DAS MAGAZIN FÜR PROFESSIONELLE IT-LÖSUNGEN

Der Entwickler

Entwickler

4 194156 605502 01

Ausgabe 1.2004
Januar/Februar

D31141F

Deutschland € 5,50
Österreich € 6,00
Schweiz SFr 10,80

S&S

mit CD

Model Driven Architecture

Vorschau auf Delphi 8 for .NET

Übersicht, Praxis und Tools

Spiele-Programmierung

Einsatz von VB.NET und Direct3D

www.derentwickler.de

Datenbanken

Hochverfügbarkeit von Datenbanken

Unter die Lupe genommen:
Microsoft SQL Server 2000 und MySQL 4.1

Delphi

ISDN-Anwendungen mit Delphi erstellen

PDF-Generierung mit Delphi

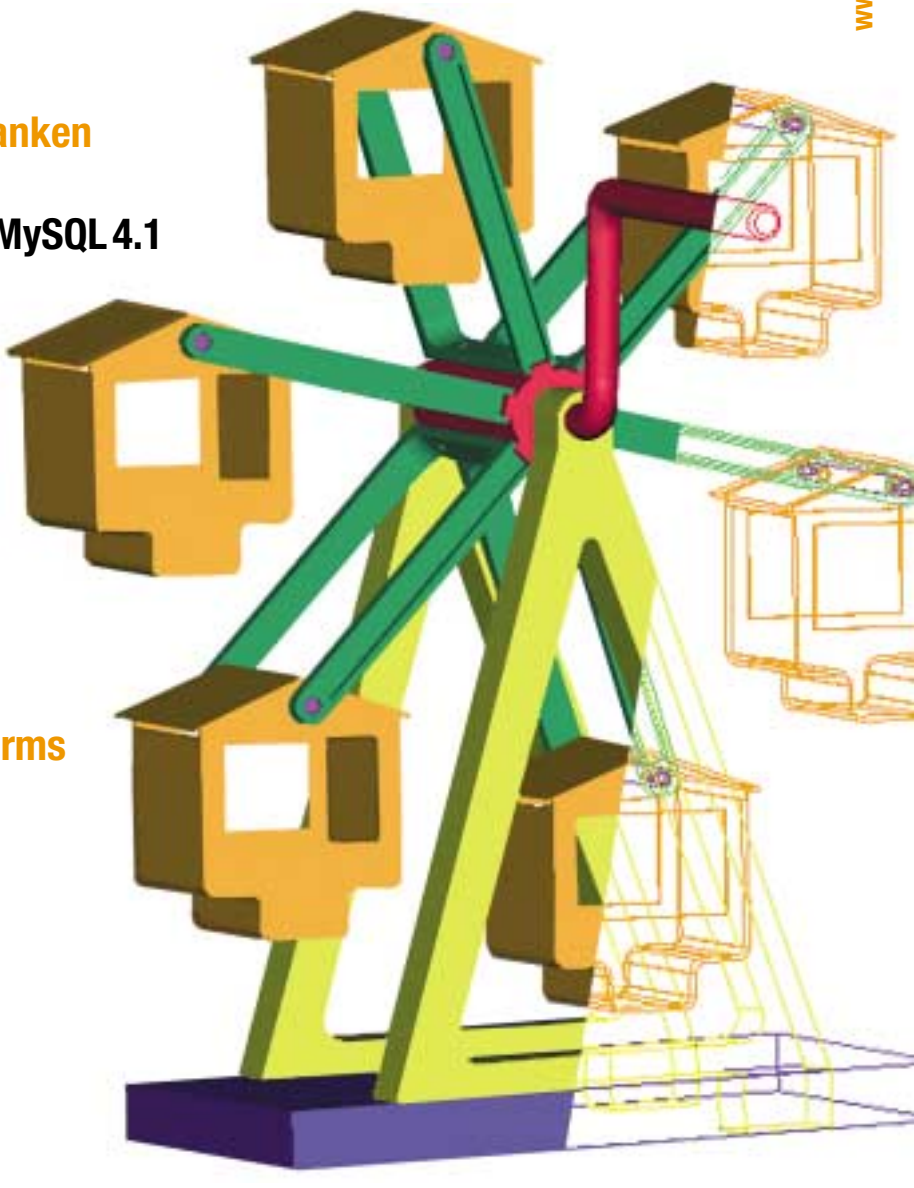
C++ & C#

Streifzug durch die Windows Forms

Die Borland Data Providers for ADO.NET vorgestellt

Special

Webdesign:
Methoden und Technologien



- MDA Tools: objectiF Visual Studio .NET Edition 5.0
- Poseidon for UML Professional 2.1.1
- XCoder 1.0.6
- Testversionen
- Datenbanken & Tools

Infos auf Seite 5

Anzeige

Anzeige

Anzeige

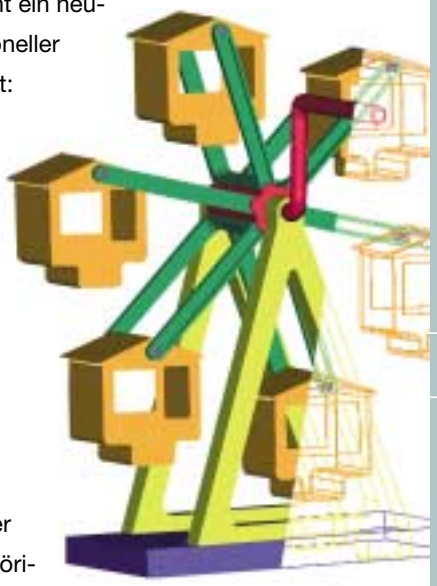
Anzeige

Anzeige

Titelthema

Model Driven Architecture

 Die Entwicklergemeinde kennt ein neues Zauberwort, das in traditioneller Form der drei Buchstaben daherkommt: MDA (Model Driven Architecture). Wird man nun weniger programmieren und vermehrt modellieren oder induziert die Toolindustrie wieder neue Bedürfnisse ? Beides, denn eine stärkere Formalisierung ist sowohl der Wunsch von uns geplagten Entwicklern wie auch die Bedingung der Industrie, Tools und Architekturen zu vereinheitlichen. In diesem Artikel möchten wir Ihnen einen Überblick über die MDA, ihrem Einsatz und den zugehörigen Tools geben.

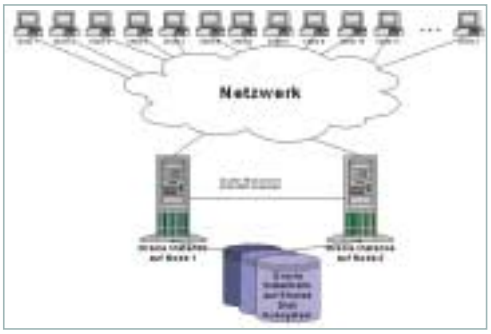


33: Model Driven Development mit Bold für Delphi

Datenbanken

Hochverfügbarkeit von Datenbanken

 Nicht nur für Aktienhandelssysteme und Online Shopping-Unternehmen gewinnt das Thema Hochverfügbarkeit zunehmend an Gewicht. Ausfallzeiten eines unternehmenskritischen Systems verursachen unter Umständen horrende Kosten. Doch die Realisierung von Ausfallsicherheit ist technisch komplex und oft ein teurer Spaß. Wir greifen einige Konzepte auf und betrachten Vor- und Nachteile der jeweiligen Lösung.



Datenbanken

MySQL wird erwachsen

 Im Gegensatz zu anderen DBMS folgt MySQL AB konsequent seiner Unternehmensphilosophie und versucht Features und Innovationen eher behutsam einzugliedern. Mit der Version 4.1 ist MySQL nicht nur auf dem richtigen Weg angelangt, sondern wird zunehmend ein ernst zu nehmender Konkurrent für kommerzielle Datenbankhersteller. Zudem scheint das Ziel von MySQL klar anvisiert zu sein – der Sprung in die Enterprise-Liga soll mit den kommenden Versionsreihen 4.1 und 5.0 gelingen.

Tools & Reviews

Titelthemen

Datenbanken

Delphi

Tipps & Tricks

C++ & C#

Special

Service

Delphi

ISDN-Anwendungen mit Delphi erstellen

63 In der letzten Zeit ist, im wahrsten Sinne des Wortes, immer mehr von Audiotex zu hören. Televoting, Logo- oder Klingeltonversand oder auch nur einfache Auskunftsdienste haben eines gemeinsam, der gegenüberliegende Teilnehmer ist ein „einfacher“ Computer. Genauso „einfach“ können Sie nach dem durchlesen des Artikels eine Audiotex-Plattform in Betrieb nehmen. Wir zeigen Ihnen, wie Sie ISDN-Anwendungen mit Delphi erstellen können.

C++ & C#

C#Builder und die Borland Data Provider for ADO.NET

85 Der Borland C#Builder wurde vor einigen Monaten in den vier Versionen Personal, Professional, Enterprise und Architect veröffentlicht. Bis auf die Personal Edition erhalten diese alle die sogenannten Borland Data Provider for ADO.NET. In diesem Artikel möchten wir Ihnen zeigen, was genau man mit den BDP for .NET und C#Builder machen kann.

Special

Spiele-Programmierung mit VB.NET und Direct3D

88 Dreidimensionale Körper oder gar ganze 3D-Welten darzustellen gehört zur „Königsdisziplin“ in Sachen Grafikausgabe. Doch längst gehört dieses Gebiet zum festen Bestandteil vieler Anwendungen, vor allem im Hinblick auf Spiele, CAD- oder Raytracing-Programme. Somit wachsen natürlich auch ständig die Anforderungen an die Programmierer solcher Anwendungen. Damit diese jedoch nicht im Regen stehen, veröffentlicht Microsoft eine ständig erweiterte Version von Direct3D, einem Bestandteil der Multimediaschnittstelle DirectX. Anhand von Beispielen möchten wir Ihnen zeigen, wie Sie Direct3D für sich nutzen können.

microTool gibt microTool Suite .NET frei

Das Berliner Softwarehaus microTool hat die UML-basierte Software microTool Suite .NET freigegeben. Die enthaltenen Tools und Prozesse sollen .NET-Entwickler, die mit C# oder VB.NET arbeiten, bei der Umsetzung sogenannter agiler Softwareentwicklungsmethoden unterstützen und sich eng mit der Microsoft-IDE Visual Studio .NET integrieren lassen. Das Produkt soll auf kleinere Projekte zugeschnitten sein und neben Modellierung Funktionalitäten wie Anforderungsmanagement und Konfigurationsmanagement bieten. Zu den Komponenten der Suite gehören unter anderem die Tools in-Step, objectiF, actiF, ein Prozess für agiles Projektmanagementment, sowie das Open Source-Tool NUnit für die Durchführung von Unit Tests. Information zu den verfügbaren Lizenzmodellen finden sich auf der Produktseiten.

www.microtool.de/suite/de/

W4 Toolkit: Web GUI-Builder für Eclipse

Ein Werkzeug für die einfache Entwicklung Java-basierter Web-Frontends stellt die Karlsruher Innoopract mit ihrem W4 Toolkit bereit. Es basiert auf Eclipse und bietet eine Reihe grafischer Komponenten, die per Drag´n Drop positioniert, eine einfache Entwicklung von Weboberflächen für J2EE-Anwendungen erlaubt. Oberflächen können dabei nach einem Baukastenprinzip aufgebaut werden, bestehende Komponenten lassen sich erweitern, ergänzen und wiederverwenden, heißt es in einer Ankündigung. Zukünftig soll das W4 Toolkit auch für die Entwicklung von Swing- und SWT (Standard Widget Toolkit) -Anwendungs-Funktionen bereit stehen. Das W4 Toolkit steht als kostenloser Download zur Verfügung. Innoopract ist seit Mai 2003 Mitglied des Eclipse-Konsortiums und versteht sich als Spezialist für die objektorientierte und visuelle Entwicklung von Programmoberflächen auf Basis der Open Source-IDE.

www2.w4toolkit.de/
www.eclipse.org/

Borlands C++BuilderX veröffentlicht

Borland bingt C++BuilderX, eine Plattform- und Compiler-unabhängige C++-Entwicklungsumgebung, auf den Markt. C++BuilderX läuft auf Windows, Linux und Solaris und unterstützt Unix auf der Basis einer integrierten Entwicklungsumgebung (IDE) für VisiBroker CORBA. Laut Angaben orientiert sich die IDE von C++BuilderX am Borland JBuilder. Damit soll die Cross-Plattform-Programmierung für C++-Entwickler erleichtert werden. Neben den Borland C++-Compilern unterstützt der C++BuilderX auch eine Reihe weiterer Standard-basierter C++-Compiler für Umgebungen wie Microsoft, Sun, Forte C++ sowie GNU Compiler Connection und Plattformen wie ARM und Intel. Des Weiteren bietet C++BuilderX Features und Funktionen zur Entwicklung von Applikationen für mobile und Embedded Systeme. Der Borland C++BuilderX ist ab sofort in einer Enterprise und Developer Edition verfügbar.

