Huib Klink is werkzaam als senior consultant bij Micro Focus.

COBOL in de .NET-omgeving

HERGEBRUIK VAN DATA EN CODE EN KORTE LEERCURVE MAAKT .NET AANTREKKELIJK

Veel mensen denken dat Cobol op sterven na dood is. Maar gebruikt u wel eens een geldautomaat? Grote kans dat Cobol betrokken is bij de verwerking van uw pintransacties. Ontvangt u een pensioen of hoopt u dat dit in de toekomst gaat gebeuren? Waarschijnlijk is er Cobol betrokken bij het monitoren van uw pensioengroei. In dit artikel maken we kennis met Cobol in de .NET-omgeving.

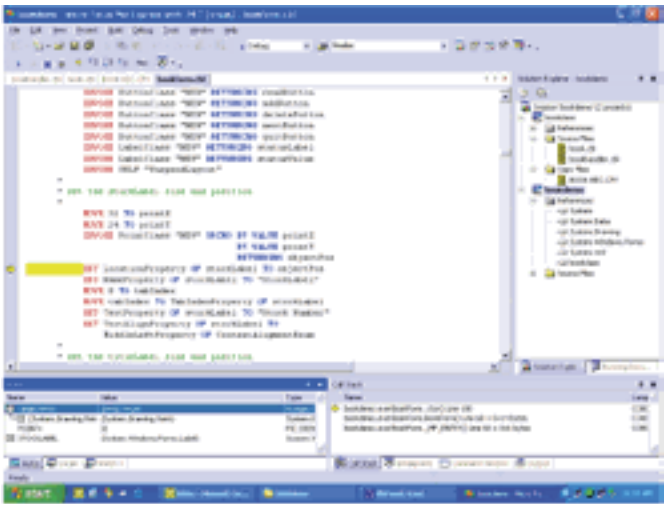
De basis voor Cobol is gelegd in de jaren '60. Inmiddels heeft de taal, die vooral bekend is van het maken van bedrijfscalculaties, zich sterk ontwikkeld. In de jaren '80 werden de benodigdheden voor gestructureerd programmeren aan Cobol toegevoegd en in de jaren '90 werd de taal objectgeoriënteerd. Op dit moment is Cobol een moderne ontwikkeltaal die een belangrijke plaats inneemt tussen technologieën als .NET, XML en webservices. Dit stelt organisaties in staat applicaties beschikbaar te stellen aan een grote groep nieuwe klanten. Volgens de Gartner Group zijn er wereldwijd tussen de 180 en 200 miljard regels Cobol-code in gebruik. De Tactical Strategy Group schat de vervangingskosten voor één regel Cobol-code op $25. Dat brengt de totale vervangingskosten op honderden miljarden dollars. Er is in de afgelopen decennia meer dan $1,5 biljoen geïnvesteerd in Cobol-applicaties. Bovendien vinden dagelijks meer dan 30 miljard Cobol-transacties plaats. Dat is meer dan het gemiddelde aantal bezochte webpagina's in een etmaal (Computerworld, 22 oktober 2003). Een goedkoper alternatief dan herschrijven is hergebruik van Cobol-code in nieuwe omgevingen.

Cobol in Visual Studio .NET

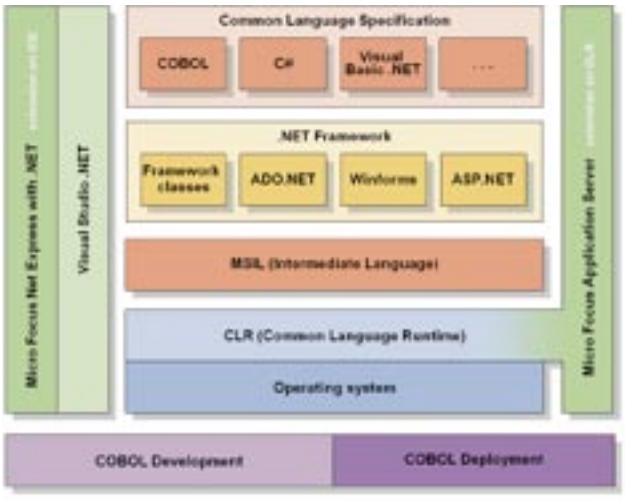
Inmiddels is de lijst met programmeertalen voor .NET imposant lang. Vanaf de introductie van .NET is Cobol van de partij. Beschikbaar is o.a. Micro Focus Net Express with .NET, software die volledig geïntegreerd is in Visual Studio .NET. De ontwikkelaar gebruikt dezelfde ontwikkelomgeving voor zowel Cobol als alle andere .NET-talen. Toegevoegd aan Visual Studio zijn voorgedefinieerde Cobol-projecttypes voor Windows-, Console-, Class Library- (DLL-) en ASP.NET webservice-applicaties. Verder zijn alle bekende zaken aanwezig zoals syntax coloring, ondersteuning van 'intellisense' bij de ontwikkeling van zowel procedurele als objectgeoriënteerde code, tooltip-informatie bij Cobol-variabelen, debugger voor Cobol-code met Watch Windows en Breakpoints.

