X	

Tabel 3. Als de keuze op de overgang naar .NET valt: herschrijven of migreren?

- Voor de timer is een macro te maken waarbij het toekennen van 0 aan Interval wordt vervangen door Timer.Enabled = false én na elke toekenning van een grotere waarde Timer.Enabled = true wordt toegevoegd.

Deze macro's kunnen veel handelingen en tijd besparen.

Migratiepaden

Migratiepaden geven in een migratieplan aan welke volgorde wordt aangehouden bij het migreren of herschrijven van de code. Bij het migratiepad wordt ervan uitgegaan dat de applicatie al modulair is opgezet. Dus het moet al uit verscheidene componenten (dll's) bestaan en het liefst meerlaags zijn opgezet. In principe zijn er twee basis migratiepaden: horizontale en verticale migratie. Horizontale migratie houdt in dat in een meerlaags model een hele laag wordt omgezet naar .NET. Verticale migratie houdt in dat door alle lagen heen voor een bepaalde use-case alle componenten worden gemigreerd. In de praktijk zijn er natuurlijk meer combinaties van beide smaken mogelijk.

De leidraad voor een migratiepad is communicatie en codetoegang. Naarmate componenten intensiever met elkaar 'praten', is de kans groter dat ze gezamenlijk worden gemigreerd. Als de ene component migreert en de andere niet, en ze moeten elkaar aanspreken, dan betekent dit dat de interoperabiliteit moet worden geregeld. Dat houdt ook in dat een extra stap in de communicatie moet worden gedaan. In het geval van vele kleine wederzijdse aanroepen (chatty calls) weegt die communicatie zwaarder dan bij grotere aanroepen (chunky calls). Voor het bepalen van de migratiepaden is het derhalve belangrijk te weten wat de code-accesspaden van de applicatie zijn. Als je meer over migratiepaden in combinatie met

COM-interop wilt weten, raad ik je aan om http://msdn.microsoft.com/library/en-us/dnbda/html/cominterop.asp te lezen.

Een goed voorbeeld van een horizontale migratie is het vervangen van een hele ASP-laag door een ASP.NET-laag, terwijl de COM-componenten die door de laag worden aangesproken onveranderd blijven. Het omzetten van ASP naar ASP.NET kan aan de serverkant overigens veel performancewinst opleveren, omdat de geïnterpreteerde ASP-code aan de serverkant wordt vervangen door gecompileerde ASP.NET-code. Microsoft geeft zelf aan dat ASP.NET-sites tweemaal tot driemaal zoveel aanvragen kunnen behandelen dan originele ASP-pagina's.
Een voorbeeld van een verticale migratie is het vervangen van een hele invoerdialoog door een .NET-invoerdialoog die zijn gegevens ophaalt, bewerkt en opslaat met een dataset die is verkregen van een onderliggende .NET-component; dit terwijl de rest van de applicatie op de oude manier blijft doorwerken.

In een combinatieoplossing kan je besluiten om juist van een webapplicatie een paar delen in ASP.NET uit te gaan voeren - daar waar de voordelen van .NET groter zijn dan de inspanningen van het migreren - en tegelijk stap voor stap onderliggende COM-componenten door .NET-componenten te vervangen. Zo houd je de migratie overzichtelijk met kleine behapbare blokken die achter elkaar kunnen worden uitgevoerd.

Het migratieplan

De basis voor een goede uitvoering van een codemigratie is het opstellen van een migratieplan. Een migratieproject is een volwaardig ontwikkeltraject dat begint bij een plan van aanpak. Het migratieplan bevat de volgende onderdelen:
- indicatie van de migratiedelen
- beschrijving van het migratiepad dat wordt doorlopen en een definitie van de tussenopleveringen aan de testomgeving
- indicatie van de tooling en hoe die wordt gebruikt en onderhouden
- een testplan
- een deployment-plan.

Het migratieproces

In het migratieproces zijn verschillende stadia te onderkennen.

1. Analysefase
Hier wordt bepaald wat gemigreerd wordt, wat oud blijft en welke delen volledig worden herschreven. Belangrijke gegevens voor dit besluitproces zijn de opbouw van de applicatie, de noodzaak componenten om te zetten, de risico's per component, de complexiteit van de code (in verband met testen) en de compatibiliteit met .NET van de broncode voor elk component. (zie ook tabel 2 en 3).

2. Het bepalen van de migratiepaden
Geeft aan in welke volgorde de componenten naar .NET worden omgezet, met welke componenten moet worden begonnen en welke (werkende) tussenstadia worden gedefinieerd. Werkende tussenstadia zijn belangrijk om de migratie overzichtelijk te houden, migratierisico's te beperken en eventuele problemen eenvoudiger te kunnen onderscheiden.