Zeitgleich mit dem C++BuilderX bingt Borland Enterprise Studio for Mobile auf den Markt. Borlands Enterprise Studio for Mobile designed, testet und entwickelt mobile Java- und C++-Lösungen. Enterprise Studio for Mobile soll alle Symbian Software Development Kits (SDK), einschließlich der Serien 60 und 80, unterstützen. Außerdem soll mithilfe des C++BuilderX, als integrierter Bestandteil des Enterprise Studio for Mobile, das Rapid Application Development (RAD) auch der mobilen Welt erschlossen werden. Per Drag`n Drop sollen sich bereits vorhandene Software-Komponenten aus internen oder externen Bibliotheken in neue Anwendungen einbauen lassen.

www.borland.com/cbuilderx/
www.borland.com/mobile/enterprise/

Advantage Database Server 7.0 veröffentlicht

Extended Systems bringt die Version 7.0 des Advantage Database Server auf den Markt. Advantage Database Server ist ein Relationales Datenbank-Management-System (RDBMS) für den Zugriff auf unternehmenskritische Daten, innerhalb eines bestehenden Netzwerks, Internet-basierend oder über mobile Datenbank-Anwendungen. Neu in der Version 7.0 sind: Trigger, Datenkompression für einen reduzierten Datenverkehr, Volltextsuche, .NET Daten-Provider sowie einen type-4 JDBC Treiber. Unterstützt werden die Betriebsysteme NetWare, Linux, Windows NT/2000/XP/2003 und Windows 98/ME. Advantage Databse Server 7.0 ist ab sofort erhältlich. Eine 30-Tage Testversion steht zum Downloaden bereit.

www.advantagedatabase.com/ads/

MDA-Tool XCoder als Open Source freigegeben

Die Liantis GmbH hat das MDA-Tool XCoder als Open Source freigegeben. Mit diesem Schritt möchte Liantis, laut Angaben, den XCoder einer größeren Entwicklergemeinde zur Verfügung stellen. XCoder ist ein Extensible Model-Transformations- und Code-Generierungs-Framework, das den Model Driven Architecture-Standard der Object Management Group ausführt. Anpassbare Standardlösungen für Java, C++, C#, J2EE und .NET liegen bereits in der Standarddistribution vor. XCoder ist komplett mit MDA erstellt, in Java geschrieben und wird auch mit MDA erweitert. Alle dem XCoder zugrunde liegenden UML-Modelle sind ebenfalls Bestandteil der Standarddistribution.

www.liantis.com/Leistungen/
sourceforge.net/projects/xcoder

SQL Tools für Datenbank-Prüfung veröffentlicht

LockwoosTech Software veröffentlicht Log Navigator 1.0. Log Navigator ist ein SQL Server Prüfungs-Tool, mit dem SQL Server Transaktions-Logs analysiert werden sollen, um Informationen über die Daten und strukturelle Änderungen zu zeigen. Log Navigator soll ohne Performance-Zuschlag und Benutzung von Triggern die Daten lesen. Neu in der Version 1.0 ist ein Live-Log-Support sowie XML- und Datenbank-Export. Eine Testversion steht zum Downloaden bereit.

www.lockwoodtech.com/index_lognavigator.htm

OpenOffice.org in der Version 1.1 veröffentlicht

Die Mitglieder des Projektes OpenOffice.org geben die Version 1.1 der quelloffenen Office-Suite für Windows, Linux und Solaris frei. Die kostenlose Office-Suite steht ab sofort in englischer und deutscher Sprache zum Downloaden bereit. OpenOffice enthält sämtliche Office-Funktionalitäten wie Textverarbeitung, Tabellenkalkulation, Präsentationssoftware, HTML-Editor, Formeleditor und Zeichenprogramm sowie eine integrierte Datenbankanbindung. Zu den wichtigsten Neuerungen von OpenOffice.org 1.1 zählen unter

anderem PDF-, Flash-, XHTML- und Text-XML-Export sowie Unterstützung mobiler Formate wie AportisDoc (Palm), Pocket Word und Pocket Excel.
www.openoffice.org/

MS SQL Utils: Werkzeuge für den SQL Server

EMS HiTech Company hat MS SQL Utils in der Version 1.1 auf den Markt gebracht. Im MS SQL Utils-Paket sind Daten-Management-Werkzeuge für den Microsoft SQL Server und der Microsoft SQL Server Desktop Engine (MSDE) enthalten. MS SQL Utils beinhaltet MS SQL Query, MS SQL DataPump, MS SQL Data Generator, MS SQL Export und MS SQL Import. Neu in der Version 1.1 sind unter anderem: BLOB- und Unicode-Unterstützung, ein BLOB-Editor und einige Neuerungen im MS SQL Query-Tool. Kostenlose 15-Tage Testversionen der einzelnen Komponenten von MS SQL Utils stehen zum Downloaden bereit.
www.ems-hitech.com/mssqlutils/

Vereinbarung zwischen IDS Scheer und Opitz Consulting

Opitz Consulting, Systemintegrator für Oracle-Lösungen, erweitert seine Zusammenarbeit, im Bereich Business Process Engineering, um einen Reselling-Vertrag mit dem Software- und Beratungshaus IDS Scheer. Opitz Consulting verwendet die ARIS-Produkte seit mehreren Jahren als Beratungsmethodik für die Geschäftsprozessoptimierung im Oracle-Umfeld. Künftig wird Opitz Consulting die ARIS Process Platform verstärkt zu Zwecken der Individualsoftware-Entwicklung und des Software-Engineering einsetzen und als Komplettanbieter im Verbund mit der entsprechenden Dienstleistung anbieten. Beispiele für den Einsatz von ARIS im Bereich Software-Engineering wurden von Opitz Consulting im Rahmen der „ARIS P2A Roadshow“ Anfang November 2003 in Berlin und Düsseldorf vorgestellt.
www.opitzconsulting.de/
www.ids-scheer.de/

Microsoft gibt Tools für SQL Server 2000 und „Yukon“ bekannt

Auf der 2003 Professional Association for SQL Server (PASS) Community Summit in Seattle gab Microsoft Details über die Tools, die die Datenbank-Administrations-Produktivität von SQL Server 2000 und „Yukon“ steigern soll, bekannt. In der Eröffnungsrede verkündete Gordon Mangione, Corporate Vice President for SQL Server, die Weiterentwicklungen an der „Extract, Transform und Load“- (ETL-)-Technologie für die nächste Version des SQL Servers „Yukon“, die bekannt ist als Data Transformation Services (DTS). DTS soll Datenbank-Entwicklern ermöglichen, Datensätze und ETL-Lösungen zu bauen. Außerdem verkündete er die Verfügbarkeit des Best Practices Analyzers (BPA) für den SQL Server 2000. BPA ist ein Tool, das Datenbank-Administratoren bei der Entwicklung von SQL Server Applikationen unterstützen soll.
www.sqlpass.org/
www.microsoft.com/sql/

IBM Datenbank-Software für mobile Mitarbeiter

IBM kündigt die Version 8 des relationalen Datenbank-Servers DB2 Everyplace an. Mit der neuen Version sollen mobile Mitarbeiter über drahtlose Geräte wie Smartphones, PDAs und Laptops in Echtzeit auf Geschäftsinformationen zugreifen können. DB2 Everyplace 8 setzt IBM WebSphere Everyplace Access und IBM DB2 Information Intregrator ein und unterstützt PalmOS, Microsoft Windows CE/Pocket PC, Symbian, Linux, QNX Neutrino, Microsoft Windows NT/2000/XP und Linux.
www3.ibm.com/software/data/db2/everyplace/

Im Wandel der Zeit

Ein Rückblick auf die 7. Entwickler Konferenz

In der Fernsehwerbung wird ja manchmal suggeriert, dass drei Wünsche auf einmal nicht erfüllt werden können. Für die EKON 7 traf diese Einschränkung jedoch nicht zu, denn dort gab es tagsüber Know-how, am Abend Entspannung und zwischendurch auch Seelsorge. Dieser wohldosierte Mix führt dazu, dass ich jedes Mal viele vertraute Gesichter in der Eröffnungs-Keynote sehe.

von Andreas Kosch

Die Sessions während der Konferenz und die ganztägigen Power Workshops im Vor- und Nachprogramm bilden das eigentliche Fundament einer jeden Entwickler Konferenz. Denn die rund 450 Teilnehmer sind ja hautsächlich deshalb angereist, um sich auf den neuesten Stand zu bringen. Und hier zeigt sich bereits der Wandel, denn seit der EKON 6 gab es beim Themen-Schwerpunkt aller sieben Entwicklerkonferenzen – Sie werden ahnen, dass es hier nur um Delphi handeln kann – keine neue Version des bisherigen Flaggschiffs. Falls Sie nun vermuten, dass es daher in diesem Jahr für die Teilnehmer weniger stressig gewesen ist, befinden Sie sich aber auf dem Holzweg. Ganz im Gegenteil, denn gerade die Delphianer wurden in diesem Jahr unter einem derart hohen Wissensberg begraben, dass zeitweise seelsorgerische Unterstützung (ich komme am Ende darauf zurück) angesagt war. Von den insgesamt 80 Session-Themen befassten sich nur noch 34 mit dem klassischen Delphi, das ich zur besseren Trennung als Delphi (Win32) bezeichne. Selbstverständlich waren die klassischen Themen auch in diesem Jahr präsent, wie die folgenden Stichpunkte zeigen:

- Datenbanken: TClientDataset, dbExpress, Oracle, TurboDB, Advantage Database Server, Rave Reports.
- Internet: WebSnap, IntraWeb, Web Services, Indy10.
- Komponentenentwicklung.

An die Delphianer richteten sich dazu weitere 16 Sessions, die sich größtenteils mit der Zukunft von Delphi – also dem .NET Framework – befassten. Allein die Aufzählung der Themengebiete ADO.NET, ASP.NET, VCL.NET, BDP.NET, .NET Enterprise Services, .NET Framework Security und Borland Janeva (Anbindung von .NET an J2EE) macht deutlich, an wie vielen Stellen Neuland betreten werden muss. Damit aber nicht genug, wenn es nach Borland geht, sollen wir uns auch noch an die Schlagworte ALM und UML gewöhnen.

ALM und UML

Als Appetit-Anreger auf die entsprechenden Sessions hat David I. auf der Eröffnungs-Keynote Borlands Strategie des gesamten Application Lifecycle Managements (ALM) vorgestellt. Unter diesem Schlagwort verbirgt sich eine Reihe von Werkzeugen, die weit über die Aufgaben klassischer Entwicklungsumgebungen hinaus reichen und die sämtliche Prozesse vom Requirement Management, über das Configuration Management bis hin zur Optimierung laufender Anwendungen umfassen. In den Gesprächen räumten die Mitarbeiter am Borland-Stand ein, dass dies ein erneuter Versuch ist, einen Fuß ins Enterprise-Geschäft zu bekommen. Sowohl ALM als auch die Software-Modellierung (UML) richten sich an Projektgruppen, die typischerweise aus 10 oder mehr Entwicklern bestehen, die sich jeweils auf ein Teilgebiet des Entwicklungsprozesses spezialisiert haben.

Borland hat auf seiner Community-Webseite im Mai dieses Jahres eine Kunden-Umfrage gestartet (BDN 2003 Survey May 12, 2003) und die Ergebnisse an jedem, der sich dort mit einer eMail-Adresse beteiligt hat, zurückgeschickt. In diesem Papier fallen im Besonderen drei Punkte massiv auf:

- 40 Prozent der Borland-Kunden sind als Software-Entwickler Einzelkämpfer.
- 38 Prozent der Borland-Kunden gehören zu einem Team von 2 bis 5 Entwicklern.
- Nur 7 Prozent der Borland-Kunden gehören einem Team an, dass aus mehr als 10 Entwicklern besteht.

Diese Zahlen bestätigen auch meine Erfahrungen aus den Konferenzen und dem Entwickler-FORUM, denn sehr viele Entwickler arbeiten nur mit der Professional-Version von Delphi. Für ALM (UML) reicht aber sogar die Enterprise-Version nicht aus, hier muss zum einen die Architect Version beschafft werden, wobei zusätzliche Lizenzen für die AML-Softwarebestandteile fällig werden.

Trotzdem lohnt es sich, diese Entwicklung im Auge zu behalten. Denn sobald sich diese Tools tatsächlich völlig transparent in den Entwicklungsprozess einblenden können und auch der zusätzliche Lernaufwand niedriger wird, kann ALM als Nachbrenner die Entwicklungszeit verkürzen und somit zu einem wesentlichen Wettbewerbsvorteil werden. Dies ist vor allem auch deshalb wichtig, weil sich Borland mit .NET direkt gegenüber Microsoft mit seinem Visual Studio .NET positionieren muss.

Codename „Octane“

Nachdem Borland auf der dot.net Konferenz 2003 als Weltpremiere den C#Builder – damals noch unter dem Projektnamen „Sidewinder“ – der Öffentlichkeit vorgestellt hat, wiederholte sich das Ganze mit der Premiere von „Octane“. Hinter diesem Projekt verbirgt sich die Abspaltung von Delphi für die .NET-Plattform, wobei die Entwicklungsumgebung den gleichen IDE-Kern (Codename „Galileo“) nutzt wie der Borland C#Builder. Anders Ohlsson (Borland) zeigte in einer Technical Keynote eine einfache FCL-Applikation (.NET Framework Class Library), die nur die sprachunabhängigen Standardkomponenten aus dem .NET Framework nutzt. In einem zweiten Schritt erstellte er in Delphi 7 eine einfache Datenbank-Applikation, die über die VCL-Komponenten auf die BDE (Borland Database Engine) zugreift. Schließlich integrierte er deren VCL-Formular ins „Octane“-Projekt, um zu demonstrieren, dass eine sanfte Migration älterer Delphi-Projekte in die Delphi for .NET-Welt möglich ist (wobei es im Detail auf die dort verwendeten VCL-Komponenten ankommt).

Spannend bleibt die Frage, welchen Namen der neue Sprössling erhält. Borland hat angekündigt, dass es Delphi 8 erst im nächsten Jahr geben wird. Aber „Octane“ wird noch in diesem Jahr ausgeliefert und da es juristische Einwände gegen die Bezeichnung „Delphi.NET“ gibt, wird ein neuer Name benötigt. Sowohl der JBuilder als auch der C#Builder haben die Zeichenkette „Builder“ in der Produktbezeichnung, sodass ein Kompromiss zwischen Alt und Neu etwa so lauten könnte: „Turbo Delphi Builder Studio für das Microsoft .NET Framework“. Wenn Sie dies lesen, wissen Sie bereits, wie weit ich mit meiner Vermutung danebenlag. Mit „Octane“ verschiebt sich auch die Bedeutung von Kylix und der Visual CLX noch mehr zu ungunsten von Kylix. Mit C++BuilderX hat Borland außerdem eine vollständig neue Version seiner C++-Entwicklungsumgebung vorgestellt, die unterschiedliche Compiler einbindet und somit Anwendungen für die unterschiedlichsten Betriebssysteme wie Windows, Linux, Solaris oder Symbian generieren kann.