Managed Cobol

De compiler genereert Microsoft Intermediate Language (MSIL) zoals ook bij andere.NET-talen gebeurt. Het resultaat is volledige managed code. De compiler accepteert traditionele procedurele en objectgeoriënteerde Cobol-statements. De bestaande procedurele Cobol-programma's kunnen uitgebreid worden met toegang tot .NET-classes door 'invoke'-statements toe te voegen. Dit vergemakkelijkt het hergebruik van bestaande code in .NET-omgevingen, zonder dat de procedurele logica omgezet hoeft te worden in objectgeoriënteerde. Hergebruik van mainframe Cobol-programma's is mogelijk doordat de compiler mainframedialecten zoals ISAM en SQL ondersteunt. Ook zonder wijzigingen in de broncode van de soms decennia oude, maar nog goed werkende Cobol-programma's, kan legacy worden hergebruikt. Het bouwen van bijvoorbeeld een OO-wrapper (in Cobol) die een procedurele call doet vanaf naar de oude programmatuur is een fluitje van een cent. Net als dat we in de andere talen de class library van het .NET Framework kunnen gebruiken en er eigen classes vanaf



Afbeelding 1. De combinatie van Micro Focus Net Express with .NET en Visual Studio .NET



Afbeelding 2. De integratie van Cobol in de .NET-omgeving

kunnen leiden, is dat ook voor Cobol-applicaties beschikbaar. .NET Framework-types stellen de ontwikkelaar in staat om verschillende, veelvoorkomende programmeertaken uit te voeren, inclusief taken zoals string management, collections en database-connectiviteit. Daarnaast bevat de class library types voor de ondersteuning van gespecialiseerde ontwikkelscenario's, bijvoorbeeld Windows GUI-applicaties (Windows Forms). Dit alles is vanuit Cobol aan te sturen.

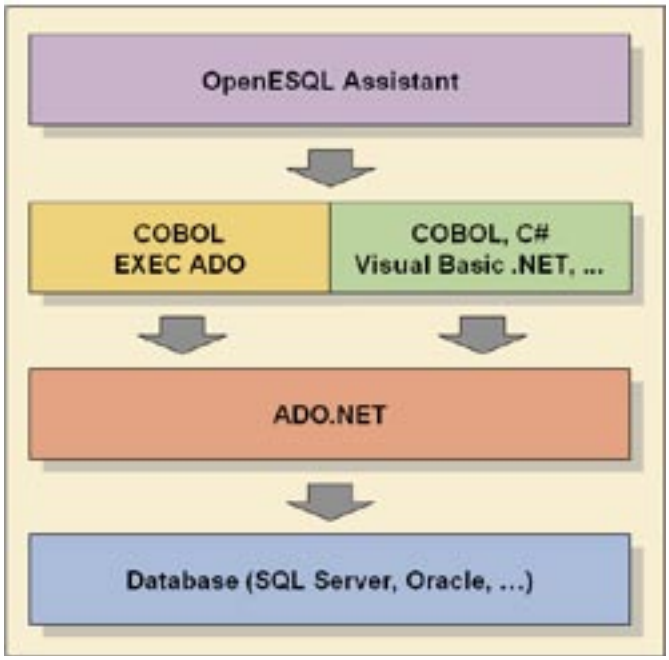
Zoals bij de meeste talen zijn er ook bij Cobol libraries met common code beschikbaar. Deze libraries bevatten routines voor het run-time systeem waarin onder andere typische Cobol-zaken zijn geïmplementeerd zoals de Cobol Intrinsic Functions, file handling en locking, error handling enzovoort. Dit kan gezien worden als de Cobol-specifieke uitbreiding op de .NET Common Language Runtime (CLR). Daarnaast zijn er de Cobol Intrinsic Functions en vele andere functies waarvoor de compiler geen inline code genereert, maar een library-call doet. Dat kan gezien worden als Cobol-specifieke uitbreidingen op de taalonafhankelijke functionaliteit die de .NET Framework-classes bieden. De Micro Focus Application Server implementeert tijdens run-time beide soorten benodigde functionaliteit, en draait zelf als managed code 'op' en in samenwerking met de .NET CLR.

