# Von SQLWindows nach .NET

## Ein automatisiertes Verfahren unterstützt Entwickler bei der Übertragung von 1,2 Millionen Codezeilen

von Frank Wuttke

Das Software-Haus *nGroup* portierte eine komplexe ERP-Anwendung von SQLWindows nach .NET. Die Motivation für diese Portierung war, zukünftig eine moderne Entwicklungsumgebung verwenden, fertige Komponenten in die eigene Software integrieren und Projektanpassungen für Kunden einfacher realisieren zu können.

Was fängt man mit einer typischen „Legacy"-Anwendung an, die in 17 Jahren auf über 1,2 Millionen Codezeilen angewachsen ist und die jetzt nach .NET portiert werden soll? An eine manuelle Portierung war aus Kosten- und Zeitgründen nicht zu denken, eine Neuentwicklung wurde ebenfalls ausgeschlossen. Nach längeren Diskussionen entschied sich das Entwicklerteam für eine automatisierte, softwaregestützte Portierung mit anschließender manueller Nachbearbeitung. Als Werkzeug wurde *IcePorter* gewählt, das speziell für die Konvertierung von Legacy-Anwendungen nach .NET geschaffen wurde [4]. Die sehr unterschiedlichen Möglichkeiten auf der Quell- und Zielplattform führten zu zahlreichen und umfangreichen Nacharbeiten. Die Bereiche Polymorphie, DB-Zugriffe, Receive-Parameter von Windows, fehlende Berichtsengines und die Nachprogrammierung von Customizing-Einstellungen waren dabei am aufwändigsten. Wesentlich einfacher gestaltete sich die Portierung vorhandener und selbst entwickelter COM-Objekte nach .NET. Nach Abschluss der insgesamt neunmonatigen Portierung plant das Unternehmen als nächsten Schritt ein Refactoring des Quellcodes und die Weiterentwicklung der Lösung unter Nutzung jener Möglichkeiten, die für die reine Portierung aus Zeitgründen nicht berücksichtigt werden konnten.

### Fortschritt tut Not

Das Produkt *eEvolution* des Software-Hauses *nGroup* aus *Hildesheim* (Deutschland) ist eine ERP-Anwendung, die aus der betriebswirtschaftlichen Lösung Microsoft *Apertum* entstanden ist und von mehr als 1.000 mittelständischen Unternehmen eingesetzt wird. Um von den in den letzten Jahren stark gestiegenen Möglichkeiten moderner Plattformen und Werkzeugen zu profitieren, entschied sich die *nGroup* für die Portierung der umfangreichen ERP-Lösung, die aus einer ganzen Reihe komplexer Anwendungen mit insgesamt mehr als 1,2 Millionen Zeilen Programmcode besteht. Die Ausgangsplattform ist die 4GL-Sprache *SQLWindows* von *Gupta*. Seit fast 20 Jahren wurden mit dieser Sprache Geschäftsanwendungen in den unterschiedlichsten Branchen entwickelt. Das Werkzeug bot damals durch eingebettetes SQL erhebliche Vorteile im Zugriff auf Datenbanken gegenüber anderen Werkzeugen, wies aber auch eine eigenwillige IDE auf. *SQLWindows* wurde in den letzten Jahren von mächtigeren Sprachen wie Java und C# eingeholt. Qualifizierte Mitarbeiter sind für *SQLWindows* zudem nur schwer zu finden, die Zukunft der Sprache ist ungewiss. Die Entscheidung für eine Portierung hatte daher nicht nur technische Gründe.

Als Zielplattform wurde das .NET Framework 2.0 mit Visual Studio 2005 gewählt. Diese Entscheidung wurde durch die zu Grunde liegenden technischen Konzepte, die Produktivität bei der Entwicklung neuer Komponenten und die Möglichkeiten zum Refactoring und der Ähnlichkeit im Look & Feel mit der bisherigen Win32-Anwendung begünstigt. Nach dem die Zielvorgabe klar war, stellt sich dem Portierungsteam als Nächstes die Frage, wie diese Aufgabe bewältigt werden kann.