Entspannung

Wenn es tagsüber so stressig zugeht, wird für den Abend ein entspannender Ausgleich benötigt. Die Stichworte „Entwickler-Kino“ und „Freibier“ kannten wir bereits von den vorangegangenen Konferenzen. Bei der basisdemokratischen Abstimmung über die vier zur Auswahl gestellten Filme hatten die namhaften Action-Streifen keine Change – statt dessen entschied sich die überwältigende Mehrheit dafür, es als Ausgleich etwas ruhiger anzugehen und nur die Lachmuskeln zu trainieren. Der Film „My Big Fat Greek Wedding“ gab dazu ausreichend Gelegenheit.

Neuland betrat der Software & Support Verlag bei der Konferenz-Party, denn zusätzlich zum obligatorischen Freibier gab es den Abend über auch noch Darbietungen von brasilianische Tänzerinnen. Vermutlich war dieses Jahr das Bier besonders hochprozentig, denn im Laufe des Abends gerieten einige Teilnehmer derart in Wallung, dass es zu Temperamentsausbrüchen kam, die man einem Software-Entwickler niemals zugetraut hätte. Ich hoffe doch, das Masoud Kamali die Termine auch für die nächsten Konferenzen schon einmal gesichert hat.

Seelsorge

Ein Resümee zur Entwickler Konferenz wäre unvollständig, wenn dort nur die Sessions, Workshops, Keynotes und die angegliederte Ausstellung der Tools- und Lösungsanbieter genannt werden. Denn die persönliche Anwesenheit ermöglicht etwas, was ein Buch, eine Fachzeitschrift oder sogar eine modernere Bühne wie das Entwickler-FORUM nicht bieten kann: Das Gespräch zwischen den Teilnehmern. Die Zeichen des Wandels sind unübersehbar und viele Delphianer betrachten .NET immer noch als „Bedrohung“. Obwohl viele Entwickler in den Gesprächen dem Thema .NET immer noch mit Skepsis gegenüberstehen, haben Sie realisiert, dass sie trotz einer hohen Arbeitsbelastung im Alltag bereits jetzt in die Umlern-Phase einsteigen müssen, um nicht in wenigen Monaten vor einem tiefen Abgrund zu stehen. Da in .NET die verwendete Programmsprache nur eine untergeordnete Rolle spielt und Delphi allein in Zukunft nicht mehr ausreichen wird (weil zum Beispiel viele Tools aus dem .NET Framework SDK nur Source Code für die Sprachen C# und VB.NET generieren können), kommt in jedem Fall eine Menge Neues auf jeden Einzelnen zu. Ein Teilnehmer äußerte dabei in einem Gespräch die Befürchtung, dass Delphi (Win32) die Zukunft eines „Alt-Herren-Kompilers“ bevorsteht, der nur noch von den „Unbelehrbaren“ verwendet wird. Wir selbst haben es in der Hand, ob dieses Schreckgespenst Wirklichkeit wird. Bereits Heute gibt es viele Möglichkeiten, wie Delphi (Win32) und .NET zusammenarbeiten kann – Sie müssen daher ein bestehendes Projekt nicht in jedem Fall auf einem Schlag vollständig migrieren. In den Gesprächen zwischen den Sessions und am Abend konnten die Teilnehmer ihre bisherigen Erfahrungen und Strategien untereinander austauschen.

Resümee

War die Auslieferung des C#Builders als erste Borland-Entwicklungsumgebung für .NET noch ein dezenter Wink mit dem Zaunspfahl, so ist das Vorziehen von „Octane“ gegenüber Delphi 8 ein nicht mehr überhörbarer Weckruf. Als ein inzwischen in die Jahr gekommener Entwickler erinnert mich die Situation an das Jahr 1993. Als Turbo Pascal für Windows erschien, wollten auch damals viele Entwickler lieber bei Turbo Pascal 6 und der DOS-Oberfläche von Turbo Vision bleiben. Wir wissen alle, wie diese Story ausgegangen ist.

Daher mein Rat an alle die Delphianer, die sich nicht in den nächsten 3 Jahren in den Ruhestand verabschieden können: Nutzen Sie bereits Heute jede freie Zeit, um sich mit den Neuheiten von .NET vertraut zu machen. Und mit der im Februar am gleichen Veranstaltungsort stattfinden Kombination der dot.net Konferenz 2004 und der BASTA!2004 haben Sie als Teilnehmer die einmalige Gelegenheit, zwischen den Sessions beider Konferenzen pendeln zu können und somit die Luft aus beiden Welten (Borland und Microsoft) zu schnuppern. Die nächste Entwickler Konferenz findet vom 20. bis 24. September 2004 statt.

BorCon 2003

Eindrücke von der 14. Borland Konferenz

Als Veranstaltungsort für die diesjährige Konferenz hat Borland dieses Jahr San Jose, das sich selbst gern als Hauptstadt des Silicon Valley bezeichnet, gewählt. Verantwortlich für die Auswahl dieser historischen Gefilde war nicht zuletzt das 20 jährige Firmenjubiläum von Borland. Größtes Event auf der Konferenz war sicherlich der offizielle Launch von Delphi 8 für Microsoft .NET, auch wenn das nicht einzige neue Produkt war.

von Bernd Ua

Die Keynote der Konferenz begann mit einem kurzen Rückblick auf die Firmengeschichte und einigen Fotodokumenten aus der Frühzeit der Firma. Vom jugendlichen David I bis zur legendären Werbeanzeige für den ersten Pascal-Compiler. Nach dem Rückblick in die Vergangenheit folgte ein Ausblick auf die Zukunft und das ist nach Meinung der Borland-Verantwortlichen die ALM (Application Lifecycle Managment)-Strategie der Firma, die als nächster evolutionärer Schritt in der Produktpalette bezeichnet wurde. Angesichts eines Marktes, bei dem die Entwicklungsumgebungen und Tools in den Hintergrund zu treten scheinen und teilweise wie Eclipse kostenlos zu haben sind, sieht Borland seine Chancen in der Komplettaufstellung von Tools für den gesamten Entwicklungsprozess und Wartungszyklus von Software. Der offizielle Launch von Delphi 8 für das .NET Framework fand am Montag morgen im Rahmen einer General Session statt.

Delphi 8

Delphi 8 für .NET, dessen IDE dem C#-Builder ähnlich ist, soll im Dezember dieses Jahres ausgeliefert werden und erzeugt ausschließlich .NET-Anwendungen. Der Schlingerkurs des letzten Jahres zu der Frage wie das Produkt nun aussieht hat damit endlich ein Ende gefunden. Delphi 8 für Microsoft .NET, wie das Produkt offiziell heißt, enthält für die native Entwicklung eine Delphi 7 IDE. Delphi 8 kann mit zwei GUI-Frameworks umgehen, dem Windows Forms Framework von Microsoft und einer .NET Version der VCL. Diese ist wesentlich kompatibler zur VCL als das Kylix-Framework CLX und enthält neben den GUI-Komponenten auch die Datenbankkomponenten für BDE und dbExpress. Durch die VCL.NET ist es möglich eine Anwendung aus einem Source Code mit Delphi 7 sowohl für natives Windows als auch für .NET zu kompilieren. Die VCL.NET soll vor allem eine leichte und einfache Migration zu .NET ermöglichen. Komponenten für Windows Forms sollen in VCL.NET-Anwendungen verwendet werden können. Die Kompatibilität zu vorhandenen Code geht soweit, dass der Produktmanager Michael Swindell live und unter dem Beifall des Publikums eines der alten 16bit Demoprogramme von Delphi 1 (*Fishfact.dpr*) auf .NET „portierte“, indem er das Projekt öffnete und die Anwendung einfach nur neu kompilierte.

Neben diesen und anderen Migrations-Features (wie einem klassischen Export von Funktionen aus .NET DLLs) hat Delphi 8 auch neue Komponenten zu bieten. Die bekannten datenbanksensitiven Controls (DBEdit, DBGrid, etc) der Palettenseite Datensteuerung liegen alle in einer Webvariante für ASP.NET vor. Die neuen Komponenten sind zudem in der Lage auch im ASP.NET Designer der Entwicklungsumgebung bereits Live-Daten anzuzeigen, wie das in Borland Umgebungen Tradition ist. Als Reporting-Tool verwendet Delphi 8 die .NET Version von Rave-Report und nicht die in C#Builder enthaltenen Komponenten von Crystal Reports.

Neben den abwärtskompatiblen Datenbank-Engines BDE und dbExpress enthält Delphi 8 auch die Borland Data Provider für ADO.NET, die bereits in C#Builder enthalten sind. Allem Anschein nach, kommt hier ein neuer Treiber für Microsoft Access hinzu. Connection Pooling scheint allerdings in diesem Release nach wie vor zu fehlen.

Microsoft und Borland

Auf der diesjährigen Konferenz demonstrierten Microsoft und Borland Partnerschaft, selbst kleine Seitenhiebe auf Microsoft blieben dieses Jahr aus. Die Launch-Party für Delphi 8 sponserte der Microsoft SQL Server 2000. Auf der Keynote am Montag präsentierte David Threadwell, Manager der Microsoft .NET Platform, einen Ausblick auf die Technologie von Longhorn und das nächste Release von ASP.NET. Simon Thornhill sprach von einer engen Zusammenarbeit mit Microsoft im Bereich der .NET-Tools und kündigte die Unterstützung des Compact Framework für die kommenden Versionen der .NET-Tools an. Auf Konferenz-Sessions zum nächsten MS SQL Server (Yukon) konnte man bereits in Delphi geschriebene .NET Assemblies in Stored Procedures des Servers sehen.

Kylix und C++

Bei soviel Microsoft war es um Kylix auf der Konferenz recht still. Wenn überhaupt kamen Aussagen dazu nur nach Fragen aus dem Publikum. Eine neue Version von Kylix stellte Borland auf der Konferenz nicht vor. Die Nachfolge von Kylix für C++ tritt nach Aussagen von Blake Stone das neue Produkt C++BuilderX an, welches bereits in der ersten Version für Windows, Linux und Solaris zur Verfügung steht. Die Zukunft von Kylix für Delphi ist zumindest ungewiss. Auf Fragen aus dem Publikum gaben die Verantwortlichen ausschließlich ausweichende Antworten nach Art des folgenden Zitats von Michael Swindell „Wir haben noch keine RoadMap für Kylix angekündigt“. Offensichtlich wartet Borland hier zur Zeit noch ab und .NET hatte dieses Jahr die höchste Priorität.

Mr. Holland`s Opus

OPUS – Online Program Update Service von Ivotec

Im Zeichen des Internets hat sich vieles verändert, so auch die Wege, Kunden bestimmte Softwareupdates zukommen zu lassen. OPUS bietet die Möglichkeit, Updates online zu beziehen, wobei der Entwickler diesen Vorgang gezielt auf seine Kunden abstimmen kann.

von Dr. Frank Gunzer

Softwareupdates sind heute eine alltägliche Sache. In vielen Fällen werden sie durchgeführt, weil neue Features zur Verfügung stehen, viel zu oft aber leider auch, weil ständig irgendwelche Mängel behoben werden müssen, die zum Produkt-Release nicht mehr beseitigt werden konnten. Für große Firmen ist es kein Problem, einen entsprechenden Update-Server zu betreiben, für weniger große aber eventuell schon. An dieser Stelle kommt OPUS von Ivotec aus Freilassing ins Spiel. OPUS bietet die Möglichkeit, Dateien auf einem Server bei Ivotec abzulegen, die dann beim Kunden installiert werden können. Dieses bezieht sich nicht nur auf Updates, sondern auch auf erstmalig zu installierende Dateien.

Das Prinzip ist das Folgende: Der Entwickler benutzt das Programm OPUS Developer, mit dem er die entsprechenden Dateien festlegt und die Installationsmöglichkeiten beziehungsweise allgemein die Auswahlmöglichkeiten für den Kunden sowie die Verfügbarkeit gewisser Setups/Updates für einzelne Kunden bestimmt. Der Kunde hingegen benutzt das Programm OPUS Customer, in dem er, nachdem er seine Identifikationsmerkmale eingegeben hat, die für ihn bestimmten Setups/Updates auswählen kann und anschließend in der vom Entwickler per OPUS Developer festgelegten Art und Weise die Dateien installiert. Diese Dateien werden, das sei hier noch einmal gesagt, vom Entwickler mit dem Developer auf den Ivotec-Server übertragen und verbleiben dort, sodass keine Ressourcen vom Entwickler verwendet werden.

OPUS Developer

Zunächst soll der OPUS Developer vorgestellt werden. Im wesentlichen besteht er aus einer kleinen Übersichtsleiste an der linken Seite, bei der man auswählt, was man gerade verändern will (Kundendaten oder Updatedaten) und einem Bereich, wo man die dazu nötigen Daten eingibt. Bei den Kundendaten kann man zunächst einmal Username und Passwort festlegen, die der Kunde beim OPUS Customer eingeben muss, um an die Updates zu gelangen. Über Kundennummer und Firma kann der Entwickler eigene Ordnungskriterien eingeben. Die Daten kann man auch über eine Datei einlesen lassen. Sie können dann an den OPUS Server übertragen werden, sodass man für die Kundendaten auch keine Ressourcen auf der Seite des Entwicklers braucht. Der wichtigste Teils ist hier, dass man dem einzelnen Kunden gewisse Projekte zuordnen kann. Ein Projekt ist dabei ein Update beziehungsweise ein Setup, für die man den Status des Kunden festlegen (Kunde, Tester, Betatester) kann und gewisse Zugriffsmöglichkeiten auf das Projekt erlaubt, wie zum Beispiel unbegrenzten oder zeitlich begrenzten Zugriff oder aber auch nur auf gewisse Versionen begrenzten Zugriff. Welche Zugriffe der Kunde in der Vergangenheit bereits durchgeführt hat, kann man hier ebenfalls erfahren.