Hybride (multi-language) applicaties

Door de toegang vanuit Cobol tot alle classes en methodes in het .NET Framework kunnen Cobol-programmeurs complete .NET-applicaties ontwikkelen. Dankzij de Common Language Specification is het mogelijk dat .NET-talen gecombineerd worden in één toepassing. Naast Cobol-modules (die misschien gemigreerd zijn) voor de business logica kunnen andere .NET-talen zoals Visual Basic. NET of C# gebruikt worden voor bijvoorbeeld de implementatie van de gebruikersinterface. Het grote voordeel hiervan is dat goedwerkende business logica wordt behouden en onderhoudbaar blijft, terwijl een andere taal gebruikt kan worden voor het uitbreiden van de applicatie. Dit alles binnen één en dezelfde IDE.

SQL en ADO.NET

Databasetoegang vanuit Cobol gebeurt meestal met embedded EXEC SQL-statements. Traditioneel vereist dat de inzet van een precompiler van de databaseleverancier die de SQL-statements vertaalt naar specifieke database API-calls. Met bijvoorbeeld OpenESQL van Micro Focus is er geen native database-precompiler meer nodig en is er toegang tot elke database waarvoor een ODBC-driver bestaat. Bij de introductie van het .NET Framework voegde Micro Focus gelijksoortige EXEC SQL-statement-ondersteuning voor ADO.NET dataproviders toe. OpenESQL biedt ook mogelijkheden om op de originele broncode (in plaats van de gegenereerde API-calling code) te debuggen.



Afbeelding 3. De relatie tussen OpenESQL en ADO.NET

ADO.NET is het onderdeel van de Microsoft .NET class library waarmee een applicatie data uit verschillende bronnen kan halen en kan manipuleren. ADO.NET is het mechanisme dat gebruikt wordt voor datatoegang via managed code.. Omdat datatoegang zeer gebruikelijk is voor Cobol-programma's is OpenESQL uitgebreid, waardoor Cobol-programmeurs toegang hebben tot ADO.NET datasets zonder dat ze de details van de .NET class library

```
Class-id. COBdemoNetmagazine as "COBdemoNETmagazine".

Repository.
  class DataSet as "System.Data.DataSet".

Factory.

Working-storage section.

  Exec ADO Declare patrons datatable
     (patron_number int16      not null,
      patron_name   string(30) not null)
  End-Exec.

*> Declare the dataset used to hold the patron data.
  Exec ADO
     Declare patron_info dataset for patrons
  End-Exec

End factory.
Object.

Method-id. GetPatronData as "GetPatronData".

Linkage Section.
01  Patron-Dataset object reference DataSet.

Procedure division returning Patron-Dataset.
  Exec ADO  *> create the dataset
     Initialize dataset returning :Patron-Dataset
  End-Exec

  Exec ADO *> Insert the record into the dataset
     Insert into patrons values (:ls-patron-number,
                                 :ls-patron-name)
  End-Exec

*> Accept all changes to the dataset so that when it is
*> sent back to us we can determine what has changed
  Exec ADO
     Accept changes for all datatables
  End-Exec
*> Unbind from the dataset. This will indicate that we
*> have finished any processing we are going to do on it.
  Exec ADO Unbind End-Exec

  Exit method.

End method GetPatronData.

End object.
End class COBdemoNetmagazine.
```

Codevoorbeeld 1.