### Portierungshilfen für SQLWindows

Unter dem Namen *IceTeaGroup* firmiert eine Gruppe von Programmierern, die sich bereits vor Jahren dem Thema *SQLWindows*-Portierung angenommen hat und unter dem Sammelnamen *Porting Project* (PPJ) ein Werkzeug mit der Bezeichnung *IcePorter* entwickelte, welches die vorhandene SQLWindows-Source auf der Basis eines Portierungs-Frameworks (PPJ/FW) in C# vollautomatisch übersetzt. Vereinfacht dargestellt, hält das PPJ/FW den vollständigen Sprachumfang von *SQLWindows* bereit. Eine Gegenüberstellung der Codezeilen von *SQLWindows* und C# zeigt am Beispiel einer einfachen Funktion den Nutzen eines solchen Frameworks für die Portierung (Listing 1).

Der deutsche Partner der *IceTeaGroup*, die Firma *Fecher* aus *Rodgau* (Hessen), erhielt den Auftrag, die Portierung in enger Zusammenarbeit mit der *nGroup*, dem Hersteller des ERP-Systems, durchzuführen. Eine Machbarkeitsanalyse versprach zu Beginn, dass man wie durch Zauberhand in Kürze ein „neues" Produkt haben werde. Wie jedes größere Projekt wurde auch die Portierung in Abschnitte unterteilt. Die folgende Beschreibung ist zwar projektbezogen, lässt sich aber auch auf andere Portierungsprojekte dieser Größenordnung anwenden.

Maßgeblich für alle Phasen ist eine transparente und strukturierte Vorgehensweise. Die erste Analyse wurde bereits mit einem Werkzeug durchgeführt. Der Quellcode wird mit dem *PPJ Inventory* analysiert. Das Ergebnis beinhaltet statistische Daten, die auch zur Berechnung der Portierungskosten herangezogen werden. Die Anzahl der *SalCompileAndEvaluate*-Funktionen steht damit fest wie auch alle eingebundenen externen DLLs und Ac-

tiveX-Objekte. Für die „Säuberung“ des Quellcodes wird eine Logdatei ausgewertet, die auf nicht vollständig qualifizierte Referenzen, auf Vertauschungen der Datentypen und andere im Sinne einer strengeren Sprache nicht korrekten Codestellen hinweist. Die Säuberung reduziert auch die Kosten, da der Preis der Portierung gemäß der Kalkulation der *IceTeaGroup* von der Anzahl der Codezeilen abhängt.

## Ein Framework standardisiert proprietäre Funktionen

Ein Projektziel war, den identischen Funktionsumfang und das identische Design in einer neuen „Welt“ zu erhalten. Die künftige Anwendung muss von den gleichen Mitarbeitern weiterentwickelt und gepflegt werden. Auch aus diesen Gründen wurde der Portierungsprozess mit einem stark strukturierten und automatisierten Verfahren durchgeführt und Entwickler mit ausgeprägten Kenntnissen beider Welten eingebunden. Durch die Verwendung eines Frameworks wird eine hohe Eindeutigkeit des Quellcodes im neuen Zielsystem erzielt und erreicht, sodass die Entwickler des Altsystems ohne wesentlichen Trainingsaufwand den neuen Quellcode weiterbearbeiten können. Klagen gab es von den „.NET-Puristen“, denn sie mussten sich zu einem gewissen Grad in die alte Sprache einarbeiten. Schwerer wiegt, dass der Code nicht automatisch objektorientiert in .NET abgebildet wird. *IcePorter* bietet deshalb eine Vielzahl von Optionen, die dafür sorgen, dass viele Konstrukte auch in der erwarteten objektorientierten Schreibweise generiert werden. Für das zuvor angeführte Beispiel steht im Code deshalb auch:

```
sName=sName.Left(6);
```