3. Opstellen van een migratieplan
Aan de hand van punten 1 en 2 kan een migratieplan worden opgesteld. Het migratieplan bevat een standaard ontwikkelproces voor de componenten die in hun geheel door nieuwe worden vervangen. Voor omzetting van de applicaties vindt hierna de codemigratie plaats.

4. *Voorbereiding van de migratiecode*
Geautomatiseerd en handmatig schonen en voorbereiden van de programmacode.

5. Migratiestap
Een migratietool migreert de code. Alle fouten in de migratie krijgen een indicatie of deze vooraf of achteraf gerepareerd moeten worden. In geval van reparatie vooraf wordt het verhaal vanaf punt 4 herhaald. Veelvoorkomende reparaties die vaak hetzelfde resultaat moeten hebben, nemen we hier eventueel op in een voorbereidingsapplicatie, zodat ook de migratie per cyclus verbetert.

6. Nabewerken van de code
Alle stukken code die niet goed gemigreerd zijn worden handmatig of geautomatiseerd gemigreerd. Alle veel voorkomende stappen kunnen hier eventueel aan een nabewerkingapplicatie worden toegevoegd.

7. Testen van de code
De code moet hier worden getest alsof hij door een regulier ontwikkelproces is gecreëerd. Deze stap kan arbeidsintensief zijn. Het voordeel bij testen is dat de oude code als referentieomgeving kan dienen.

8. Bugfixen, deployment et cetera
Het betreft hier werkzaamheden van het reguliere softwareontwikkelproces.

De juiste keuzes

In de overstap van Visual Basic 6.0 naar .NET is geautomatiseerde migratie een optie, mits de code aan een aantal voorwaarden voldoet; zie tabel 3. Het is niet verplicht alle code om te zetten naar .NET. Soms is het verstandiger oude code te behouden en deze vanuit .NET aan te spreken.
Houd er rekening mee dat het uitvoeren van een migratie geen triviale bezigheid is. Om de risico's van een migratie zoveel mogelijk binnen de perken te houden verdient de migratie een goede analyse en een gedegen migratieplan. Alle gemigreerde broncode moet opnieuw worden getest alsof hij net is ontwikkeld. Denk ook na over goede tooling om de code (semi)geautomatiseerd voor te bereiden en na te bewerken. Elke stap die je kunt automatiseren moet je ook automatiseren. Dit kan heel veel tijd en werk besparen, vooral als meer dan één conversie moet worden uitgevoerd.

Ir. Ing. Gert-Jan Messchendorp is senior consultant bij Capgemini (MCSD.Net), heeft ervaring met C++, Visual Basic 6.0 en SQL-Server en is specialist in C# .NET. Zijn e-mailadres is gert-jan.messchendorp@capgemini.com. Website: www.capgemini.com

Nuttige internetadressen

Algemene informatie over de upgrade van Visual Basic 6.0 naar Visual Basic .NET inclusief stappenplan:
http://msdn.Microsoft.com/library/en-us/dnvb600/html/vb6tovbdotnet.asp
Anbevelingen voor het voorbereiden van Visual Basic-code voor de .NET-conversie:
http://msdn.Microsoft.com/library/en-us/vbcon/html/vbconpreparingvisualbasic60applicationformigration.asp
Overzicht met compatibiliteit tussen Visual Basic 6.0 en Visual Basic .NET met een mogelijke oplossing erbij: http://msdn.Microsoft.com/library/en-us/vbcon/html/vboriVisualBasic60CompatibilityReference.asp
Een selectie met verschillende artikelen over de migratie van projecten geschreven in oude Visual Basic-versies naar Visual Basic .NET: http://msdn.Microsoft.com/library/en-us/vbcon/html/vbconupgradingapplicationscreatedinpreviousversionsofvisualbasic.asp
Visual Basic upgrade wizard: http://www.artinsoft.com
Projectanalysetool voor Visual Basic 6.0: http://www.aivosto.com/
Abandoning the fantasy of Visual Basic migration Wizardry, interview met Frederico Zoufaly (ArtinSoft): http://www.devx.com/dotnet/Article/16822/0/page/1
The Visual Basic .NET Upgrade Wizard, Need To Upgrade Visual Basic 6.0 to Visual Basic .NET? Consider This First! Gary Cornell: http://visualbasic.about.com/library/weekly/aa120102a.htm
Referentiemateriaal voor ASP-ontwikkelaars:
http://www.microsoft.com/netherlands/msdn/aspnet/aspkit/asptoaspnetmigration.aspx