Creëren, aanpassen en wegschrijven van dataset

hoeven te kennen. Daarnaast, in het kader van hergebruik zonder de broncode te wijzigen, is er, door middel van mapping, managed data-access aanwezig van bestaande OpenESQL-applicaties naar ADO.NET. Deze applicaties kunnen snel en eenvoudig uitgebreid worden door het gebruik van Exec ADO syntax gegenereerd door de OpenESQL wizard. Het formaat van EXEC ADO-statements komt overeen met dat van standaard ingebouwde SQL-statements, waardoor een ADO.NET-programma eenvoudig in Cobol gemaakt kan worden. Met de nieuwe EXEC ADO-syntax kunnen ADO.NET-datasets door programma's in verschillende talen gedeeld worden, waardoor de data-uitwisseling tussen samengestelde (multi-language) applicaties verbetert.

Met ADO.NET kunt u in connected mode werken met real-time toegang tot data, bijvoorbeeld in een database of dataset. Daarnaast kan ook in een 'disconnected mode' gewerkt worden, waardoor het mogelijk is om met een kopie van de data aan de slag te gaan. Alle wijzigingen worden onthouden, waarna ze teruggezet worden in de originele data-store. De relatie tussen OpenESQL Assistant, SQL en ADO.NET wordt geïllustreerd in het diagram in afbeelding 3. Codevoorbeeld 1 geeft aan hoe, in Cobol, met EXEC ADO een patroon dataset gecreëerd wordt. Ook laat het voorbeeld zien hoe een nieuw record aan de dataset wordt toegevoegd en dat uiteindelijk alle wijzigingen worden geaccepteerd, zodat die worden doorgevoerd in de resultaat dataset. Cobol is niet hoofdlettergevoelig, behalve uiteraard in literal strings zoals "COBdemoNETmagazine".

Cobol met XML

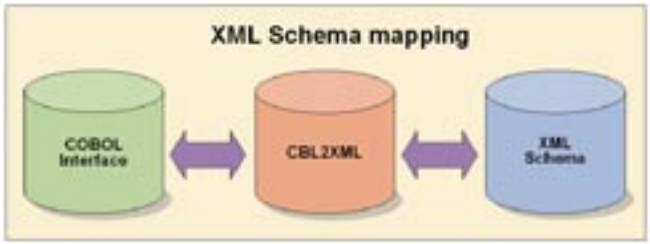
XML is inmiddels uitgegroeid tot de ruggengraat van e-commerce. Het voorziet in de mogelijkheid om data te delen tussen discrete applicaties op een loosely-coupled manier. Het is inmiddels redelijk eenvoudig om met Cobol-programma's XML-documenten/berichten te maken, te lezen en te wijzigen. Op basis van Cobol File Handling syntax is de Cobol-programmeur in staat XML-documenten op de bekende manier als 'gewone' data te bewerken via open/close-, read/write- en rewrite-statements. Syntax-uitbreidingen bieden mechanismen om dynamische XML-documenten te verwerken, XML-tags naar Cobol-datanamen om te zetten en XML-processen, zoals name spaces en attributen, te verwerken. Een

```
identification division.
program-id. calc-quote.

environment division.
input-output section.
file-control.
   select quote-info
      assign to "myfile"
      organization is xml
      document-type is external
        http://www.schemalocation.com/my_schema"
      file status is filestat.

data division.
file section.
Xd quote-info.
01 x-customer external-form       identified by 'customer'.
   02 cust-name         pic X(30) identified by 'name'.
   02 cust-age          pic 9(3)  identified by 'age'.
   02 cust-policy-value pic 9(12) identified by 'policyValue'.
   02 cust-policy-cost  pic 9(12) identified by 'policyCost'.
working-storage section.
01 filestat pic s9(9).

procedure division.
start-here.
open-xml-file.
   open i-o quote-info
   read quote-info.
Calculate-premium.
*> perform calculations here
update-XML-file.
   write x-customer key is cust-policy-cost
   write x-customer
   close quote-info
   goback.
```