Bei den Updatedaten, der zweiten Möglichkeit auf der Übersichtsleiste des OPUS Developers, legt man die Eigenschaften eines Projektes fest. Hier werden zum Beispiel die zugehörigen Dateien ausgewählt, die Einträge in die Registry angegeben sowie die Verknüpfungen im Startmenü und/oder auf dem Desktop. Wenn man auf die Updatedaten wechselt, so sieht man rechts von der Übersichtsleiste sechs Tabs: *Einstellungen*, *Variablen*, *Dateien*, *Registry*, *Dialoge* und *Script*. Bei den Einstellungen legt man im wesentlichen die Sprache und den Status des Projektes fest. Die Sprache erlaubt es, für gewisse Sprachen bestimmte Setups/Updates zusammenzustellen. Dazu legt man am Anfang die Titel für die Sprachen fest (unter denen kann der Kunde später wählen) und wählt hier bei den Einstellungen für die Setups den entsprechenden Eintrag. Es ist ebenfalls einstellbar, ob der Computer nach dem Setup neu gestartet werden soll oder nicht. Das zweite Tab, die *Variablen*, ist sehr wichtig für die Gestaltung des Layouts, das der Kunde später im Customer sieht. Die voreingestellten Variablen sind Ordnernamen für die Installation, aber wenn man irgendwelche Texte später im Customer darstellen will, so müssen diese hier unter einem Variablennamen angelegt werden. Unter dem Tab *Dateien* legt man sich Dateigruppen mit den beim Kunden zu installierenden Dateien an und ordnet sie sogenannten Komponenten beziehungsweise Unterkomponenten zu. Diese

Anzeige

Anzeige

kann der Kunde dann auswählen, wenn er die entsprechende Installation durchführen möchte. Man kann hier noch als weiteres Gruppierungsmerkmal Setup-Typen festlegen und die Startmenü/Desktop-Verknüpfungen anlegen. Das Tab *Registry* dient zur Eingabe von Schlüsseln, die beim Kunden dann in die *Registry* eingefügt werden.

Das Tab *Dialoge* ist dafür da, Dialoge festzulegen, über die der Kunde dann gewisse Sachen einstellen kann, zum Beispiel den Ordner, in den die Installation erfolgen soll. Hier werden die üblichen Windows-Elemente (Buttons, Editfelder etc.) auf ein Formular gezogen und so praktisch eine GUI erstellt. Man findet auch eine Art Object Inspector, mit dem man die Eigenschaften der Objekte festlegt. Wichtig ist hier der Eintrag *Variable*, da man mit ihm steuert, unter welcher Variable die Usereingabe gespeichert wird, beziehungsweise welche Variable durch die Usereingabe verändert wird. Wenn diese zum Beispiel die selbe ist, die angibt, wohin die unter *Dateien* angegebene Dateigruppe installiert werden soll, dann kann der User so das Installationsverzeichnis auswählen.

Aus allen diesen Angaben wird am Ende unter dem Tab *Script* ein Installationsskript erstellt. Ein Wizard übernimmt dieses, aber man kann auch mittels einiger vorgegebener Funktionen das Skript erweitern. Wenn man sich dieser anschaut, so fällt die Sprache auf, denn die ist sehr stark an Delphi angelehnt.

OPUS Customer

Das Gegenstück zum OPUS Developer ist der OPUS Customer. Er ist das Programm, was letztendlich an den Kunden ausgeliefert wird. Nachdem der Kunde seinen Username und sein Passwort eingegeben hat, kann er neue Programme installieren (also neue, ihm zugeordnete Projekte) oder für bereits installierte ein Update durchführen. Die Deinstallation wird auch mit dem Customer durchgeführt. Bei der Installation werden zunächst die einzelnen Projekte angeboten, die dem Kunden zugeordnet sind. Anschließend sieht man die Komponenten, die für das zuvor gewählte Projekt mit dem Developer angelegt worden sind. Daraufhin erscheint der ebenfalls im Developer angelegte Dialog (oder auch mehrere, man ist nicht nur auf einen angewiesen, sondern kann auch eine Vielzahl von Dialogen einrichten), woraufhin schließlich den Download-Vorgang startet. Während dieses Vorganges wird zum einen angezeigt, welche Datei gerade heruntergeladen wird, und zum anderen per Statusbar der Status des Downloads dargestellt .

Für den Kunden gestaltet sich somit der Installationsbeziehungsweise Updatevorgang sehr einfach. Dem Entwickler wird auf der anderen Seite genug Flexibilität geboten, um diesen Vorgang zu steuern und auf bestimmte Kunden gezielt abzustimmen. Prinzipiell ist alles da, was in einem normalen Setup/Update-Vorgang benötigt wird: Unterstützung für verschiedene Sprachen, Dateigruppierungen, verschiedene Setup-Typen, durch den Benutzer auswählbare Komponenten und Interaktion mit ihm durch einen oder auch mehrere frei konfigurierbare Dialoge. Wenn man als Entwickler sein Produkt fertiggestellt hat, dann bietet OPUS eine sehr bequeme Art, das Produkt an den Mann zu bringen. Kundendatenbank etc. liegen auf dem OPUS- Server, und das Mitloggen der Downloads geschieht automatisch. Dieser Service wird von Ivotec natürlich nicht kostenfrei angeboten, aber eine einfache Kalkulation reicht in der Regel schon, um abzuschätzen, ob CD oder DVD mit entsprechendem Vertrieb wirtschaftlich günstiger sind oder nicht.

Der Aufwand auf Entwicklerseite ist eher gering, sofern man das Prinzip einmal verstanden hat. Leider ist die Hilfe nicht ausführlich genug, um gewisse Anfangsfehler zu vermeiden. Es steht zwar alles da, aber es fehlt der Hinweis, welche Schritte absolut notwendig sind und welche nicht. Wenn man erst mal festgestellt hat, dass außer den Registry-Einträgen und Verknüpfungen alle Schritte durchzuführen sind (insbesondere, dass man Komponenten und Dialoge anlegen muss, was einem am Anfang überhaupt nicht als zwingend notwendig erscheint), dann ist man sehr schnell im Geschäft. Das Anlegen von Kunden bereitet keine Probleme und das Zuordnen von Setups/Updates ebenso wenig. Durch OPUS wird es verlockend einfach, seinen Kunden einen entsprechenden Service zur Verfügung zu stellen. Es bleiben aber zwei Probleme zu berücksichtigen, die weniger mit OPUS selbst, als mehr mit dem grundsätzlichen Prinzip zu tun haben. Das erste ist Vertrauen und Sicherheit. Dass man seine Executables und zugehörige Dateien aus der Hand gibt, ist ja ein normaler Prozess, wenn man ein Produkt vertreibt. Userdaten auf einem nicht-eigenen Server zu speichern gehört aber in der Regel nicht dazu. Hier werden allerdings nur Firmennamen und OPUS betreffende Informationen abgespeichert, sodass auch übervorsichtigen Personen sehr schnell klar werden sollte, dass letztendlich keine Informationen in die Hände Dritter gelegt werden, die besser in den eigenen Wänden bleiben sollten. Das zweite Problem ist die Übertragungsgeschwindigkeit. Von Ivotec selber erfährt man, dass die Serververbindung sehr schnell ist und derzeit weiter ausgebaut wird (Details bitte bei Ivotec erfragen), aber dennoch traten beim Testen, wenn auch selten, kleine Wartezeiten auf. Beim Transferieren größerer Dateien könnte das unangenehm werden, erst Recht für Kunden mit einer langsamen Internetverbindung. In wie weit die momentane Verbreitung schneller Internetverbindungen einen kompletten Umstieg auf die Online-Verbreitung von Software erlaubt, ist derzeit schwer zu sagen, aber dieses wird ohne Zweifel die Zukunft sein. Sollte sich die beschriebene Möglichkeit anbieten, dann hat man mit der Kombination des OPUS Developers und des OPUS Customer ein Werkzeug zur Verfügung, mit dem man diese Aufgabe sehr bequem lösen kann. Auch in Sachen Zuverlässigkeit traten keine Probleme auf. Ob es zudem in wirtschaftlicher Hinsicht eine Alternative ist, kommt dabei aber auf den Einzelfall an.

OPUS kann zu Testzwecken über die Ivotec-Homepage (www.ivotec.com) erhalten werden; der Customer ist dabei ca. 4 mb groß, der Developer, der per Customer heruntergeladen wird, noch einmal ca.6 mb.

Links & Literatur

- www.ivotec.com

3D-Spiele mit C++ und DirectX

Alexander Rudolph

Dieses Buch zur Spiele-Programmierung ist in 21 Lektionen (hier „Tage“) eingeteilt und richtet sich an ambitionierte Einsteiger und Fortgeschrittene. Wie bei Büchern fürs Selbststudium üblich, wird jedes Kapitel mit einer Verständniseinheit (Quiz und Übungen) abgeschlossen.

Die erste Woche behandelt die Grundlagen der Spieleprogrammierung. Zuerst wird ein allgemeiner Game-Loop vorgestellt, worauf eine Beschreibung effizienter Berechungstechniken wie Lookup-Tabellen folgt. Wie gewöhnlich wird auch eine Einführung in 2D- und 3D-Welten mit zugehörigen mathematischen Grundlagen gegeben. Der Abschluss der ersten Woche bilden die wichtigen Themen Physik, Kollision und Kollisionserkennung. Da dieses Gebiet extrem wichtig ist, um realistisch wirkende Spiele zu erstellen, fällt hier die Themenvielfalt und Tiefe sehr positiv auf. In Sachen Physik werden neben den Grundlagen wie Beschleunigung und Impuls auch Reibung und andere Einflüsse beschrieben. Auch bei der Kollisionsabfrage werden mehrere Ansätze gezeigt und Testalgorithmen für viele verschiedene Objekte gegeben.

In der zweiten Woche wird dann DirectX 9 mit DirectX Graphics, DirectInput und DirectX Audio behandelt. Neben dem Basiswissen werden auch fortgeschrittene Techniken wie Billboardeffekte, Multitexturing oder Alpha Blending vermittelt. Die zweite Woche endet mit den Beschreibungen und Implementierungen eines Terrain- und eines Indoor-Renderers. In dieser Woche hätte ich eigentlich auch eine Einführung in Multiplayer-Spiele erwartet, was allerdings im kompletten Buch nicht behandelt wird.

Die dritte Woche ist der Erstellung eines größeren 3D-Spiels (Echtzeitstrategie/Space-Combat-Simulation) gewidmet. Hier werden die bisher vorgestellten Themen eingesetzt und viel Quellcode geschrieben. Darüber hinaus wird auch neues wie Partikeleffekte, Sky-Boxen oder Explosionen vorgestellt. Neben der Grafik spielen auch die KI und das Gameplay eine wichtige Rolle.

Insgesamt behandelt das Buch sehr viele Themen rund um die Spieleprogrammierung. Das einzige, was noch zu einem umfassenden Werk fehlt, ist das Thema Multiplayer. Die behandelten Themen gehen dafür ins Detail, sind meist ausreichend beschrieben und bieten Anfängern und Fortgeschrittenen viel Stoff für eigene Spiele. Was ebenfalls positiv auffällt, ist dass der Autor viel Wert auf die praktische Umsetzbarkeit legt, indem er stets effiziente Techniken verwendet. Als Leser sollte man Kenntnisse in C++ mitbringen, da bis auf einen Crashkurs auf der beiliegenden CD keine Erklärungen zu C++-Spezifischem gegeben wird. Anfänger in Sachen Spiele-Programmierung sollten zudem ein gutes Durchhaltevermögen besitzen, da die Themenvielfalt einiges fordert. Obwohl die Kapitel aufeinander aufbauen, können Fortgeschrittene das Buch auch querlesen und Tipps zu bestimmten Themengebieten herausziehen. Wer also ein umfassendes Werk zur Spieleprogrammierung sucht, sollte sich dieses Buch ansehen.

Steffen Rendle

Alexander Rudolph
3D-Spiele mit C++ und DirectX in 21 Tagen
768 Seiten, € 39,95
Markt + Technik, 2003
ISBN 3827264537

C++ – Einführung und professionelle Programmierung

Ulrich Breymann

Nachdem mir in der letzten Zeit eine ganze Reihe an C/C++-Büchern in die Hände gefallen waren, die sich recht bald als Mogelpackungen erwiesen, kann das C++-Buch von Ulrich Breymann auf positiv überraschen. Endlich ein C++-Buch, das nicht das kleine Einmaleins von C zum x-ten Male durchkaut, sondern ein echtes C++-Fachbuch, das sich praxisnah am internationalen C++-Standard ISO 14882 orientiert und im Sinne der Objektorientierung von C++ die Programmiersprache aufbaut. Nach Verlagsangaben wird das Buch an vielen Hochschulen eingesetzt. Und das merkt man. Sowohl positiv – die fachliche Kompetenz, die an UML ausgerichteten und sehr aussagekräftigen Grafiken, die Vollständigkeit, die Prägnanz der Sprache oder die didaktische Gliederung – als auch negativ – unterhaltsam ist das Buch in keinster Weise. Wobei Letzteres auch nicht von Nachteil sein muss, denn zur Unterhaltung kann man andere Bücher lesen. Allerdings fordert der akademische Anspruch den potenziellen Leser. Ob echte Anfänger (wobei die Definition, was für ein Anfänger, diskutiert werden kann), die der Verlag auf dem Einband als eine der potentiellen Zielgruppen angibt, damit glücklich werden, kann sehr bezweifelt werden. Fortgeschrittene Programmierer jedoch, die auf C++ umsteigen wollen, werden damit besser klar kommen.

Das Buch ist in zwei Teile untergliedert. Der umfangreichere Teil 1 wird als Einführung in die Sprache zusammengefasst und ist als Lehrbuch zu sehen. Teil 2 zur C++-Standardbibliothek ist als Nachschlagewerk und Erweiterung der Themen aus Teil 1 gedacht.

