

RPG unter .NET mit Asna Visual RPG 7.1

Angriff auf IBM

IBMs Empfehlung, AS/400-Projekte unter Java und Websphere weiterzuentwickeln, war aufgrund des Aufwands erfolglos. Doch für die Anwender wird die Zeit knapp, denn das Ende von OS/400 zeichnet sich ab. Visual RPG bietet einen Ausweg über die .NET-Welt.

Die IBM-Rechnerfamilie S/3x gibt es schon seit den 1970er Jahren. Seitdem hat sie sich unter verschiedenen Namen präsentiert: AS/400, später iSeries und nun i5. Der Name änderte sich, doch die Anwendungsprogramme dieser Maschinen zeigen sich von der zurückliegenden 30-jährigen Entwicklung der IT weitgehend unbeeindruckt. iSeries-Anwender waren kaum zur Erneuerung gezwungen. Vielfach blieben auch die Software-Entwickler auf dem einmal erarbeiteten Wissensstand stehen.

Durch den aufgestauten Erneuerungsdruck stehen viele Unternehmen vor der Frage, wie sich ihre IT weiterentwickeln kann. Die meisten betreiben neben iSe-

ries auch Windows-Server und haben festgestellt, dass Windows-Systeme mittlerweile ebenso stabil laufen wie die Systeme der iSeries. Bei vielen Anwendern sind Windows- oder Webprojekte im Einsatz, die iSeries-Daten verwenden. Doch der Zugriff von Windows aus auf die iSeries ist nicht so einfach.

RPG versus Java

Die meisten Projekte auf der iSeries sind in der Sprache RPG erstellt (siehe Kasten „RPG auf IBM iSeries“). IBM legt den Anwendern seit Jahren einen Umstieg auf Java/Websphere nahe. Für viele Unternehmen ist eine Migration jedoch zu teu-

er, zumal die Folgekosten oft nicht abzusehen sind. Einen wesentlichen Anteil daran haben die Schulungskosten für die IT. Auch der lange Produktionsausfall ist beim Umstieg auf Java zu berücksichtigen, denn die Entwicklung von Projekten in Java unterscheidet sich erheblich von der RPG-Programmierung.

In dieser Situation bietet die .NET-Plattform zusammen mit den Produkten des Microsoft-Partners Asna [1] eine interessante Alternative. Dieses texanische Software-Unternehmen entwickelt seit den 80er Jahren Software für die iSeries und konnte viele große Anwender für seine Plattform gewinnen. Als strategischer Midrange-Partner von Microsoft entwickelte das Unternehmen die Sprache Visual RPG .NET. Derzeit ist Version 7.1 verfügbar. Die Bibliotheken aus Visual RPG stehen auch C#- und VB.NET-Programmierern zur Verfügung. Für die kommenden Versionen von Visual Studio

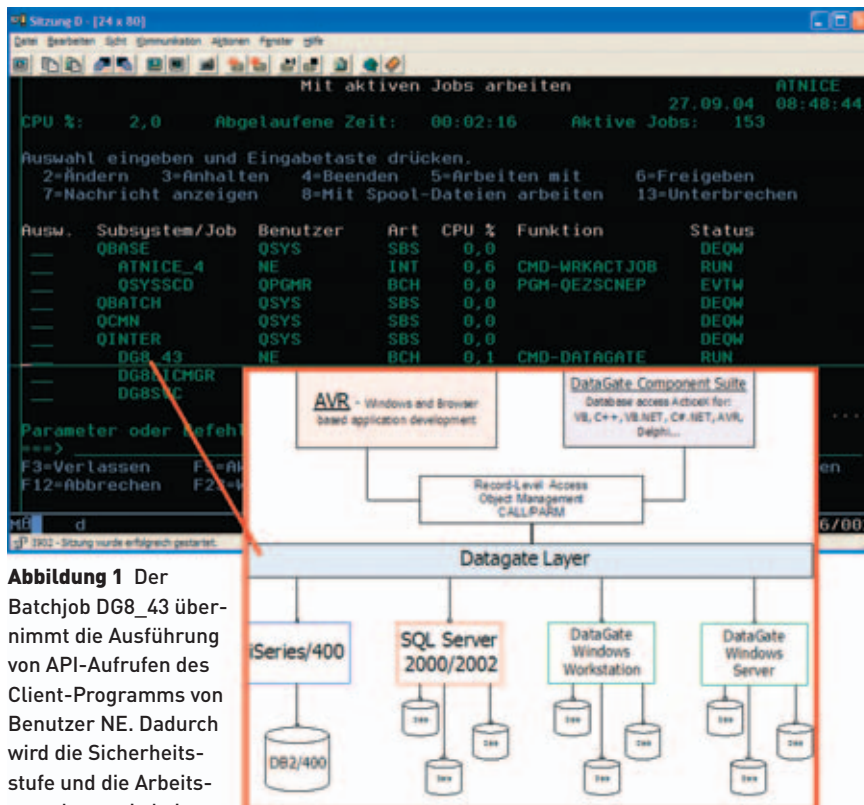


Abbildung 1 Der Batchjob DG8_43 übernimmt die Ausführung von API-Aufrufen des Client-Programms von Benutzer NE. Dadurch wird die Sicherheitsstufe und die Arbeitsumgebung wie bei interaktiven iSeries-Jobs eingestellt.