Eine umfangreiche Warenwirtschaft ist ein stark dialogorientiertes Programm mit unzähligen SQL-Zugriffen. Der Datenbankzugriff unter *SQLWindows* wird mit Inline-Bindevariablen programmiert. Der Befehl

```
Select name1, name2 into :sName1, :sName2 from
                                    kunde where…
```

selektiert *Name1* und *Name2* in die Variablen *sName1* und *sName2*. Das Porting-Framework verwendet aber kein proprietäres Datenbankprotokoll, um diesen Befehl an die Datenbank zu übermitteln. Die programmierten SQL-Zugriffe finden sich in der gleichen Syntax im Quellcode wieder. Das Porting-Framework leistet bei der Übernahme dieser Befehle die wesentliche Arbeit der Standardisierung, denn es setzt jeden SQL-Befehl in ADO .NET um. Die Verbindung zur Datenbank kann von außen konfiguriert werden. Für Anwendungen, die bisher lediglich die Datenbank *SqlBase* von *Gupta* verwendet haben, steht zudem ein Übersetzer zur Verfügung, der ohne weitere Eingriffe die Verbindung zum Microsoft SQL Server oder Oracle erlaubt.

Auch wenn C# insgesamt eine modernere Sprache ist, gerade die Möglichkeiten des eingebetteten SQL zeichnen eine 4GL-Sprache aus. Es gibt daher in einer *SQLWindows*-Anwendung zahlreiche Funktionen, die sehr komplexe Sachverhalte abbilden, ohne dass viel programmiert werden muss. Betrachtet man das Ergebnis der Funktion

```
Sal.TblPopulate( hWndGrid, hSqlHandle, 'SELECT *
                                FROM COMPANY', TBL_FillAll);
```

wird deutlich, dass das PPJ/FW nicht nur eine temporäre Bibliothek ist, um den Übergang nach .NET zu schaffen. Es ist vor allem ein Framework, mit dem in .NET 4GL-Funktionen zur Verfügung gestellt werden, die natürlich auch bei Weiterentwicklungen wertvolle Dienste leisten. Der obige Befehl füllt ein sehr stark erweitertes *VSGrid* mit allen Daten der Tabelle *Company*. Es wäre unsinnig diese Funktion mit einer nativen .NET-Implementierung zu ersetzen.

## Besonderheiten des Quellsystems müssen berücksichtigt werden

*SQLWindows* kennt keine strengen Datentypen. Es ist objektorientiert, kennt aber keine Überladung der Methoden sowie keine privaten und öffentlichen Variablen. Dafür erlaubt es eine Mehrfachvererbung und spät gebundene Methoden. Die Möglichkeit COM-Server mit dieser Sprache zu schreiben wird wenig genutzt. Die Anwendung konsumiert stattdessen häufig als COM-Client eine Reihe von Standard-COM- und ActiveX-Objekten, aber auch in C++ geschriebene COM-Server. Dies ist im Zusammenhang mit einem Portierungprojekt erwähnenswert, da diese COM-Server die Runtime von *Gupta* einbinden. Die Interpretersprache *SQLWindows* benötigt zur Laufzeit eine Runtime, die aus einer Vielzahl von DLLs besteht. Natürlich dürfen die COM-Server in der späteren .NET-Anwendung keine Referenz auf diese Runtime aufweisen. Die besonderen Herausforderungen bei der Portierung liegen vor allem in der „Nachlässigkeit“ des Compilers von *SQLWindows* begründet. Die geringe Strenge in den Datentypen führt dazu, dass *Boolean* mit *Number* und *Window Handle* mit *File-* und *Sql*-Handle jederzeit vertauscht werden können. Auch vollständig unqualifizierte Zugriffe auf Fenster, Funktionen, Variablen und Controls sind erlaubt und führen erst zu einem Laufzeitfehler, wenn die Objekte zu diesem Zeitpunkt nicht zur Verfügung stehen. Das Message-System erlaubt ein *PostMessage*, das eine Nachricht an das Ende der Warteschlange stellt, aber erst nachdem die gerade laufende Funktion beendet ist und dessen Funktionsweise leider nicht wirklich immer vorhersehbar ist. Eine der mächtigsten Funktionen ist *SalCompileAndEvaluate*. Mit dieser kann jeder beliebige Sal-Befehl zur Laufzeit ausgeführt werden.