Die Einführung in C++ beginnt mit einer kurzen Abhandlung der Programmiergrundlagen. Da der Autor – wie schon gesagt – den objektorientierten Charakter von C++ von Anfang an im Auge hat, werden Klassen und Objekte schon auf der zweiten Seite erstmals eingeführt und schnell kommen Datentypen, Operatoren, Kontrollstrukturen u.ä. Grundtechniken zur Sprache. Für den erfahrenen Programmierer ist es sehr erfreulich, dass dabei Selbstverständlichkeiten einer jeden Programmiersprache wie Schleifen auf kleinem Raum kompakt und sehr abstrakt abgehandelt werden. Schon in Kapitel 2 kommen Themen wie Vektoren, Aufzählungstypen, zusammengesetzte und Benutzerdefinierte Datentypen oder Arrays zur Sprache. Nach einem kurzen Intermezzo zu Ein- und Ausgabe wird in Kapitel 4 die Programmstrukturierung auf einer sehr abstrakten Ebene behandelt. Funktionen, Prototypen, Gültigkeitsbereiche, Sichtbarkeit, Datenübergabevarianten, Compilerdirektiven, Makros, Templates, Inline-Funktionen, Überladen von Funktionen. Die Auflistung macht deutlich, das hier kein dünner Kaffee gekocht wird. Kapitel 5 widmet sich explizit der Objekorientierung und detailliert abstrakten Datentypen, Klassen, Objekten, Konstruktoren und Destruktoren in C++. Kapitel 6 unterbricht kurzzeitig den OOP-Themenstrang und kümmert sich um Zeiger, C-Arrays, C-Zeichenketten, *char*-Arrays, dynamische Datenobjekte und binäre Ein-/Ausgabe. Ein Einschub, der auf Grund der nicht vollkonsistenten OO-Philosophie von C++ an der Stelle viel Sinn macht und Basis für das nachfolgende OOP-Kapitel ist, in dem diese Kenntnisse vorausgesetzt werden. String-Klassen, Klassenspezifische Daten und Funktionen und Klassentemplates sind die Themen, denen in den nächsten Kapitel die weiteren, unabdingbaren OOP-Themen Vererbung, Polymorphismus, Operatorenüberladung und Exceptionhandling folgen. Die drei abschließenden Kapitel von Teil 1 bilden eine Art Gemischtwarenladen und drehen sich um die Ein- und Ausgabe sowie diverse Nützlichkeiten wie Container, Iteratoren, sortierte Listen, binäre Suchbäume, Handles, Matrizen, *trace*-Objekte, Einbindung von C-Funktionen, Unions, Bitfelder, Netzwerkprogarmmieru. GUI-Programmierung oder Typumwandlungen.

Dem Nachschlage-Teil 2 folgt zur Vervollständigung ein umfangreicher Anhang sowie ein mehrfach unterteilter Index, was den Nutzwert noch weiter steigert.

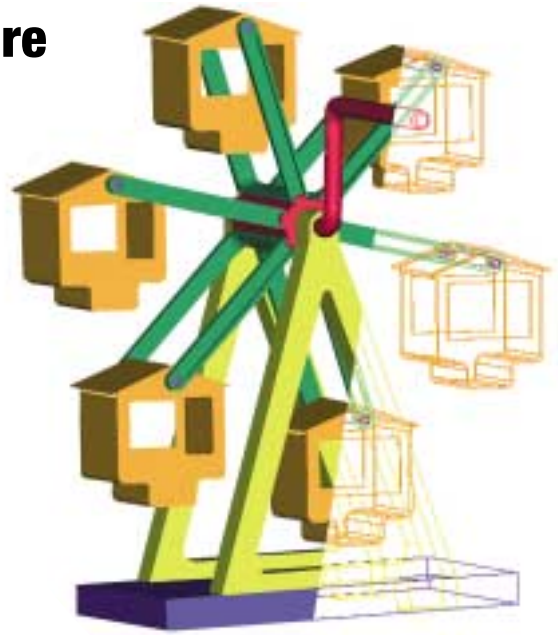
Fazit: Dem Anspruch, ein C++-Fachbuch zu sein, wird das Werk voll gerecht. Zwar würde ich das Buch nicht als Anfänger-geeignet bezeichnen, aber für mich persönlich ist es eines der besten C++-Bücher, mit denen ich mich bisher näher beschäftigt habe.

Ralph Steyer

Ulrich Breymann
C++, Einführung und professionelle Programmierung
704 Seiten, € 39,90
Hanser Verlag, 2003
ISBN 3446223304

MDA am Horizont

Model Driven Architecture



Die Entwicklergemeinde kennt ein neues Zauberwort, das in traditioneller Form der drei Buchstaben daherkommt: MDA (Model Driven Architecture). Wird man nun weniger programmieren und vermehrt modellieren oder induziert die Toolindustrie wieder neue Bedürfnisse? Beides, denn eine stärkere Formalisierung ist sowohl der Wunsch von uns geplagten Entwicklern wie auch die Bedingung der Industrie, Tools und Architekturen zu vereinheitlichen. All diese Vorhaben in UML 2.0 laufen unter dem Begriff MDA zusammen. Das Ziel ist es, ausführbare Modelle zu generieren.

von Max Kleiner

MDA im Banne der Technik

In einem ersten Teil möchte ich die Grundzüge der MDA aufzeigen, der zweite Teil ist dem praktischen Einsatz der MDA inklusive Projektstudie gewidmet. Der dritte Teil stellt die zugehörigen Tools vor. Solche CASE-Tools, die MDA als Etikett andrucken, erleben momentan ein Aufblühen, bei näherer Betrachtung aber eher ein Verglühen.

John Siegel, Vice President of Technology Transfer bei der Object Management Group (OMG), spricht von einer „enthusiastischen" Reaktion des Marktes auf MDA. Nach Siegel wird MDA schneller angenommen als jeder OMG-Standard vorher – und dazu gehören immerhin so weit verbreitete Standards wie Corba oder UML.

Ich kann dem hinzufügen, dass man den Begriff MDA fast inflationär braucht. Neue Geschäftszweige bauen sich auf, wie das EDOC (Enterprise Distributed Object Computing) oder die PDA (Process Driven Architecture). Das erklärte Ziel der OMG in Bezug auf EDOC ist beispielsweise die Vereinfachung der Komponentenentwicklung mit Hilfe eines auf UML 1.4 basierenden Frameworks, das zusätzlich noch mit der MDA kompatibel ist. Bei allem Respekt, aber die technologieunabhängigen Metamodelle stehen hier noch ziemlich am Anfang in Bezug auf EDOC. Jedes Metamodell der UML ist eben so gut wie seine Umsetzung, UML 2.0 hat darin noch einige Arbeit zu bewältigen.

Die letzte freigegebene Version von UML betrifft den Release 1.4, den die OMG im Mai 2001 verabschiedete. Der Release 1.4 betraf vor allem Erweiterungen in der Geschäftsprozessmodellierung sowie Anpassungen an die OCL.

Seit dieser Zeit arbeitet man an der Version 2.0, die den modellgetriebenen Ansatz (MDA) in den Vordergrund stellen will. Patterns bilden darin die Brücke zwischen der fachlichen und der technischen Sicht [1]. Ivar Jacobson erwähnte an einem Seminar im Dezember 2002 in Zürich den Begriff „Executable UML", welcher einer der fünf Makrotrends im künftigen Softwarebau sein soll.

Dass ein Architekturstil nicht nur etwas mit dem Bau und dem Zusammenspiel von Komponenten zwischen den Schichten zu tun hat, sondern auch mit den Personen und Rollen in der Organisation, die solche Komponenten generieren, macht MDA als Technik wie auch als Prozess interessant. Diese Einsicht kam schon im Juni `99 zu Tage, als das Gremium der OMG [2] eine neuartige Codegenerierung aus Modellen und Anforderungen begründete.

Wenn aber die Anforderungen schon im Modellansatz bestehen, spricht wenig dagegen, diese modellgetriebene Architektur bei ähnlichen Anforderungen immer wieder einzusetzen. MDA gibt es bereits für existierende Software, kommerziell wie auch Open Source [3]. Zumal MDA auf der höheren Abstraktion ja sprachunabhängig ist. Nun, was meint Model Driven Architecture?

Jede Sprache mit der zugehörigen Entwicklungsumgebung ist von einer inneren Architektur von Subsystemen abhängig, z.B. dem Framework der GUI, der Trennung von Interface und Implementierung oder den typischen Datenbankzugriffs-Komponenten mit ihren Connection-Strings.

Ein MDA-Generator sollte diese Architektur kennen, damit man sich in den Modellen nicht um diese technischen Details kümmern muss.

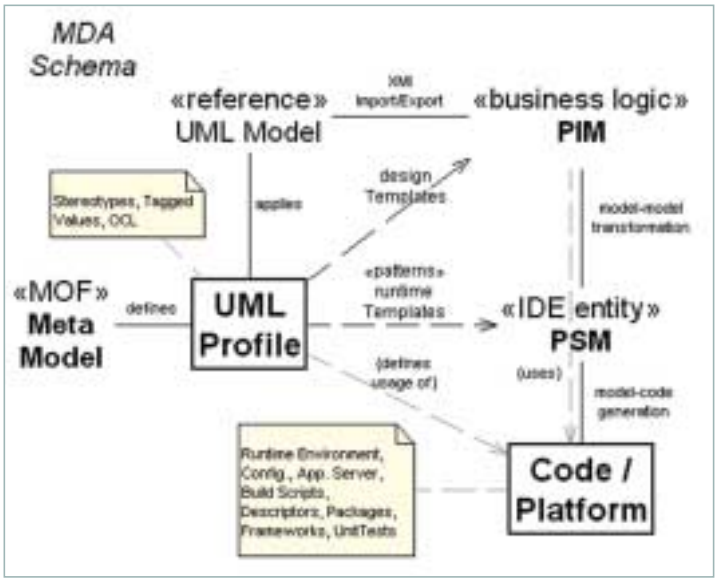


Abb. 1: Das MDA-Prinzipschema

Mein Paketdiagramm sollte also bereits wissen, ob ich z.B. dbExpress mit SOAP statt ADO mit COM einsetzen werde, wenn ich das Modell zum Generieren einsetzen will! Um diese „Probleme“ bei der Generierung zu lösen, schlägt die OMG vor, ein „Platform Independent Model“ (PIM) durch einen MDA-Generator zu erzeugen. Ein PIM ist ein UML-Modell, in dem die technischen Details nicht ersichtlich sind. Somit befindet sich das Modell auf einer höheren Abstraktionsebene und ist unabhängig von technischen Fakten wie der bekannten Programmiersprache oder Datenbanktechnik. Das PIM muss aber genug Details besitzen, um das generische Modell durch definierte Abbildungsregeln und Architekturmuster mit einem plattformspezifisches Modell (PSM) auf die Zielsprache abzubilden, d.h. innerhalb der Plattform mit zu generieren. Im Klartext heißt dies, es lässt sich nichts generieren, was nicht schon vorher in Form einer Referenzimplementation validiert, plausibilisiert und bestätigt wurde! Diese Referenz wird dann in Form von konkreten UML-Profilen weitergegeben. Skalierbarkeit, Wartbarkeit und Performance sind dann genauso gut wie die Referenz, aus der man die Abbildungsregeln ableitet (auch Transformation genannt).

Zurzeit existieren aber noch keine OMG-Spezifikationen für eine standardisierte Transformationssprache. Also, wo MDA draufsteht muss es nicht drin sein. Angenommen es gibt ein UML-Profil für dbExpress/CLX und eines für ADO.NET oder EJB. Mit einer Standardisierung der Transformation ließen sich dann die Profile austauschen! Ein PSM (Platform Specific Model) ist das spezifische und technisch abhängige Model, das die Architektur erzeugt.

Wer sich jetzt fragt, wie die Abgrenzung zu Design Patterns zu setzen ist, der weiß aus eigener Erfahrung, dass ein Tool aus einem Patternkatalog direkt sprachspezifischen Code produziert. So etwas wie einen Design-Pattern-Generator für Geschäftsprozesse gibt es noch nicht, aber einige Tools sind in dieser Hinsicht schon weit fortgeschritten. Also positioniert sich die MDA einen Schritt vor dem Design und ermöglicht bereits in der Analyse sprachunabhängige Klassendiagramme zu generieren.

MDA ist also mehr als ein Pattern-Expander, denn beim Erzeugen eines Pattern und den daraus resultierenden Klassenrümpfen ist dieser Schritt einmalig und nicht umkehrbar. MDA lässt sich aber wiederholt unter Erhaltung des eigenen Codes (geschützter Code) generativ einsetzen. Wobei auch die MDA nicht umkehrbar ist, will heißen aus einem generierten PSM lässt sich kein PIM mehr gewinnen.

Kernidee

Kernidee der MDA ist also die schrittweise Verfeinerung von der Analyse zum Design ausgehend von einer Modellierung der fachlichen Applikations-/Geschäftslogik, die völlig unabhängig von der technischen Implementierung und der umgebenden IT-Infrastruktur ist.

Abbildung 1 soll den Weg vom abstrakten Metamodell über die unabhängige Fachlogik zum konkreten Code auf der Plattform verdeutlichen, ähnlich einer CORBA-Spezifikation, der IDL (Interface Definition Language) und dem daraus folgenden Code. Was bedeutet dies nun in der Praxis? Man beginnt mit der Erstellung des Designs oder einer Referenz mit einem konventionellen UML-Tool. Das Tool muss aber fähig sein, mit UML-Profilen und Constraints arbeiten zu können. Durch einen XMI-Export erhält man eine austauschbare Designdatei (PIM), die man mit dem gewünschten UML-Profil einem MDA-Generator füttert. Der Generator transformiert die Designdatei mit dem Profil nach spezifizierten Abbildungsregeln (Template) in einen konkreten Architekturrahmen als typisierte Packages. Nun hat man das PSM, das immer noch ein Modell ist.

Mit der Generierung wird dann erstmals Code im Sinne eines Implementationsrahmen erzeugt. Die geschützten Bereiche sind zunächst noch leer. Es erfolgt dann die Entwicklung der eigentlichen Fachlogik in den Klassen, die mit einer weiteren Generierung durch Interpreter oder Compiler zum ausführbaren Code bezüglich der gewählten Plattform führt. Die einzelnen Schritte:

- Modelliere das PIM;
- Transformieren auf ein PSM;
- Generieren von Code aus dem PSM;
- Codieren der Fachlogik;
- Integrieren in die Plattform.

Wenn nun das Tool aus dem PIM ein plattformspezifisches Modell (PSM) generiert, sollte Einigkeit über die zu implementierende Plattform herrschen. Ist nun eine Plattform CLX, .NET oder J2EE, dann sollten die entsprechenden Klassen und Typen zum Zuge kommen, wie J2EE 1.2compliant oder CORBA 2.3compliant ORB. Was aber ist, wenn die Plattform eine Makro-Sprache mit Objektmodell oder sogar eine Spezifikation wie CORBA ohne Applikationsserver hat? Der PSM UML-Generator muss also nicht nur die Zielsprache, sondern auch die Zielplattform aus den Profilen kennen. Im Hinblick auf

Anzeige

die Ausarbeitung solcher Details stehen die MDA-Bewegung und die Toolhersteller noch ziemlich am Anfang. Zusätzlich soll in die MDA das Wissen über die zugehörige Businessdomäne einfließen.