Auf einen Blick

Autor



Christian Neißl arbeitet als selbständiger Berater und SW-Entwickler. Seine Schwerpunkte sind Entwicklungen auf Rechnern der iSeries und .NET. Sie erreichen ihn unter c.neissl@niceware.at.

dotnetpro.code
A0503RPG



Sprachen RPG, Visual RPG .NET, Visual Basic .NET, VB6, XBase++

Technik iSeries, Desktop- und Webentwicklung

Voraussetzungen Visual Studio .NET, Visual RPG

RPG auf IBM iSeries

Report Program Generator (RPG)

RPG ist eine Programmiersprache von IBM, die ursprünglich für das Verarbeiten von Lochkarten gedacht war, hauptsächlich auf Midrange-Rechnern des Büromaschinenherstellers wie S/36 oder der AS/400 (iSeries). Die Hauptaufgabe der Sprache besteht im Lesen, Berechnen und der formatierten Ausgabe von Datenbankdateien.

RPG-Programme werden in einem so genannten Programmzyklus verarbeitet, dessen Schritte sich in der Regel für jeden Datensatz wiederholen. Dieser Programmzyklus übernimmt die vom Programmierer festgelegten Verarbeitungsregeln für die gewünschte Datenbankdatei. Der Programmierer braucht sich also nicht um den Zugriff auf jeden einzelnen Datensatz kümmern, das Programm wird automatisch auf jeden einzelnen Datensatz angewendet.

Im ersten Schritt werden allgemeine Informationen der Kopfzeile verarbeitet. Danach folgt das Lesen eines Datensatzes, dessen Verarbeitung und die Ausgabe der verarbeiteten Daten. Das Konzept des Programmzyklus ermöglicht keine komplexeren Berechnungen wie zum Beispiel rekursive Algorithmen.

Programme, die mit RPG geschrieben wurden, werden auch heute noch gepflegt. Die aktuelle Sprachversion heißt ILE RPG. ILE steht für Integrated Language Environment und bezeichnet die dazugehörigen Entwicklungskonzepte, die IBM 1993 eingeführt hat. Gleichzeitig hatte IBM die Sprache RPG erheblich erweitert.

Asna Visual RPG (AVR)

Unter diesem Namen bietet die Firma Asna eine Entwicklungsumgebung mit einem RPG-Compiler zum Erstellen von Web- und Windows-Anwendungen in Form von *exe*- oder *dll*-Dateien an. AVR-Programme bieten Zugriff auf Datenbanken der iSeries/400 von IBM und auf Microsoft SQL Server. Der Zugriff erfolgt dabei transparent, das heißt eine ursprünglich für iSeries/400 vorgesehene Anwendung lässt sich auch mit MS SQL Server verwenden. AVR unterstützt XML, SOAP und Active Server Pages.

AVR for .NET

Inzwischen hat Asna das .NET Framework in AVR integriert. Dieses Entwicklungstool integriert sich in Visual Studio .NET 2003 und führt AVR, Web und .NET unter Windows zusammen. Das IDE-Plug-In erzeugt verwalteten Code, die Verbindung zu Datenbanken der iSeries/400-Rechner erfolgt durch Asnas DataGate.

DataGate Komponente Suite

Asnas Komponenten-Suite DataGate bietet Schnittstellen zur iSeries für Sprachen wie C# oder Visual Basic an. Die dabei verwendeten DLLs sind die Basis für den Zugriff auf AVR. Die Suite ist für .NET und COM-basierende Sprachen verfügbar. DataGate bietet einen direkten Lese-/Schreibzugriff auf Satzebene und abstrahiert so den Datenbankzugriff von der Datenbank selbst. Unterstützt werden DB2/400, Microsoft SQL Server und Asnas eigene Acceler8DB. SQL ist dabei nicht notwendig, als Übertragungsprotokoll dient TCP/IP.

(Whidbey) und SQL Server (Yukon) wurde Version 8.0 angekündigt.

Die Middleware DataGate bildet dabei die Datenbankschnittstelle zu iSeries, SQL Server oder Acceler8DB. Ein .NET-Client kommuniziert über ein internes Protokoll mit DataGate und übergibt API-Aufrufe an die Datenbankschnittstelle. DataGate gibt den Aufruf an das Betriebssystem weiter und sendet die angeforderten Datensätze an den Client zurück. Der Server-Job läuft unter dem am Client angemeldeten User. Dies gewährleistet, dass Umgebung und Sicherheit der inter-

aktiven iSeries-Sitzung entspricht. Der Datenbankzugriff erfolgt dabei auf Satzebene, ähnlich wie in DAO mit Befehlen (OpCodes), wie sie unter RPG üblich sind.

Zusätzlich zu den genannten Datenbanksystemen will Asna 2005 auch Oracle-Datenbanken unterstützen. Die Architektur des Systems zeigt Abbildung 1.

Das Konzept der AS/400

Die AS/400 wurde in den 70er Jahren entwickelt. Sie war eines der ersten, wenn nicht überhaupt das erste System das die

Hardware von der darauf laufenden Software durch eine Abstraktionsebene trennte. Sie war auch mit dem Einspeicherkonzept führend, indem das Betriebssystem entschied, welche Anteile aus dem Arbeitsspeicher im Memory behalten werden und was auf die Plattensubsysteme auszulagern ist.