Codevoorbeeld 2.
Bewerken van XML

```
class-id. Service1 as "Service1"
          inherits CLASS-WEBSERVICE.
environment division.
configuration section.
repository.
interface INTERFACE-ICONTAINER as "System.ComponentModel.IContainer"
  class CLASS-STRING as "System.String"
  class CLASS-WEBMETHODATTRIBUTE as
       "System.Web.Services.WebMethodAttribute"
  class CLASS-WEBSERVICE as "System.Web.Services.WebService".

object.
working-storage section.
01 components object reference INTERFACE-ICONTAINER.

method-id. NEW.
procedure division.
  invoke self "InitializeComponent".
end method NEW.

* Required method for Designer support - do not modify
* the contents of this method with the code editor.
method-id. "InitializeComponent" is private.
local-storage section.
procedure division.
end method "InitializeComponent".

* Clean up any resources being used.
method-id. "Dispose" override is protected.
local-storage section.
linkage section.
01 disposing usage condition-value.
procedure division using by value disposing.
  if disposing then
     if components not = null then
       invoke components "Dispose"
     end-if
  end-if.
  invoke super "Dispose" using by value disposing.
end method "Dispose".

* WEB SERVICE EXAMPLE
* The "HelloWorld" example service returns the string Hello World
* To build, uncomment the following lines then save and build the project
* To test this web service, press F5

*method-id. "HelloWorld" custom-attribute is CLASS-WEBMETHODATTRIBUTE.
*linkage section.
*01 lnkReturnString object reference CLASS-STRING.
*procedure division returning lnkReturnString.
*    set lnkReturnString to "Hello World".
*end method "HelloWorld".

 end object.
end class Service1.
```

Codevoorbeeld 3.
Webservice in Cobol



Afbeelding 4. XML schema mapping.

W3C compliant XML Schema kan naar en van een Cobol-interface geleid worden met de Cobol/XML conversietool (CBL2XML), zoals in afbeelding 4 weergegeven.

Codevoorbeeld 2 illustreert XML-verwerking met een Cobol-programma dat een XML-bestand leest, berekeningen uitvoert en daarna het XML-bestand bijwerkt.

Webservices in Cobol

Cobol heeft allerminst stilgestaan want webservices in Cobol zijn geen enkel punt. Het aanroepen van een webservice en het creëren ervan is mogelijk. Een bestaande Cobol-component kan zonder extra programmeerwerk getransformeerd worden tot een webservice. Dit vereenvoudigt hergebruik van bestaande businesslogica in en vanuit applicaties en systemen op verschillende platforms en in diverse talen. Om van Cobol-programma's webservices te maken hoeft de Cobol-programmeur geen expert in webservices, XML of .NET te zijn. Wanneer u een nieuw project in Visual Studio aanmaakt en als projecttype 'ASP.NET webservice' opgeeft, maakt de IDE een skelet basisclass waarin men de code voor de webservice kan plaatsen. De IDE creëert ook folders voor References en alle bestanden. De basisclass is vrij lang (± 50 regels code), maar blijft in het algemeen ongewijzigd. De programmeur hoeft alleen de laatste methode aan te passen door daarin de oude en vertrouwde businesslogica te plaatsen of aan te roepen of door nieuwe logica toe te voegen. Codevoorbeeld 3 toont hoe de basisclass er uitziet.

Samenvatting

Organisaties kunnen hun Cobol-applicaties die tot de kern van hun bedrijfssysteem behoren nu beschikbaar stellen aan een compleet nieuwe wereld van gebruikers en tegelijkertijd besparen op hardware en software. Naast kennis over bestaande code en data kunnen Cobol-programmeurs hun programmeerkennis in een .NET-omgeving benutten, zonder dat zij een langdurige leercurve moeten doormaken om zich op de hoogte te stellen van alle details van .NET. Dat is mogelijk doordat tools zoals Micro Focus Net Express with .NET wizards en templates bieden om met deze details om te gaan. Cobol-programmeurs hebben volledige toegang tot alle .NET-mogelijkheden, inclusief de voordelen van managed code, .NET Framework classes, ADO.NET, XML, webservices en uitwisselbaarheid met andere programmeertalen. Hergebruik van legacy, een kritische succesfactor vanuit een kostenperspectief, is zondermeer mogelijk.

Huib Klink is werkzaam als senior consultant bij Micro Focus (www.microfocus.com). Zijn e-mailadres is Huib.Klink@MicroFocus.com

Nuttige internetadressen

http://www.cobolportal.com/
http://www.mainframemigration.org/
http://support.microsoft.com/default.aspx?scid=kb;en-us;103226
http://www.microfocus.com/products/NetExpress/index.asp
http://supportline.microfocus.com/examplesandutilities/devforum.asp, met name de onderdelen over .NET (EI040 en EI050)