Auch das Wissen über die zugehörige Fachlogik soll einfließen. Für Domänen wie z.B. Finanz- oder Versicherungswesen, Telekommunikation, Medizinaltechnik sollten die typischen abstrakten Prozesse schon vordefiniert sein und für die Erstellung der PIM als sogenannte „domain-specific core models“ bereitgestellt werden. Ein erster Ansatz ist Statemate, ausführbare Diagramme für Embedded Systeme [4]. Für den Austausch der Modellinformationen über Tool-Grenzen hinweg wird XML Metadata Interchange (XMI) eingesetzt. Der Name zeigt es bereits deutlich: XMI ist ein Mitglied der XML-Sprachfamilie. Die XMI-Norm definiert für das Metamodell eine XML Document Type Definition (DTD) oder ein Schema. XMI ist ebenfalls ein Standard der OMG.

Also bei der Transformation von einem abstrakten Modell hin zu einem konkreteren Modell soll man nicht nur Wissen über die technische Infrastruktur verwenden, sondern es wird auch Wissen berücksichtigt über die Businessdomäne, in der die Applikation eingesetzt werden soll. Die MDA birgt zusätzliches Potenzial für die Wiederverwendbarkeit von Modellen, da diese ja plattformunabhängig sind. Durch den Einsatz von standardisierter Transformation ist auch die Qualitätssicherung erhöht. Nach OMG sind dies die Hauptvorteile von MDA:

- Reduzierte Kosten bei der Entwicklung.
- Erhöhte Qualitätssicherung durch Selbstähnlichkeit.
- Schnellere Integration neuer Technologien.

Als Nachteil von MDA muss man erwähnen, dass strikt nach Forward Engineering gearbeitet wird. Nimmt man eine Änderung am Code vor oder ändert man das PSM entsprechend, aktualisiert das Tool die jeweils andere Seite nicht automatisch. Typische Refactoring-Techniken, die auch in einem Review zum Tragen kommen, sind deshalb bei MDA ein Problem.

Am 6. Januar 2003 wurde die überarbeitete Fassung der OMG übergeben. Am Ende dieses Jahres, da erfahrungsgemäß die Finalization Task Force 9 Monate dauert, erwartet die Entwicklergemeinde eine Freigabe der Version 2. Diese teilt sich auf in drei separate aber zusammenhängende Gebiete:

- **UML Infrastructure**, welche eine Vereinfachung und Modularisierung des Metamodells vorsieht und keinen Einfluss auf den Benutzer hat. Dieses Gebiet ist für die Toolhersteller und Methodiker interessant, die an Profilen und möglichen Stereotypen Freude und Nutzen haben. Zudem wird das Metamodell von sprachabhängigen Konstrukten (wie C++ oder ADA) gesäubert und sprachneutral, d.h. in OCL definiert.
- **UML Superstructure**, mit den eigentlichen Erweiterungen bezüglich der Notation und Syntax und somit für den Benutzer sichtbar und verwertbar. Ziemlich aufwerten will die OMG die komponentenbasierte Entwicklung mit z.B. einer präziseren Definition einer Schnittstelle mit erweiterten Signalen oder Nachrichten. Statische und dynamische Elemente werden auf Echtzeitfähigkeit getrimmt.
- **UML OCL**, die eine erhöhte Integration in das Metamodell und die Syntax vorsieht, sodass die Spezifikation durch ein Modell zum Code auch eine formale Sprache beinhaltet. Die OCL (Kasten „Glossar“) wird von einer Sprache der Bedingungen zu einer generellen Ausdruckssprache erweitert. Weiter folgt der verbesserte Modellaustausch durch die XMI, z.B. mit Layout Informationen zwischen den einzelnen Tools.

Es ist klar, das auf der einen Seite die Toolhersteller nun gefordert sind. Auf der anderen Seite wird die Vision des „Executable UML“ eine gravierende Änderung des Entwickleralltages zur Folge haben, sofern die Welt und schlussendlich auch unsere Kunden mit „standardisierter Individualsoftware“ und einheitlichen Geschäftsprozessen zufrieden sind. Etwas in dieser Art ist beim BizTalk Server von MS zu finden, der eine Generierung des Dokumenten-Workflows erlaubt. Mehr noch, bezüglich der Wartung und Pflege einer Anwendung sind dann die selben Fehler zu erwarten, da aus dem weltweiten Modellkatalog in den Codegeneratoren auch ähnliche Fehler entstehen können. Die wichtigsten Neuerungen (über 540 wurden im Sommer 02 der Revision Task Force überreicht) des Release 2 folgen nun in geraffter Form auf einen Blick:

- Für das Metamodell entsteht ein Sprachkernel;
- Zusätzliche Notationen bauen auf dem Kernel auf;
- Support für den Einsatz von Komponenten untereinander;
- Ausbau der Geschäftsprozessmodellierung (BPM);
- Interne Struktur für Bezeichner, Schnittstellen, Klassen, Typen und Rollen;
- OCL-Integration als generelle Spezifikationssprache eines Modells;
- State Event-Generalisierungen und Echtzeiterweiterungen;
- Erweiterungen für kompaktere Sequenzdiagramme;
- Präziseres Abbilden der Notation zur Syntax.

Praxis

Nun, wie man zum Modell/Code gelangt ist eine Frage der Technik, z.B. mit Design Patterns oder Idiome, oder man generiert mit der MDA schon in einer frühen Phase ganze Architekturpatterns [5]. Auch wenn es mittlerweile standardisierte Idiome wie *getter* und *setter*-Methoden, IDL-Generierung oder Listenklassen gibt, die sich mit Templates auch erzeugen lassen, hört die eigentliche Generierung innerhalb der Methoden einer Klasse auf. Die Geschäftslogik eines Systems, d.h. ihr eigentlicher Wert, wird auf einem hohen Abstraktionsniveau in einem plattformunabhängigen Modell modelliert.

In der Praxis sind bereits fertige PIMs erhältlich. Über automatisierte Transformationen in Modelle niederen Abstraktionsgrades werden dem PIM implementierungsspezifische Details hinzugefügt, wobei sogenannte plattformspezifische Modelle entstehen. Ein PSM dient dann als Grundlage für die Generierung aller Infrastrukturkomponenten der Anwendung (Code, Configuration, Test- und Build-

scripts, Deployment). Ein PSM lässt sich dann konkret als Paket- oder Komponenten Diagramm generieren.

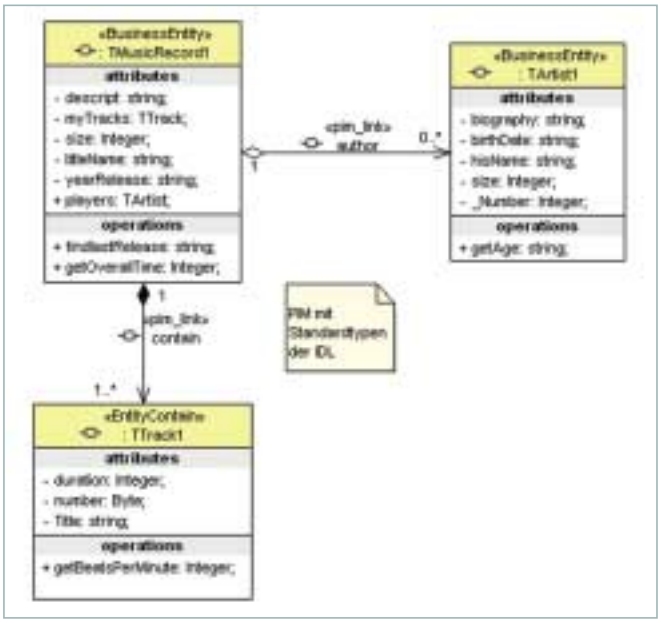
Auf der anderen Seite gibt es Prozesse, die ein modellgesteuertes Vorgehen vorsehen, sodass z.B. aus dem Paketdiagramm auf Stufe Design Patterns möglich ist, Code fast konstruktionsfertig zu erzeugen. In der folgenden Tabelle zeige ich, basierend auf dem MDA Schema, eine Übersicht zu den einzelnen Schritten. Fast allen Schritten gemeinsam ist der Einsatz der UML-Notation und Objektorientierung, iteratives Vorgehen sowie ein vermehrtes Ausrichten auf Patterns (Tab. 1).

Codieren wie Architekten, das klingt gut. Wichtig ist hier, bereits am Anfang eine klare Trennung von fachlichen und technischen Anteilen in den Klassen zu erreichen. In der Praxis kann man hier bereits einen Hauch von MDA in sein Klassendesign einfließen lassen. Stellen Sie sich eine Klasse *TCustome*r vor, mit den Methoden *Store* und *Undo*. *Store* und *Undo* sind technische Aspekte, die mit der Fachwelt des Kunden nichts zu tun haben. Oder kennen Sie einen Kunden, der als Mensch ein *Undo* besitzt. Ich kann ja auch nicht meinen vor Jahren erlebten Unfall rückgängig machen!

Im PSM sind in der Praxis Informationen über die eingesetzte Softwareinfrastruktur enthalten, wie z.B. ein J2EE-Applikationsserver, ein dbExpress Provider, oder ein ADO.NET-Zugriff. Die Transformation kann dann ausgehend vom PIM über mehrere unterschiedlich detaillierte PSM hin bis zum Code erfolgen. In der MDA sind die notwendigen Schritte bei dieser Transformation definiert und durch den Einsatz von Tools halb oder vollständig automatisierbar. Diese Techniken generativer Softwareentwicklung sind mit MDA und praxiserprobten Architekturbausteinen möglich.

Der Vorteil: Beim Austausch einer Middleware muss der Entwickler also nur die

Abb. 2: Unser erstes PIM als XMI-Import



geeignete regelbasierte Transformation verwenden, um aus dem unveränderten PIM ein neues, passendes PSM zu generieren.

Zentral ist hier das Modellieren der Geschäftslogik und -anwendung ohne Details zu ihrer technischen Umsetzung. Diese Trennung ist am Anfang recht gewöhnungsbedürftig, da man oft an irgendwelche technische Methoden aus dem Prototyping gebunden ist. Darauf aufbauend werden technologiespezifische Modelle erstellt, die eine konkrete Umsetzung in einer gewählten Technologie, wie z.B. J2EE, CLX, oder .NET beschreiben. Die Aufteilung in plattformunabhängige und plattformspezifische Modelle ermöglicht eine langfristige Anpassbarkeit an zukünftige Technologien, ähnlich dem Auswechseln des Datenbank-Providers.

Auch in der modernen Architektur ist der Detaillierungsgrad anscheinend verloren gegangen: Für auf dem Reißbrett entstandene Architektur haben sich Ornamente und Schnörkel an der Außenfassade als unnötig erwiesen. Doch die innere Komplexität der Gebäude ist gewachsen. Damit wird klar, dass nebst der Architektur auch die Funktionalität einen hohen Detaillierungsgrad aufweisen kann. Der Trend im Softwarebau wird sein, die Architektur zu standardisieren (J2EE, CLX, .NET) aber die Funktionalität zu erhöhen. Standardisieren kann auch vereinfachen. Denn in den seltensten Fällen wird künftig jeder Teil eines Modells ein anderer Sprachentyp sein, das entspräche der Arbeit mit Einzelelementen.

Schritt	Techniken	Beispiel	Output	Transformer
Fach Spezifikation Analyse/ Design	UML-Profile OCL	CD-Album Verwaltung	PIM	Model-Model Transformation
Technisches Modell Design	UML-Profile Patterns	XML-File Storing	PSM	Model-Code Generation
Implementierung	Compiler	Grid&File	Code	Micromethode
Integration	Generator	.NET	Plattform	Macromethode

Tab. 1: Prozessorientiertes Architekturmuster

MDA-Projekt

In dieser Kurzübersicht will ich den Einsatz von MDA nun pragmatisch darstellen. Ich werde die in der Tabelle erwähnten vier Schritte von MDA durchgängig erklären. Beim Projekt handelt es sich um eine Musikverwaltung mit den entsprechenden Medien wie Audio-CD oder Schallplatten. Als Tool kommt ModelMaker und ein selbstgebauter pseudo MDA/ D-Generator (D für Development) mit Templates zum Einsatz. Dieser Generator hat aber nur didaktischen Nutzen, demzufolge soll die Grundidee von MDA praktisch nachvollziehbar sein. Eine erste Umsetzung des Pseudogenerators war die Weiterentwicklung von DelphiWebStart [6]. Ich habe das Projekt mittlerweile der Community zugänglich gemacht und hoffe auf rege Mitarbeit unter: sourceforge.net/projects/delphiwebstart.

Wie oft hatte man schon die Idee, einem bestimmten Klassendesign aus fachlicher Sicht auch gleich die Bauanweisung

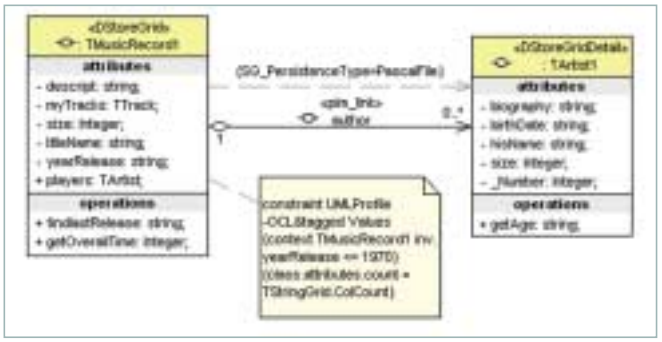


Abb. 3: Das Profil entsteht aus einer Referenz

für Speicherung und Darstellung der Daten mitzugeben. Denn die Umgebung von Fachanwendungen impliziert meistens eine ähnliche Struktur, sodass weitere Anwendungen wie eine Instanz aus einem Metamodell oder dem konkreten Profil entstehen könnten. Wenn man dann ein Grid (Gitter) generieren lässt möchte man auch gleich das Navigieren zwischen dem Grid und der Einzelansicht erledigt haben.