Das zum Rechner gehörende Betriebssystem OS/400 geht, wie die meisten Systeme aus dieser Zeit, sehr sparsam mit Maschinenkapazitäten um. Es trennt den Programmspeicher und den Datenspeicher pro Sitzung. Somit ist der Programmcode für alle Anwender, die gleichzeitig mit diesem Programm arbeiten, nur einmal im Speicher vorhanden. Die AS/400 war auch eines der ersten Systeme, die den 64-Bit-Power-Prozessor verwendeten. Für den Anwender machte sich dies nur durch die gestiegene Leistung bemerkbar.

Datenbank und Programmierung der AS/400

Die iSeries-Datenbank hat sich seit den Anfängen nicht wesentlich verändert. Sie besteht nach wie vor aus Tabellen mit einem fixen Satzformat, so genannten physischen Files. Zugriffswege zu diesen Tabellen mit einem anderem Schlüsselaufbau werden logische Files genannt. Dahinter stehen eigene Indextabellen, in denen sich Datensätze per fixer, in der Tabellen-Source (DDS) hinterlegter Definition auswählen lassen.

Es ist durchaus Ansichtssache, ob dieses System als Datenbank gelten kann. Das fixe Satzformat bewirkt, dass beim Erstellen eines Satzes über einen RPG-Befehl grundsätzlich alle Datenfelder initialisiert werden. Somit bleibt die datenbanktypische Forderung nach einem Nullwert für eine nicht vorhandene Information unerfüllt.

Ein in alten Anwendungen verbreiteter Datentyp ist das gepackte numerische Datenfeld, das in einem Byte zwei numerische Zeichen speichert. Das Vorzeichen ist im äußeren rechten Zeichen des letzten Halb-Bytes des Feldes enthalten. Dieser Datentyp war in früheren Zeiten auf EBCDIC-basierenden Systemen Standard für die Darstellung numerischer Werte. Er benötigt ungefähr die Hälfte des Speicherbereichs gegenüber einer ungepackten Darstellung.

Im Gegensatz zu ASCII-basierenden Systemen verwendet die iSeries ein Bibliothekssystem, das man sich als einstu-

figes Dateisystem vorstellen kann. Es gibt unter einer Bibliothek keine Verzeichnisse. Dafür gibt es in der Datei selbst die Möglichkeit, Daten in so genannten Teildateien zu speichern. Jede Datei kann bis zu 32 767 Members enthalten.

Datentypen wie Date, Timestamp, LOB, Integer und so weiter kennt die Datenbank mittlerweile, sie werden aber von den wenigsten iSeries-Programmierern verwendet.

Die Programmierung unter OS/400 erfolgt einerseits über die Steuersprache CL, andererseits über die Dialogprogrammierung mit RPG, COBOL oder PLI. CL dient zum Steuern von Jobs und zur Manipulation von Dateien, ähnlich Batch-Programmen. Die iSeries betrachtet dabei jede Peripherie als Datei, das heißt, Ausgaben an Drucker und Bildschirm werden über Dateien abgewickelt.

Neue Anwendungen für die iSeries gibt es so gut wie keine. Die Anwender sind in ihrem IT-Umfeld quasi von Windows-Anwendungen umzingelt und tauschen Daten mit diesen beispielsweise mittels FTP aus. Software-Häuser im Microsoft-Umfeld stehen häufig vor dem Problem, dass die benötigten Daten auf einer iSeries liegen. Auf ihrer Seite fehlen aber sehr oft das Wissen und die geeigneten Werkzeuge, um Lösungen zu liefern, die den Kunden zufrieden stellen.

Problematisch ist der zu den iSeries mitgelieferte ODBC-Treiber. Er verführt manche Entwickler dazu, die iSeries wie einen SQL Server zu verwenden. Für größere Anwendungen ist die Leistung dieses Treibers jedoch indiskutabel. Außerdem ist der Treiber insgesamt doch zu instabil.

Stored Procedures können das Problem lösen. Das Windows-Programm ruft dazu über *CALL/PAAM* ein iSeries-Programm auf, das die Datenzugriffe und die Business-Logik kapselt. Dies ist der einzige Weg, um eine vernünftige Leistung ohne Einsatz zusätzlicher Software zu erreichen. Die wesentlichen Nachteile sind:

- Es muss eine Parameterliste als Schnittstelle zwischen den Anwendungen definiert werden.
- Die Parameter sind in Typ und Länge fix, die maximale Parametergröße ist 64 KByte.
- Gepackte Datenfelder können nicht direkt in einer Struktur übergeben werden, da der hexadezimale Wert 00 vom ASCII-System als String-Ende interpretiert wird. Somit gehen alle Daten nach diesem Wert verloren.

Abbildung 2
Der Migrations-Prozess.