Die Portierungstechnologie muss für diese technischen Möglichkeiten des Altsystems Lösungen zur Verfügung stellen. Analytisch müssen Referenzierungen und Datentypen untersucht werden. Änderungen können zum Teil vollständig automatisiert umgesetzt werden. In speziellen Fällen gibt es aber auch nur Lösungsvorschläge, die manuell geprüft und freigegeben werden müssen. Wie geht man aber mit der Herausforderung um, eine Funktion in .NET zu benötigen, die den vollständigen Sprachumfang des Altsystems beherrscht? Hier wird klar, dass traditionelle Portierungsansätze in der Form

**Listing 1**

**SQLWindows**

```
Set sName = ´ngroup eEvolution´
Call SalStrLeft(sName, 6, sname)
Call SalMessageBox(sName, ´Hinweis´, 0)
```

**C# .NET**

```
sName= “ngroup eEvolution”;
Sal.StrLeft(sName, 6, sName);
Sal.MessageBox(sName, “Hinweis“, 0);
```

eines semiautomatisierten „Neuschreibens“ versagen.

## Manuelle Nacharbeiten

Sofern Teile des Codes des Quellsystems nicht bereits unter einer Versionskontrolle standen, wurden gleichzeitig mit dem Projekt ein Versionsmanagement und ein Defect-Tracking-System eingeführt. Diese qualitätssichernden Maßnahmen mussten getroffen werden, um den Erfolg des Portierungsprojektes sicherzustellen. Der aufbereitete Code wurde dann der automatischen Konvertierung übergeben und so lange weiterbearbeitet, bis der Quellcode unter .NET fehlerfrei kompilierbar war. Die erste .NET-Version von *eEvolution* hatte damit das Licht der Welt erblickt. Nun reicht es leider nicht aus, seinen Kunden kompilierbaren Code zu liefern, er sollte auch funktionieren und die lieferfertige Anwendung sollte zusätzlich noch gut aussehen, am liebsten besser als vorher. Um das zu erreichen, wurden im Anschluss umfangreiche Schritte zur manuellen Nachbearbeitung der Funktionen initiiert, die von *IcePorter* nicht im ersten Durchgang umgesetzt werden konnten.

Die erste Version wurde zuerst auf Designfehler geprüft. Sollte *IcePorter* nicht korrekt gearbeitet haben, so wären „tausende Stellen“ nachzubearbeiten. Es kann also durchaus sein, dass eine Version sofort vernichtet wird. Die Portierungsspezialisten von *Fecher* gehen hier ganz pragmatisch vor: Enthält die portierte und kompilierbare Version von einer Fehlerklasse mehr als 20 Vorkommnisse, so wird nach der Ursache gesucht und eine Änderung bei *IcePorter* vorgenommen.

Der Portierungsspezialist kann eigene Plug-ins schreiben, die vor und nach dem Aufruf einer externen Funktion zusätzlichen Quellcode in die neue Anwendung einbauen. Verbesserungen bei *IcePorter* werden solange vorgenommen, bis der Aufwand, den automatischen Portierungsprozess weiter zu verbessern, nicht mehr in einem sinnvollen Verhältnis zur manuellen Nacharbeit steht. Erst nach dem letzten Durchgang werden die Restarbeiten manuell erledigt. Beispiele hierfür sind die Überprüfung der automatisch nachqualifizierten Zugriffe, Änderungen des Codes bei Rückgabewerten von Fenstern (die *SQLWindows* im Gegensatz zu .NET zulässt) und Reimplementierung von Funktionen, die von Drittanbietern in das SQL-Windows-Projekt eingebunden wurden, aber noch nicht im Porting-Framework abgebildet sind.