Mit einem ersten PIM und den Fachklassen *TMusicRecord*, *TArtist* und *TTrack* ist die Trennung der technischen Aspekte vollzogen. Ein Use Case in der Anforderungsanalyse führt indirekt zum vorliegenden PIM, das keine sprachspezifischen Typen aufweist. Dieser PIM soll jetzt mit einem Profile ein weiteres Modell erzeugen, das mir die Probleme der Persistenz und der Darstellung lösen soll. Mir schwebt vor, die Daten der Musikmedien filebasiert zu speichern und in einem Grid bearbeiten zu können. Die Geschäftsprozesse wie Suchen von Artisten oder Erstellen einer Statistik habe ich vorgängig in einem Activity modelliert. Zwischen einem Activity und der MDA gibt es keinen direkten Zusammenhang, die MDA geht bei einem PIM von Klassen- oder Zustandsdiagrammen aus, die Implementierung der Fachlogik erfolgt nach herkömmlicher UML-Art.

Was ich nun benötige ist das vorgängig erstellte Profil, das aus einer manuell erstellten Referenzimplementierung entstanden ist. Es ist kompliziert und auch wenig sinnvoll, direkt mit einer Template-Entwicklung zu beginnen, wenn noch keine Basis-Architektur vorliegt. So habe ich die Mechanik der Speicherung und Darstellung zuerst gebaut und dann mit Tags und Makros sozusagen ein Template als Profil erstellt. Dieses Profil will ich künftig *DStoreGrid* nennen. Zu sehen ist beispielweise eine Constraint die besagt, dass die Anzahl Kolonnen im Grid genau der Anzahl Attribute der Klassen entspricht, bezogen auf die einfachen Datentypen (*descript, size, titleName und yearRelease*):

```
{class.attributes.count = TStringGrid.ColCount}
```

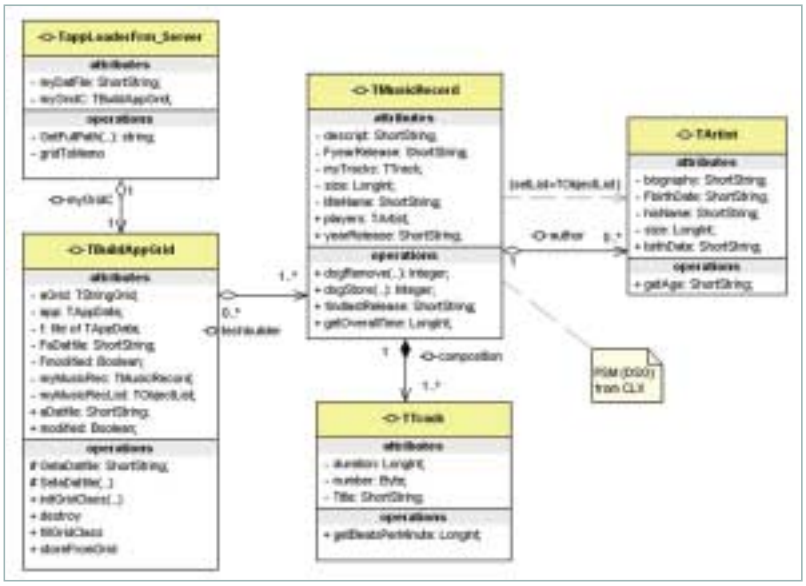


Abb. 4: Das PSM (aus der Transformation entstanden)

Die beiden Typen *TTrack* und *TArtist* sind Master-Detail Beziehungen die mit Listen oder Kollektionen aufgelöst werden. Die Klasse *TTrack* ist im Profil nicht ersichtlich, gehört aber im Template mit dazu. Auch eine Tagged Value ist mit von der Partie, das Profil kennt nun die Art der Speicherung als PascalFile, die als Constraint modelliert ist. Ein weiteres Value *{yearRelease <= 1975}* macht auf einen Schlag das gesamte System zu einer Oldies-Sammlung.

Mit dem Erweitern durch die Profile in UML 2.0, welches eine Art Sammlung von Stereotypen und Values darstellt, erhalten auch Packages eine Aufwertung. Auf diese Weise erhalten Klassen oder Pakete innerhalb einer gemeinsamen Fachdomäne ihr eigenes Stereotypenpaket, z.B. das Profil Musikmarkt. Ein Profil mit den zugehörigen Typen und sogar Operationen wie *getOverallTime* erlaubt, in einem Durchzug und in allen Diagrammen den Typ *Integer* auf *Float* oder *DateTime* zu wechseln! Ein Profil sollte gemäß der Spezifikation von Version 2 auch Toolübergreifend austauschbar und einsetzbar sein.

Das Klassendiagramm wurde in UML 2.0 verfeinert und im Hinblick auf den vermehrten Einsatz von Komponenten erweitert. Alle bestehenden Notationen wurden unverändert übernommen. Klassen und Interfaces erhalten neu einen *<part>*, das ist ein Teil einer Klasse. Dieses Teil ermöglicht in einer frühen Phase den Typ einer Klasse zu bestimmen, der dann später in der Implementierung mit Operationen und Attributen ausgearbeitet wird. Das transformierte PSM erhält den Namensraum DSG (*DStoreGrid*) und wird als Package gespeichert. Auch die Fachklassen sind nun mit technischen Details wie *dsgRemove* oder *dsgStore* angereichert, Design Patterns wie der Builder, Composite oder die Facade sind im definierten Package mit eingebaut (Abb. 4).

Packages sind in UML 2.0 genauer geworden, das Anzeigen der Klassen erhöht den Informationsgehalt, zudem lässt sich nun zwischen *<access>* und *<import>* als Abhängigkeitslinie unterscheiden; bei *<access>* greift eine Klasse auf die öffentlichen Elemente der anderen Klasse zu, bei *<import>* wird der ganze Geltungsbereich eines Paketes dem anderen Paket hinzuge-

Anzeige

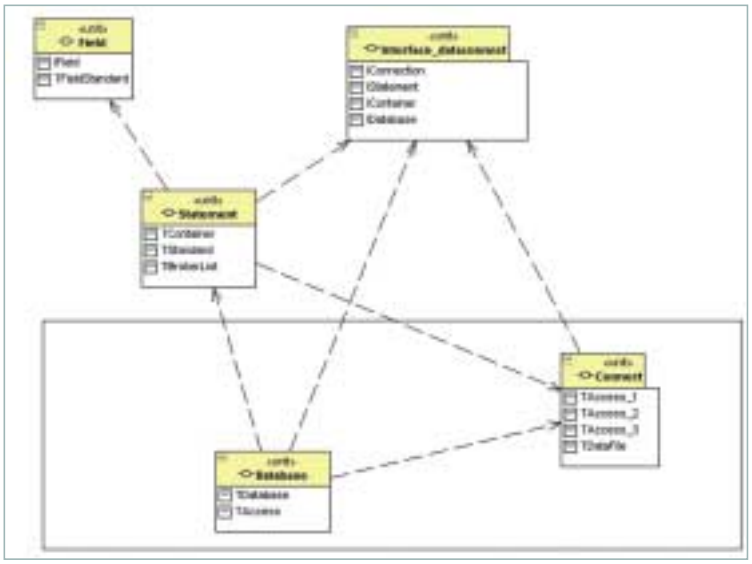


Abb. 5: Start zum Bau eines neuen Profils „dbExpress/CLX“

fügt oder eben importiert, z.B. bei einer IDL, DLL oder bei Type Libraries.

Auch Komponenten sind in UML 2.0 aufgewertet. Vermehrt ermöglichen <*ports*> und <*connectors*> zwischen einer angebotenen (Technik, Typ, Laufzeit und Signatur) und einer erforderlichen Schnittstelle das mögliche „Zusammenstecken“ prüfen zu lassen, um z.B. festzustellen, dass eine MSCOM nicht zu einem EJB oder einer CLX kompatibel ist. Durch die Detailsicht mit Hilfe von <*ports*> sind neu, geschachtelte Komponenten oder komplizierte Strukturen wie ganze Container besser beschreibbar.

Aus diesem PSM lässt sich nun im letzten Schritt der Implementationsrahmen erzeugen, den man mit Fachlogik, wie die Methode *findLastRelease*, weiter ausbaut. Diese Bereiche sind dann als geschützt markiert, sodass bei der nächsten Generierung kein Überschreiben erfolgt:

```
{**************** TBuildAppGrid *****************}
constructor TBuildAppGrid.initGridClass(vGrid: TStringGrid;
              vFile: shortString);
begin
 aGrid:= vGrid;
 aDatfile:= vFile;
 myMusicRecList:= TObjectList.create;
 myMusicRecList.ownsObjects:= true;
 with aGrid do begin
  ScrollBars:= ssAutoVertical;
  FixedRows := 1;
  FixedCols:= 0;
  ColCount:= 4;
  RowCount:= 2; //title is one row
 end;
end;
```

Der von der PSM generierte und wiederverwendbare Code hat vor allem die technischen Details gelöst, auszugsweise der Abschnitt wo das Einlesen der Daten aus dem Filesystem in die Objekte erfolgt und die Implementierung zusätzlich das im Konstruktor übergebene StringGrid mit den Musikdaten abfüllt (Led Zeppelin II, 38:90, 1972, Sec. Album by Zep with Whole Lotta Love Single):

```
.....
  AssignFile(f,aDatFile);
  Reset(f);
  try
   while not Eof(F) do begin
    Read (F, app);
    myMusicRec:= TMusicRecord.create;
    with myMusicRec do begin
     titleName:= app.Name;
     size:= app.Size;
     FyearRelease:= app.Release;
     descript:= app.descript;
     Cells[0,crow]:= titleName;
     Cells[1,crow]:= intToStr(size);
     Cells[2,crow]:= FyearRelease;
     Cells[3,crow]:= descript;
     myMusicRecList.add(myMusicRec);
    end;
    Inc(cRow);
    RowCount:= cRow +1; //new entry
   end;
  finally
.....
```

In einem weiteren Ausbau ist ein Profil zur Datenbankspeicherung gefragt, das in einem generativen Sinne im ersten Entwurf wie in Abbildung 5 aussehen kann.

Weitere Techniken sind im Gespräch, die alle eine gewisse Gemeinsamkeit mit MDA aufweisen, die sich „generative Programmierung mit adaptiven Mustern“ nennt. Diese Technik lohnt sich vor allem dort, wo man lange in der gleichen Fachrichtung entwickelt, also gute Kenntnisse der „Business Domain“ hat.

Auch der Begriff „Intentional Programming“ [7] gehört zur Thematik, welcher vorderhand ein interessanter MS-Traum ist, aber durchaus Potenzial im Zusammenhang mit .NET besitzt.

Muster im Großen wie im Kleinen, von Architekturmustern über Entwurfsmuster bis hin zu Codemustern, spielen bei der OO-Entwicklung eine Rolle, wenn es darum geht, Effizienz und Qualität gleichermaßen entscheidend zu steigern.

Ohne ein Tool allerdings, das die Verwendung solcher Muster aktiv unterstützt, ist nicht viel gewonnen. Zu vieles beschränkt sich dann auf das fehleranfällige, manuelle Abschreiben und Kopieren von Vorlagen. Ich komme zur abschließenden Tool-Übersicht.

Bold/ECO Framework

Im Folgenden lässt sich der Begriff Bold mit ECO gleichsetzen. Borland hat ja das Framework mit gesamtem Team innerhalb der Boldsoft übernommen, welche auch am ECO Projekt beteiligt sind. Octane Architect wird demzufolge das ECO Framework enthalten (EnterpriseCoreObjects) wie Delphi Architect, das die Bold-Komponenten beinhaltet. Boldsoft mit seinem UML-Modellierungs- und Architekturtool für Delphi ist ein weiteres Highlight des MVC Architektur Patterns. Zudem gibt es auch ein anerkanntes MVC Konzept für die Realisierung von Web-Applikationen in der Java-Welt [8]. Im Gegensatz zu Bold, benötigt ECO keine eigenen Controls mit der zugehörigen visuellen Anbindung (Achtung, verwechseln Sie den Begriff ECO nicht mit der eCommerce Initiative, eCo Framework genannt). Bold ermöglicht die Implementierung und Steuerung der MVC Architektur, d.h. eine Mittelschicht von Rendering-Klassen (Presentation Mapping) vermittelt zwischen den Geschäftsobjekten und der grafischen Repräsentation der Views. Zusätzlich bietet Bold ein durchdachtes Datenbankhand-

ling an, das neben dem Klassen zu Tabellen Mapping in eine relationale Datenbank auch Cached Updates sowie Transaction Handling erlaubt.

Bold benutzt ähnlich den UML-Profilen auch Tagged Values, die das Modell genauer spezifizieren, wie z.B. <*PMapper*> für die Art der Persistenz von Klassen. Auch mit der OCL lässt sich das Modell weiter steuern. Das von mir erstellte Modell, enthält bspw. die nötigen Informationen zur automatischen Generierung der Datenbank und den zugehörigen Tabellen. Dieses Klassen zu Tabellen Mapping lässt sich nach der Generierung weiter steuern, da in der Regel die gewünschte Datenbank mit Indexen und Triggern erweitert wird.

Das UML-Profil von Bold besteht aus vier Namen, die in den unterschiedlichen Elementen zum Tragen kommen. Bold setzt die Namen automatisch zum zugehörigen Kontext, sei es Modell, Profil, Code oder Plattform (Tab. 2).