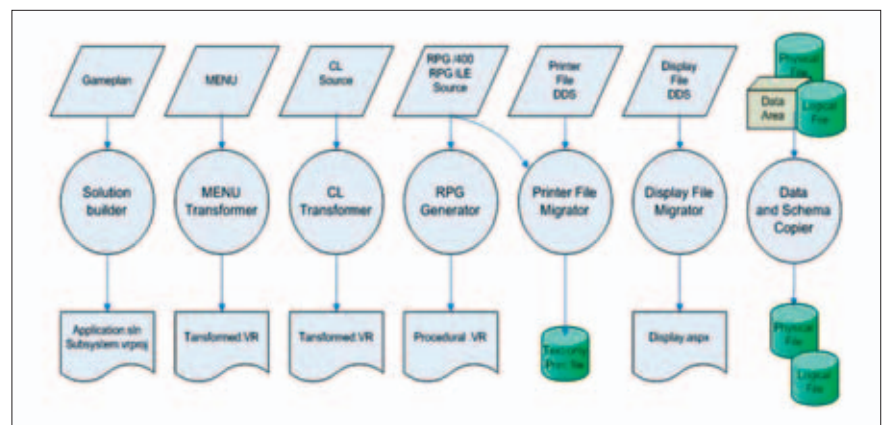
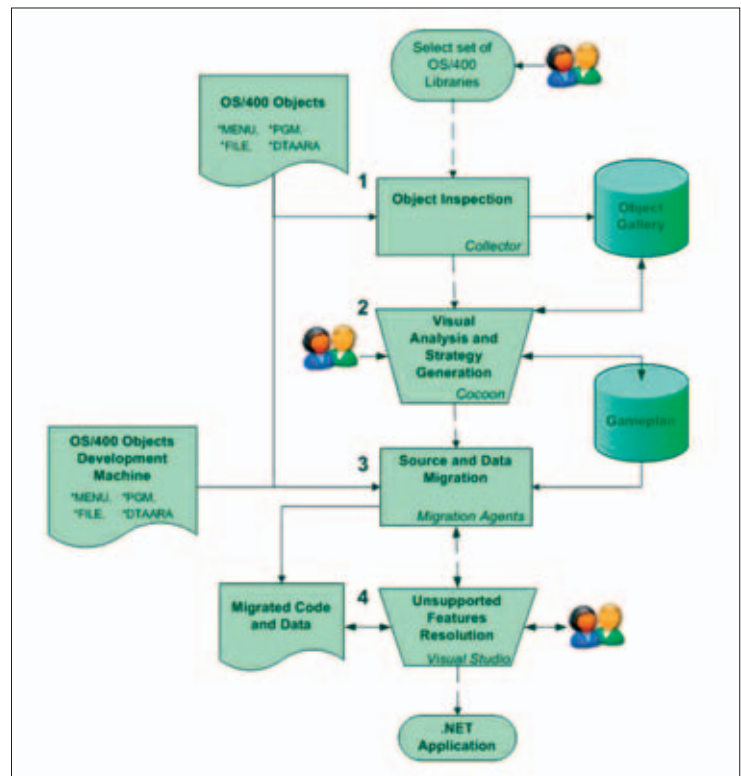


Abbildung 3 Monarch-Assistenten unterstützen beim Migrieren von iSeries-Anwendungen.

- In den seltensten Fällen wird die Entwicklung auf beiden Plattformen durch denselben Programmierer abgedeckt.
- Die zurückgegebene Parameterliste wird von beiden Programmen bearbeitet. Diese Doppelgleisigkeit fordert erheblichen Aufwand bei Änderung und Erweiterung.

Letztlich ergeben sich aus diesen Nachteilen Kosten in unbekannter Höhe.

Visual RPG füllt die Lücke

Asna Visual RPG ist derzeit der einzige RPG-Compiler für .NET. Dieser Compiler

ist vollständig in Visual Studio .NET integriert und erzeugt einwandfreien MSIL-Code. Die aussagekräftigen Hilfetexte sind auch in Deutsch vorhanden. Die Größe der *exe*-Datei eines Beispielprogramms, das auf die iSeries zugreift, Daten liest und anzeigt, beträgt 44 KByte. Visual RPG lebt die vom RPG-Programmierer gewohnten Konzepte weiter und bietet somit eine sehr gute Basis, um Anwendungen zu modernisieren. Die Sprache erfordert beim Umstieg keinen Neuanfang, wie Java ihn erfordern würde. Für Unternehmen bedeutet der Einsatz von Visual RPG, dass das Wissen und die Erfahrung der Mitarbeiter weiterhin genutzt und gefördert werden.

Als Alternative zu ODBC bietet Asna für Windows-Entwickler die DataGate Component Suite, um auf Programme und Daten der iSeries in Echtzeit zuzugreifen. Es handelt sich dabei um dieselben Bibliotheken, wie sie auch Visual RPG verwendet. Sie stellen Objekte zur Verfügung, die den Zugriff auf die iSeries in professioneller Qualität garantieren.

Die Migration zu Java oder C

In den vergangenen Jahren ist ein regelrechter Kampf um die iSeries-Plattform ausgebrochen. Einige Produkte bieten eine Umwandlung der Quellcodes von RPG nach Java oder C/C++ an. Das Problem des Anwenders ist dabei, dass er den Code weiter betreuen muss. Die Praxis hat gezeigt, dass viele Programme über Jahre hinweg „zu Tode gewartet“ wurden. Sie sind schon im RPG-Code nicht mehr nachzuvollziehen. Die Migration in eine andere Sprache würde die Lesbarkeit kaum erhöhen. Dazu kommt, dass RPG-Programmierern die Sprache fremd ist. Manche Merkmale von RPG lassen sich außerdem nicht migrieren, da es kein Äquivalent dazu in der Zielsprache gibt.