## Und nun zu den Kosten

Der Kostenanteil der manuellen Nacharbeit an den Kosten des Gesamtprojektes liegt durchschnittlich bei 25 Prozent. Bei überdurchschnittlicher Komplexität, die sich an der Anzahl der Module, eingebundener externer Funktionen und COM-Server, verwendeter *SQLWindows*-Bibliotheken von Drittanbietern und eingebauter Skripttechnologie bemisst, kann der Umfang aber auch deutlich höher liegen.

## Umfangreiche Testverfahren – auf Wunsch mit Toolunterstützung

Die Finalisierungsphase findet ihren Abschluss in der Übergabe des Projektes an das Testteam des Auftraggebers. Dort ist das notwendige Fachwissen vorhanden, um die Anwendung tiefgehend zu testen. Die Phase der Qualitätssicherung darf nicht unterschätzt werden, da alle Bereiche der Anwendung vollständig getestet werden müssen. Auch in dieser Phase

## Fünf Fragen an den Entwickler

**Wann und warum fiel die Entscheidung für das .NET Framework? Standen Alternativen zur Diskussion?**
Als ein Unternehmen, das Microsoft nahe steht, war die *nGroup* aus anderen Kundenprojekten mit dem .NET Framework bereits vertraut. Das Konzept überzeugt, die Integration in Visual Studio ist wichtig. Ziel der Portierung war vorrangig der Wechsel auf eine zeitgemäße IT-Plattform. In Frage kommen hier nur Java oder .NET. Die Erwartungshaltungen an eine Java-Anwendung sind bezüglich der mehrschichtigen Architektur erheblich größer als bei einer Portierung nach .NET. Eine SQLWindows-Client-Server-Anwendung ist zweischichtig aufgebaut (die Oberfläche steht im Mittelpunkt). 4GL-Funktionen führen in einer Zeile komplexe Anweisungen durch. Eine Portierung dieser komplexen Anweisungen mit gleichzeitiger Änderung der Architektur ist nur sehr schwer möglich. Daher erschien .NET als die einzig tragbare Lösung, mit der ein schneller Projekterfolg garantiert werden konnte.

**Wie hat das .NET Framework die Entwicklung positiv beeinflusst? Was waren die größten Hindernisse, die überwunden werden mussten?**
Die Entwicklung des Porting-Projekts war nur auf der Basis der Fähigkeiten von .NET und der großen Flexibilität von C# möglich. Es wäre unmöglich einen ähnlich hohen Grad der Kompatibilität zwischen der alten und neuen Welt mit Java zu erreichen. Die Überladung von Operatoren, Rückgabe-Parameter und Delegates waren die wichtigsten Kernfunktionen von C#, die benötigt wurden, um das Porting-Framework zu schreiben. WinForms erleichterte die Entwicklung von visuellen Controls für das PPJ-Framework. Die größten Hindernisse waren die Nachbildung von Mehrfachvererbungen durch Benutzung von Delegates und die Überladung von Operatoren.

**Wurden bei der Umsetzung Schwachstellen im .NET Framework deutlich oder gibt es Dinge, die Microsoft hätte anders oder besser lösen können?**
Es gibt im Wesentlichen zwei Aspekte, die unserer Ansicht nach durch Microsoft besser gelöst werden können: 1. ADO.NET hat keine allgemeine Basisklasse für Exceptions. Dies hat dazu geführt, dass wir im Quellcode bekannte Exceptions programmieren und den Rest über Reflection lösen mussten. 2. Das Design von „Kurzschluß-Überladungen“ im C#-Compiler ist konzeptionell falsch (Abschnitt 7.11.2. der C#-Spezifikation). Es vermischt unnötigerweise die Operatoren *op_True* und *op_False* mit den bitweisen Operatoren. Das macht es unmöglich, in einer Klasse einen bitweisen Operator und einen logischen Operator zu implementieren. Dieses Designproblem wird während der Übersetzungsphase gelöst, indem ein expliziter Cast zu *boolean* erzeugt wird.
Es gab darüber hinaus Probleme bei der Integration von Oracle-Datenbanken bzw. des Datentyps *NUMBER* und es gibt Probleme mit der Performance des WinForm-Editors bei komplexen Formularen mit sehr vielen Elementen. Hier kam es zu Abstürzen und langen Laufzeiten.