Mit Monarch, ebenfalls von Asna, steht für den Anwender ein Set aus Programmen und Werkzeugen bereit, das die Migration von iSeries-RPG-Applikationen auf Visual RPG für die .NET-Plattform unterstützt. Der Ausgangspunkt einer Migration ist die Object-Gallery, in der die iSeries-Objekte aufgelistet sind. Ein so genannter Game-Plan beschreibt die Methodik der Migration. Ein Beispiel dafür ist in Abbildung 2 zu sehen.

Monarch stellt verschiedene Assistenten bereit (siehe Abbildung 3):

- Der Solution-Builder erstellt ein .NET-Projekt.
- Der Menu-Transformer erzeugt Menüs als Visual-RPG-Programme.
- Der CL-Transformer erstellt Visual-RPG-Programme aus CL-Code.
- Der RPG-Generator generiert prozedurale Visual-RPG-Programme aus RPG-Code.
- Der PrintFile-Migrator wandelt iSeries-Druckdateien und die O-Bestimmungen von RPG-Programmen in Asna-Druckdateien um.
- Der DisplayFile-Migrator baut ASPX-Seiten aus den 5250-Display-Files der iSeries auf.
- Der Data and Schema-Copier migriert Dateien und Data-Areas in Dateien.

Abbildung 4 Das Monarch-Framework steht parallel zu ASP.NET und deckt die Anforderungen der Programme aus dem speziellen iSeries-Umfeld an die .NET-Anwendungen ab.

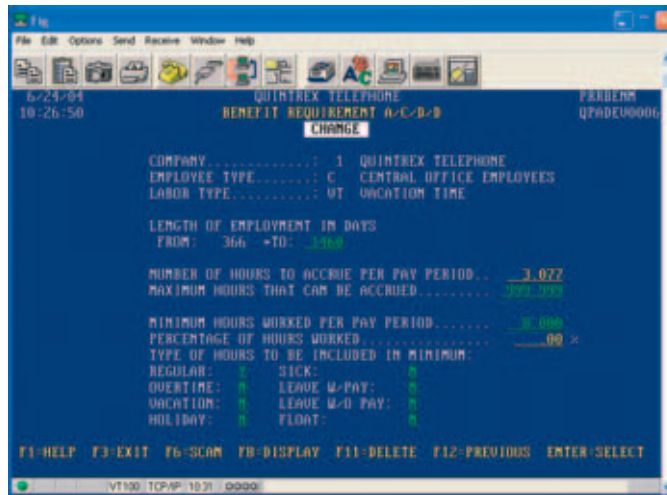
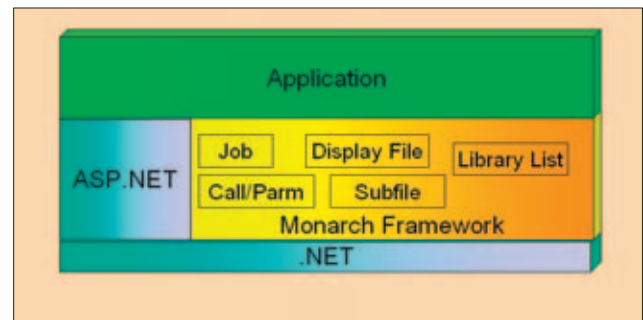


Abbildung 5 So sieht die Oberfläche vor der Migration aus.



Abbildung 6 Eine ASP-Seite nach der Migration mit Monarch.

Zwischen Anwendung und .NET Framework liegt das Monarch-Framework. Es vermittelt zwischen dem spezifischen iSeries-Umfeld und der .NET-Welt. Das Framework enthält Funktionen zur Jobsteuerung, von der Bibliotheksliste bis zum Dialog mit ASP-Anwendungen unter Verwendung der iSeries-Spezifika. Es ermöglicht RPG-Dialog-Befehle wie zum Beispiel *EXFMT*, Subfile-Handling und Message-Dateien. Abbildung 4 verdeutlicht, dass die Entwickler bei Asna das Ziel

einer vollständigen Integration der migrierten iSeries-Anwendungen in die .NET-Welt verfolgen. Es wird keine spezielle Umgebung rund um .NET aufgebaut. Die Abbildungen 5 bis 7 zeigen die Oberfläche in den einzelnen Migrationsstadien. Abbildung 5 zeigt die Anwendung auf der iSeries, Abbildung 6 die Bildschirmmaske nach der Migration mit Monarch. Abbildung 7 stellt dieselbe Bildschirmmaske wie Abbildung 6 dar, jedoch unter Verwendung von CSS (Stylesheets).

Das Ziel ist sauberer Code

Die Sprache RPG hat im Laufe ihrer fast 40-jährigen Geschichte Merkmale angesammelt, die nicht in eine zeitgemäße Umgebung passen. Daher kann die Migration keine Push-Button-Solution sein.

Unter .NET ist bekanntlich alles Objekt. Daraus ergeben sich zwangsweise Probleme mit überlappenden Datenstrukturen, mit Dateien, die über fixe Satzlänge anstatt feldorientiert eingelesen werden und vieles mehr.

Sehr oft ist es dem RPG-Programmierer gar nicht bewusst, dass er „dirty Code“ produziert. Bei den meisten Migrationen müssen Code-Blöcke manuell aufgelöst werden. Doch die Mühe lohnt sich, wie folgende Vorteile belegen:

- Die Anwendung lebt auf der modernen Entwicklungsumgebung weiter.
- Sie ist weiterhin zu warten, da die Sprachelemente bekannt sind.
- Visual RPG ist für VB.NET- und C#-Programmierer leicht zu erlernen. Der Generationswechsel ist für Programmierer daher kein großes Problem.
- Die Anwendung lässt sich mit wenigen Schritten weiter modernisieren, beispielsweise durch die einfache Integration der iSeries-Daten in die Office-Produktfamilie.
- Visual RPG ermöglicht den Zugriff auf Datenbanken von iSeries und SQL Server und die Migration der Daten aus dem einen System in das jeweils andere.

Beispiel Stammdatenwartung

Visual RPG 7.1 enthält zahlreiche Anwendungsbeispiele sowohl für Windows als auch für das Web. Die Anwendung *Cust-LookUp* ist ein Beispiel für eine typische Stammdatenwartung, die in Abbildung 8 zu sehen ist.

Über das *DataGrid* wird ein Stammdatensatz zur Bearbeitung ausgewählt. Die-



Abbildung 7 Eine ASP-Seite nach Migration, die mit Stylesheets aufgepeppt wurde.

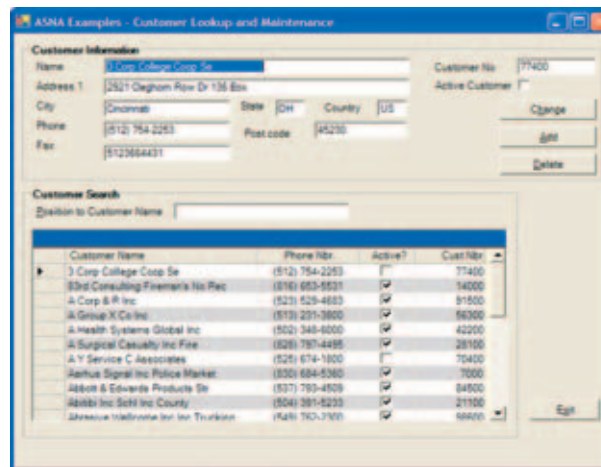


Abbildung 8 Beispielanwendung Stammdatenwartung.

se Methode ist auch auf der iSeries üblich, das Äquivalent zum *DataGrid* ist in der 5250-Welt das Subfile. Es handelt sich dabei um eine Datei, in der die Datensätze über eine relative Satznummer als Schlüssel organisiert sind. Unter Visual RPG übernimmt der Objekttyp *Memory-*

file diese Aufgabe. Listing 1 zeigt die F-Specs.

Die Beschreibung des *MemoryFiles* liegt, wie jede andere Datei, in der Datenbank und wird über den Dateinamen angesprochen. Zur Laufzeit wird das *DataGrid* mit folgendem Code an das *DataSet* des *Memoryfiles* gebunden:

Listing 1

F-Specs in Visual RPG.

```
Dc1DB ProdDB DBName("DG NET Local")
Dc1DiskFile Name(Customer_Num) Type(*Update) AddRec(*Yes) Org(*Indexed) Impopen(*No) DB(ProdDB)
  File("**LIBL/CMastNew1")
Dc1DiskFile Name(Customer_Name) Type(*Input) Org(*Indexed) Impopen(*No) DB(ProdDB)
  File("**LIBL/CMastNew2")
Dc1MemoryFile Name(memCustomer) Impopen(*No) DBDesc(ProdDB) FileDesc("**LIBL/CMastNew")
```

```
Connect ProdDB
Open Customer_Num
Open Customer_Name
Open memCustomer
```

```
dgCustomers.SetDataBinding(
  memCustomer.dataSet, ("RCMaster")
LoadGrid()
```

Die Funktion *LoadGrid()* füllt den *MemoryFile* in einer Schleife mit Daten:

Listing 2

Das Changed-Ereignis des DataGrids ruft die Funktion LoadData() auf.

```

BegSr dgCustomers_CurrentCellChanged
Event(*this.dgCustomers.CurrentCellChanged)
DclSrParm sender Type(*Object)
DclSrParm e Type(System.EventArgs)
LoadData(dgCustomers.CurrentCell.
RowNumber) // retrieve data from dataset
EndSr

BegSr LoadData Access(*Private)
DclSrParm inRow Type(*Integer4)

CMCustNo = dgCustomers.Item(dgcustomers.
CurrentRowIndex, 3).ToString()
Chain Customer_Num Key(CMCustNo)
Err(*Extended)
// Chain memCustomer Key(inRow + 1)

tbxCustNo.Text = CMCustNo
tbxName.Text = CMName
tbxAddr1.Text = CMAddr1
tbxCntry.Text = CMCntry
tbxCity.Text = CMCity
tbxState.Text = CMState
tbxZip.Text = CMPostCode
tbxPhone.Text = CMPhone
tbxFax.Text = CMFax
chkActive.Checked = CMActive

EndSr
    
```

```

BegSr LoadGrid Access(*Private)
DclFld x Type(Integer4)
memCustomer.Dataset.Clear()
Do FromVal(0) ToVal(24) Index(x)
Read Customer_Name
If Customer_Name.IsEOF
Leave
EndIf
Write memCustomer
EndDo
EndSr
    
```