**Wieso erfolgt die Entscheidung für C# und welche Eigenschaften der Sprache sprachen für die Entscheidung?**
In anderen Projekten wurde bereits C# eingesetzt und insgesamt bietet C# mehr Möglichkeiten als Visual Basic. Die notwendige Abbildung des gesamten Sprachumfangs von *SQLWindows* im .NET Framework und die Transformation des projektspezifischen Codes nach .NET konnten zu Zeiten des .NET Framework 1.1 nicht erfolgen. Mittlerweile wird die Portierungstechnologie auch für Visual Basic zur Verfügung gestellt.

**Wie sieht die Zukunft des Projekts aus? Gibt es bereits Überlegungen künftige Versionen des.NET Framework und wenn ja, welche Features ganz speziell, zu nutzen?**
Ein wichtiger Punkt für die Zukunft ist die Überführung von Teilen der Applikation in eine Web-Applikation und die Nutzung des .NET Framework bzw. von Visual Studio bei der Unterstützung einer solchen Migration.

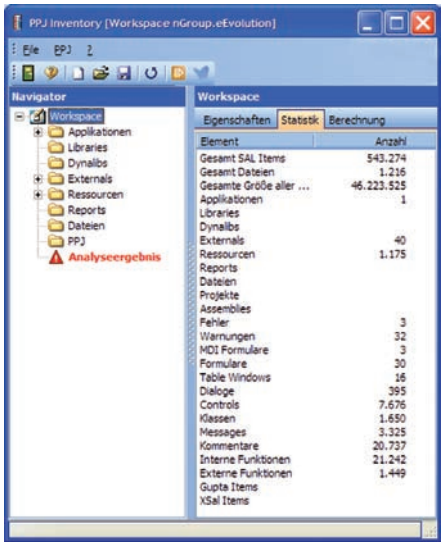


Abb. 1.: Automatisierte Portierung von SQLWindows auf .NET mit Toolunterstützung

stehen inzwischen automatisierte Werkzeuge zur Verfügung – Portierungspartner *Fecher* kann auf Wunsch die Kernbereiche der *SQLWindows*-Anwendung mit dem Testwerkzeug *Tosca Commander* als Teil der *Tosca*-Testsuite automatisieren. Dieses Werkzeug schreibt keine Skripte, sondern legt die technischen Informationen in sog. XML-GUI-Maps ab. Da im Portierungsprozess eine eindeutige Transformation der Anwendung hinsichtlich Design und Funktionalität stattfindet, reicht es aus, die XML-GUI-Maps ebenfalls zu portieren. *Tosca Commander* stellt Adaptoren für *SQLWindows* und .NET zur Verfügung, sodass nach einer Portierung sofort alle automatisierten Testfälle gestartet werden können. Ein schneller Überblick über die Qualität der neuen Software ist so sichergestellt.