Beim Ereignis *Cell-Changed* in dem Beispiel wird der ausgewählte Datensatz im Detail angezeigt, siehe Listing 2. In der Routine *LoadData()* hat sich in den Beispielen ein Fehler eingeschlichen, der hier korrigiert ist. Mithilfe des Befehls *CHAIN* und des Parameters *inRow* erfolgt über die relative Satznummer der Zugriff auf den Satz im *Memoryfile*. Das funktioniert, solange sich die Sortierung des *DataGrids* nicht ändert, denn dadurch ändert sich auch die Reihenfolge der Sätze im *DataGrid* und die relative Satznum-

Listing 3

Deklaration einer Parameterliste zum Aufruf eines iSeries-Programms.

```

// Note that Timeofday is packed.
DclPlist Name(Parms)
DclParm CustName Len(40) DBDirection(*Output) // only from iSeries
DclParm Timeofday Len(6,0) Type(*Packed)
DBDirection(*Output) // only from iSeries
DclParm Quit400App Len(1) DBDirection(*Input) // only to iSeries If "1" then CALL400 will terminate
    
```

Listing 4

Aufruf eines iSeries-Programms mit Übergabe der Parameter aus Listing 5.

```

Try
CustName = %Trim(txtName.Text)
Call Pgm("**Lib1/Call400") ParmList(Parms) DB(ProdDB)
Catch Err Exception
ShowStatus (CallErrMsg + NL + Err.Message)
LeaveSr
EndTry
    
```

mer stimmt nicht mehr mit der Satznummer im *Memoryfile* überein. Die Korrektur ist in diesem Fall sehr einfach, indem der Zugriff auf die Stammdatei über den Key aus Spalte 3 des *DataGrids* erfolgt. Dieser Fehler findet sich übrigens in den meisten Beispielanwendungen mit *DataGrid*.

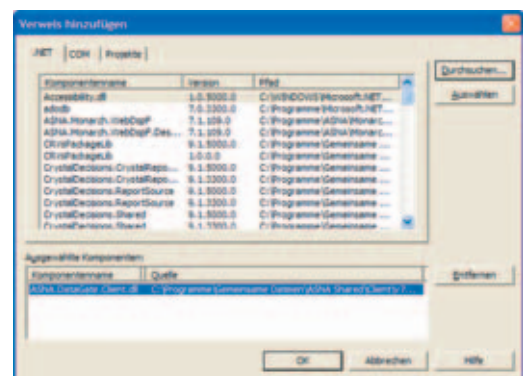
Aufruf eines iSeries-Programms mit Visual RPG

Das Beispiel *Call400Pgm* demonstriert, wie Programme auf der iSeries aufgerufen werden. Der wesentliche Unterschied zum Aufruf über OLE-DB ist, dass bei *DataGate* jeder Datentyp als Parameter verwendet werden kann. Der Aufruf aus AVR-.NET ist somit identisch mit dem Aufruf durch ein iSeries-RPG-Programm.

Zu beachten ist jedoch, dass die Übersetzung von EBCDIC zu ASCII auf dem iSeries-Rechner erfolgt. Das birgt Tücken in sich. Wenn eine Datenstruktur als Parameter mit dem Typ *Character* übergeben wird, dürfen darin keine gepackten Felder enthalten sein. Der Grund ist, dass gepackte numerische Felder häufig – wie schon erwähnt – den hexadezimalen Wert *00* enthalten, ASCII-Systeme diesen aber als String-Ende interpretieren und somit alle im String folgenden Zeichen ignorieren. Ein gepacktes Feld als Parameter mit Typ *Packed* interpretiert *DataGate* jedoch richtig und konvertiert es korrekt, siehe Listing 3.

Listing 4 zeigt den Aufruf des iSeries-Programms mit Parameterübergabe, wobei die iSeries über den Datenbanknamen angesprochen wird. Die Datenbank

Abbildung 9 Ein Verweis auf die Bibliothek *Asna.DataGate.Client* macht die Asna-Funktionen in jeder .NET-Sprache zugänglich.



Listing 5

Objektdeklaration für den Programmzugriff aus VB.NET.

```
'Declare the parm types
Dim ModeParm As New ProgParmType("ParmMODE", 0, FieldType.NewChar(4))
Dim DataParm As New ProgParmType("ParmData", 0, FieldType.NewChar(1000))
Dim ErrorParm As New ProgParmType("ParmError", 0, FieldType.NewChar(2000))
Dim RCParm As New ProgParmType("ParmRC", 0, FieldType.NewChar(1))

'Declare the parms; assign the types and the direction the parm goes
Dim dbConn As AdgConnection
Dim bChgConnection As Boolean = True
Dim bConnectionError As Boolean = False
Dim ParmMODE As New ProgParm(ModeParm, DataDirection.Input)
Dim ParmData As New ProgParm(DataParm, DataDirection.InputOutput)
Dim ParmError As New ProgParm(ErrorParm, DataDirection.Output)
Dim ParmRC As New ProgParm(RCParm, DataDirection.Output)
```

wird am Client oder am Server – abhängig vom Applikationstyp – definiert und lässt sich zur Laufzeit einstellen.