## Kundenspezifische Anpassungen ins Zielsystem übernehmen

Eine betriebswirtschaftliche Standard-Software wird im Allgemeinen an kundenspezifische Besonderheiten angepasst. Diese Anpassungen müssen auch im Zielsystem weiterhin funktionsfähig sein, um eine hohe Akzeptanz und geringen Update-Aufwand sicherzustellen. Kein Kunde möchte gerne für bereits bezahlte Anpassungen im Rahmen eines Updates noch einmal bezahlen. Parallel zur Konvertierung des Codes musste daher ein Weg gefunden werden, kundenspezifische Berichte zu erhalten und ebenfalls zu konvertieren. Die im Quellsystem verwendete Reporting Engine ist proprietär und benutzt keine eigene DB-Schnittstelle. Die zu druckenden Berichte werden in Form von Templates definiert und beziehen ihre Daten über eine API aus der Anwendung. Die zu Grunde liegenden Abfragen stehen entweder in der Datenbank, wenn sie vom Kunden geändert werden können, oder sind direkt in der Anwendung hinterlegt. Da die bei Visual Studio 2005 enthaltene keine .NET-Anwendung ist, wurde eine Alternative gesucht. Die Entscheidung fiel auf *List&Label*, das eine ähnliche Philosophie bei der Datenübergabe besitzt wie *SQLWindows* existiert, und weil die Lizenzpolitik der Herstellerfirma *Combit* zudem sehr angenehm ist.

Die letzte große Hürde stellte der sog. *Customizer* dar. Unter *SQLWindows* ist es nicht ohne Weiteres möglich, eine komplexe Anwendung während der Laufzeit vernünftig und kundengerecht anzupassen. Optische Veränderung der Bildschirmansichten und das Hinzufügen neuer Objekte und Code sollte releasefähig bleiben und nicht bei jedem Update der Software neu programmiert werden müssen. In *SQLWindows* gibt es zwar den leistungsfähigen Befehl *SALCompileAndEvaluate*, der es erlaubt, externen Code zur Laufzeit im Interpreter auszuführen. Aber diese Möglichkeiten reichten nicht aus. Bei *eEvolution* sind im Laufe der Jahre und mit wachsendem Wissensstand der Entwicklergemeinde viele Lösungen geschaffen worden, die dieses Defizit behoben. Leider waren sämtliche Lösungen unter .NET nicht einsatzfähig. Daher entstand ein völlig neuer Customizer unter .NET, der alle Anforderungen erfüllt, die man sich schon seit Jahren gewünscht hatte. Benutzerspezifische Änderungen der Bildschirme inklusive Code in *SQLWindows*- oder C#-Syntax mit Zugriff auf alle Programmfunktionen, Klassen und deren Methoden können damit durchgeführt werden. Hier zeigte sich auch eine der größten Stärken der neuen .NET-Technologie gegenüber dem Vorgänger: Eine große Entwicklergemeinde bietet eine noch größere Auswahl an fertigen Tools, die man vergleichsweise günstig inklusive Sourcecode erwerben kann, ohne jedes Rad neu erfinden zu müssen. Für den Customizer wurde ein .NET-Control von *Greatis* verwendet [3], welches die Basisfunktionalität liefert.

## Fazit

Nach insgesamt neun Monaten wurde die Konvertierung erfolgreich abgeschlossen und es wurden letzte Feinheiten angepasst. Unter anderem wurde ein neuer Installer benötigt und die Sourcecode-Verwaltung musste neu eingerichtet werden. Hier zeigten sich abermals die Stärken von Visual Studio 2005. Die Unterstützung der Entwicklungsumgebung grenzt bei diesen Themen beinahe an ein Wunder. In der Ausgangsumgebung musste man derartige Anforderungen aufwändig von Hand lösen. Rückblickend lässt sich feststellen, dass der Aufwand an einigen Stellen höher war als zunächst erwartet. Alle Beteiligten würden das gleiche Vorgehensmodell aber erneut wählen. Diese Vereinfachungen und der Produktivitätsgewinn bei der Software-Entwicklung ermöglichen ein sinnvolles und effektives Refactoring an den notwendigen Stellen. Dieses Arbeiten werden neben der Weiterentwicklung der Lösung die nächsten Großprojekte der *nGroup* sein.

**Frank Wuttke** ist Geschäftsführer der nGroup und verantwortlich für das Portierungsprojekt auf .NET. Sie erreichen den Autor per E-Mail unter wuttke@ngroup.info

### Links & Literatur

[1] www.ngroup.info
[2] www.fecher.de
[3] www.greatis.com/dotnet/
[4] www.iceteagroup.com