Dieses Beispiel nutzt eine weitere beachtenswerte Funktion: das RPG-Merkmal *LastRecord* über den Parameter *Quit400App*. Der Schalter **INLR* steuert am Ende der Programmlogik, ob das Programm für den Job aus dem Arbeitsspeicher der iSeries entfernt wird. Ist **INLR* auf **ON*, also *True* gesetzt, wird das Programm beendet. Andernfalls werden die Parameter zurückgegeben und das Programm bleibt offen. Wird die Schaltfläche *CALL400* gedrückt, bleibt das Programm in der Lese-Schleife und holt den nächsten Datensatz. Wird die Schaltfläche *TERMINATE* aktiviert, sendet das .NET-Programm eine Beendigungsanforderung an das iSeries-Programm.

Dies bewirkt einen Aufruf, wobei der Parameter *Quit400App* auf **ON* gesetzt ist. Bei erneutem Aufruf mit *CALL400* startet das Programm vom Beginn der Datei oder dem angegebenen Aufsetzpunkt.

Aufruf eines iSeries-Programms aus VB.NET

Falls Visual RPG .NET installiert ist, lassen sich die Bibliotheken jeder .NET-Sprache nutzen. Es genügt, im Projekt einen Verweis auf die Bibliothek *ASNA.DataGate.Client* einzurichten. Die Parameterdeklarationen zum Programmaufruf sind in Listing 5 dargestellt. Listing 6 erzeugt ein Programmobjekt. Dabei fängt es einen möglichen Fehler in der Verbindung zur iSeries über das Objekt *dbConn* ab. Danach werden die Parameter dem Programmobjekt zugewiesen und das iSeries-Programm aufgerufen. Abhängig vom Zielsystem – iSeries oder SQL-Server – werden Programmaufrufe auf Programmobjekte (iSeries) oder Stored Procedures (SQL-Server) bezogen.

Fazit

Visual RPG .NET ist eine ideale Möglichkeit, Investments in die iSeries-Welt in

eine zeitgemäße IT-Landschaft zu retten. Durch die Abstraktion des RPG-Codes von der Hardware-Plattform gewinnt der Anwender viele neue Möglichkeiten. Er kann die iSeries weiter verwenden und seine Anwendungen schrittweise modernisieren. Mit Monarch kann er aber auch komplett von der iSeries migrieren.

Besonders interessant ist dieses Angebot für Software-Häuser, die ihre Anwendungen auf eine neue Plattform bringen müssen. Für Unternehmen, die iSeries einsetzen, bietet Visual RPG die Möglichkeit, die Anwendungen mitsamt der Betreuungsmannschaft auf die .NET-Plattform überzusiedeln. Für die iSeries-Programmierer ist Visual RPG eine Herausforderung. Die Schreibweise der Befehle (OpCodes) wirkt zwar vertraut, die Umgebung an sich ist aber für einen an den „Green Screen“ gewöhnten Programmierer gewöhnungsbedürftig.

Die iSeries kann sich dem Erneuerungsbedarf, der sich in der Midrange-Welt angestaut hat, nicht verschließen. Vor einigen Jahren wurden iSeries-Programme zwar mit webähnlichen Oberflächen versehen, unter der hübschen Maske blieb aber alles beim Alten und eine echte Erneuerung wurde dadurch nur auf unbestimmte Zeit verschoben.

Ein nicht zu unterschätzender Druck kommt auch aus dem Umfeld, wenn Anwender Funktionen fordern, die erst unter .NET einfach umzusetzen sind. Dazu gehören zum Beispiel der B2B-Datenaustausch, das Anbieten und Nutzen von Web Services oder der Datenaustausch mit Office-Anwendungen. Grundsätzlich ist dies auch unter OS/400 möglich. Einfacher, moderner und somit kostengünstiger geht es aber mit Visual RPG .NET. |||||

[1] Asna, www.asna.com

Listing 6

Aufruf eines iSeries-Programms aus VB.NET.

```
'Declare a program object
Try
    Dim Prog As New As400Program(dbConn, txtAPIProgram.Text)
Catch Exception As dgException
    ShowStatus(ConnErrMsg + vbCrLf + Exception.Message)
    MsgBox(Exception.Message)
    bConnectionError = True
End Try

'Fill Parameter
Prog.ObjectToParm(ParmMODE, txtMode.Text, 0)
Prog.ObjectToParm(ParmData, txtDATA.Text, 0)
Prog.ObjectToParm(ParmError, txtErrorMessageDetails.Text, 0)
Prog.ObjectToParm(ParmRC, txtReturnCode.Text, 0)

ShowStatus(CallMsg + txtAPIProgram.Text)

'Attempt to execute the program on the 400, catch any errors
Try
    Prog.Execute()
Catch Exception As dgException
    ShowStatus(CallErrMsg + vbCrLf + Exception.Message)
    MsgBox(Exception.Message)
End Try
```