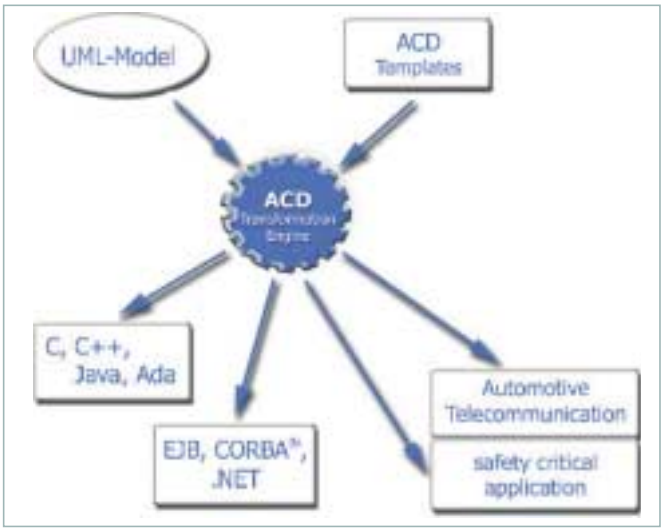
Bold kann in diesem Sinne nur die clientseitigen Regeln beeinflussen. Der Schemagenerator ist betreffend Änderungen des Modells vorbereitet, die Wahrscheinlichkeit ist groß, dass es Änderungen gibt. Das direkte Einbinden der Business-Objekte ist wohl der größte Vorteil von Bold. Der Klassencode für die Geschäftsobjekte lässt sich größtenteils automatisieren und auch bei Änderungen aktualisieren, ohne den Einsatz von bekannten Round-Trip Flags im Code, welche auf Veränderungen hin prüfen. Konkret erzeugt Bold innerhalb der Codegenerierung mindestens drei Units:

- <*UnitName*>.*pas* und <*UnitName*>_*Interface.inc* werden von Bold stets aktualisiert
- <*UnitName*>.*inc* beherbergt den individuellen Code

Modellname	Dieser Name wird im UML-Modell eingesetzt.
Expression-Name	Dieser Name kommt in der OCL als Ausdruck oder Profil vor
Code-Name	Wenn Code generiert wird, als konkreter Delphi Code.
SQL-Name	Als spezifische Namen in der Datenbank.

Tab. 2: Die vier Namen des UML-Profils von Bold

Abb. 6: ACD als eine Umsetzung der MDA



Der sogenannte Objektraum (objectspace) ist eine Instanz des Modells, wie das Objekt eine Instanz der Klasse ist. Bei der Implementierung der GUI sind wir auf die Bold eigenen Komponenten wie Grid oder Navigator angewiesen, da nur sie Verknüpfungen zur Business-Schicht ermöglichen und zudem bei Änderungen auch Modellgetrieben sind. Das Positionieren und Verknüpfen der Komponenten erfolgt größtenteils ohne manuelles Codieren (www.boldsoft.com).

MDA in StP

Software through Pictures nennt es „Architecture Component Development“ (ACD) Technologie. Auch unter dem Aspekt, dass die OMG dieses Thema unter dem eigenen Namen MDA weiterentwickelt, sind die Ideen hinter ACD und MDA identisch, mit dem Unterschied, dass sich Aonix diesem Thema schon seit Jahren widmet.

Einzigartig ist eben Aonixs ACD Technologie. ACD schlägt eine Brücke zwischen UML und realem Code, sodass mit einer Transformationsregel welche wiederum mit Hilfe von Templates automatisch fast 70 Prozent Code aus den Modellen generieren soll. Dahinter steckt eine spezielle Templatesprache, die man dem Transformator füttert.

StP/ACD ermöglicht die Trennung der fachlichen Anforderungen von den technischen Details der Implementierung. Die fachlichen Aspekte sind mit StP/UML Modellen und die technischen Aspekte als Template für den Transformator modellierbar. Dadurch erhält man einen sehr hohen Abstraktionsgrad in den Modellen und ein hohes Maß an Wiederverwendung durch die technischen Musterlösungen in den Templates. Mittels der Templatesprache, welche eine Art Abbildungsregeln definiert, kann der generierte Teil des Codes mehr als die Hälfte betragen.

Die Integration mit weiteren Produkten wie Config-Management-System, IDEs für verschiedene Programmiersprachen und Testwerkzeuge sorgen für die Abdeckung des gesamten Softwarezyklus.

Ameos ist bei StP das Modellierungswerkzeug der nächsten Generation. Mit den UML-Profilen bietet Aonix eine einfache Möglichkeit die Standard UML Notation zu erweitern und an projektspezifische Anforderungen anzupassen. Der Profile Editor von Ameos ermöglicht die Definition von Stereotypen und Tagged Values und die Verknüpfung mit Elementen des Metamodells. Damit wird der Entwurf, die Dokumentation und die einfache Wiederverwendung von UML Profilen sichergestellt. Einzigartig ist die Zuordnung von Farbe in der Profile Definition. Farbe wird dadurch auf semantischer Ebene verwendet und führt zu einer erheblich gesteigerten Lesbarkeit der erstellten Modelle.

Ein weiteres Kernstück von Ameos stellt der Modelltransformator auf Basis der MDA dar. Die PIMs werden dann über Transformatoren in die Zielumgebung überführt. Mit speziell zu diesem Zweck erstellten UML Profilen und den darauf abgestimmten Transformationsregeln, können die genannten Vorteile direkt in die Praxis umgesetzt werden. Neben den Standard-

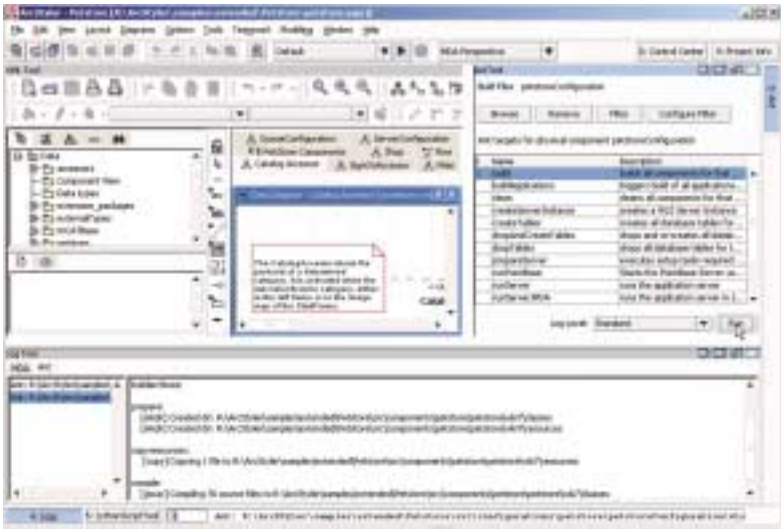


Abb.7: ArcStyler als MDA-Generator

lösungen für C, C++, Ada, Java und EJB, werden auch branchenspezifische Lösungen für den Automobilsektor und für sicherheitskritische Anwendungen angeboten. Ameos ist auf Win2000/XP, Solaris und Linux verfügbar und ist somit für den Einsatz in heterogenen Netzwerken prädestiniert.

Die Linux Version von StP 8.3 steht ab sofort zur Verfügung. Mit der neuen Linux Version vervollständigt Aonix das Angebot im Unix-Bereich. Die Positionierung von StP wird damit konsequent als Multi-Plattform Modellierungswerkzeug ausgebaut. Zu den unterstützten Plattformen gehören nun Linux (SuSE, Red Hat, Mandrake), Sun Solaris, HP UX und Win. StP lässt sich auch in einem heterogenen Umfeld mit mehreren Plattformen gemeinsam nutzen. (www.aonix.de).

ArcStyler 4.0

Der neue ArcStyler 4.0 biete jeweils das beste aus beiden Welten. Das Produkt ist in unterschiedlichen Versionen verfügbar, sodass der Anwender die für seine spezifische Aufgabenstellung optimale Lösung wählen kann. Die Standardversion verfügt über eine eigene, leistungsfähige UML- Engine (MagicDraw OEM von Nomagic). Zu den Features gehören u.a. Unterstützung von Profilen, MOF und Meta-Modellierung.

Aus eigener Erfahrung ist der ArcStyler in zwei Prozessen erhältlich: Als Komplettool oder als MDA/D Generator.

Viele Unternehmen verfügen bereits über Modellierungswerkzeuge. Um diese Investition zu schützen, wurde von Interactive Objects die Tool Adapter Standard (TAS) API für ArcStyler entwickelt. TAS ermöglicht es alle Arten von Modellierungswerkzeugen nahtlos in den ArcStyler einzubinden und diese damit zu einer leistungsstarken MDA-Lösung auszubauen. Wie kommt man nun zu den Profilen? Die anpass- und erweiterbaren MDA-Cartridges (eine Art Profile) lösen dieses Problem. Als Template enthalten sie sämtliche technologiespezifischen Mechanismen für die automatischen Transformationen von Modellen. Dazu gehören Mapping und Verifier Engines, Pattern Matchers und erweiterbare Generatoren.

Es können Cartridges für prinzipiell jede Architektur erstellt werden. Die in diesen Cartridges enthaltene Automatisierung wird ebenfalls in UML modelliert. ArcStyler wird mit Standard- und Referenz-Cartridges für Java, J2EE, und .NET ausgeliefert, die bei Bedarf individuell anpassbar sind. Eine Vielzahl weiterer Cartridges (Open Source) steht den Anwendern über die weltweite MDA-Community zur Verfügung (www.mda-at-work.com).

Alle ArcStyler-Module verwenden ein gemeinsames und offenes UML/MOF/ JMI-Repository. Damit entfällt nicht nur ständiges Importieren, Exportieren und Synchronisieren. Der ArcStyler deckt weit mehr Aspekte der Softwareentwicklung ab als reine UML-Modellierung und einfache Code-Generierung über simple Patterns. Mit seiner MDA-Engine bietet er nicht nur intelligente Modellierungsunterstützung mit Validierung auf Modellebene (Modell-Debugging unter Berücksichtigung der Plattform), sondern gleichzeitige Unterstützung mehrerer Zielplattformen und Architekturen, flexible und vollständig anpassbare Transformations- und Generierungsfunktionen (www.iO-Software.com/ mda).

XCoder von Liantis

Die Liantis GmbH hat ihr MDA-Tool XCoder als Open Source freigegeben. Mit diesem Schritt möchte sie den in zahlreichen Projekten praxisbewährten XCoder einer noch größeren Entwicklergemeinde zur Verfügung stellen. Durch die Freigabe als Open Source haben Unternehmen die Sicherheit, alle notwendigen Ressourcen zur eigenen Anpassung bzw. Weiterentwicklung zur Verfügung zu haben. Anpassbare Standardlösungen für Java, C++, C#, J2EE und .NET liegen bereits in der Standarddistribution vor.

XCoder ist komplett mit MDA erstellt, in Java geschrieben und wird auch mit MDA erweitert. Alle dem XCoder zugrunde liegenden UML Modelle sind ebenfalls Bestandteil der Standarddistribution. Liantis verspricht sich ein schnelleres und häufigeres Feedback, wenn weitere Entwickler und Anwender an dem Projekt beteiligt sind. Die schnellere Beseitigung von Fehlern wird ebenfalls allen Anwendern zugute kommen. Durch offene Schnittstellen wie XMI sind praktisch alle marktgängigen UML-Tools unterstützt (www.liantis.com, sourceforge.net/projects/xcoder/).

Tau von Telelogic

Im Oktober 2002 hat Telelogic bereits Tau eingeführt, das UML 2.0 unterstützt. Tau ist in der Industrie auch als Tool für die Echtzeitmodellierung geschätzt.

Änderungen und immer wieder neue Anforderungen während des Entwicklungsprozesses machen Software immer komplizierter. Gut, wenn man mit dem eingesetzten Tool die Architektur des Systems dann überarbeiten kann, ohne dass sich das nach außen sichtbare Verhalten der Anwendung ändert. Tau hat einen Syntax-Check um die Formulierung des Modells zu prüfen inklusive eingebauter Simulation von Sequenzdiagrammen!

TAU gibt es in zwei Versionen mit UML 2.0:

Anzeige

- TAU/Architect für die System- und Software Architektur.
- TAU/Developer für Echtzeit Systeme und Codegenerierung.

Die beiden Tools lassen sich auch nahtlos in das firmeneigene Tool Doors integrieren. Doors unterstützt das Anforderungsmanagement. In dieser Situation sind funktionierende Schnittstellen zwischen den einzelnen Tools unabdingbar. Diagramme für Use-Cases lassen sich beispielsweise elegant sowohl in Doors als auch in Tau integrieren. Für den Architekturentwurf bietet Tau mit Architekturdiagrammen sehr weitreichende Unterstützung an: Die einfache Darstellung hilft, IT-Systeme zu gliedern, Komplexität zu reduzieren und so robuste Systemarchitekturen zu bauen. In den Diagrammen definieren Sie grafisch die Abhängigkeiten und Hierarchiebeziehungen zwischen Paketen oder Komponenten. Auf diese Weise kann man unter anderem vorgeben, welche Schnittstellen eines Paketes sich benutzen lassen und welche zu implementieren sind (www.telelogic.com/resources/index.cfm).

Glossar

UML-Profile
Ein UML-Modell wird mithilfe von spezifischen Erweiterungen (Stereotypen, Tagged Values, OCL) innerhalb einer formalen Designsprache (UML-Modellierungssprache) genauer beschrieben.

Stereotyp
Stereotypen geben kommentierend die möglichen Verwendungszusammenhänge einer Klasse, einer Assoziation oder eines Paketes an. Stereotypen klassifizieren die möglichen Verwendungen eines Modellelementes. Ein Element, beispielsweise eine Klasse, kann beliebig viele Stereotypen aufweisen (*self, import, global, #*).

Tagged Values
Werte die in einem Modell konkret als Implementierungsanweisung zugewiesen werden, wie: *PersistenceType = Transient, Derived = True, InitialValue = Ord(`A`)*

OCL
Die Object Constraint Language ist eine semiformulare Sprache zur Definition von Bedingungen/Einschränkungen. Sie definiert eine standardisierte Sprache zur Beschreibung von Zusicherungen innerhalb von Modellen (*{FIndex < FList.Count}, {salesman.knowledgeLevel >= 5}*)

Plattform
Dieser Begriff ist innerhalb der MDA kritisch zu betrachten, es ist das Laufzeitsystem gemeint, dazu zählen Frameworks wie auch Bibliotheken, die dann in der spezifischen Ausprägung auf der jeweiligen Hardware-Plattform laufen. Der Generator benutzt die Plattform.

MDA-Generator
Der Parser des Generator Backend interpretiert ein UML-Profil oder ein Template, erzeugt den plattformspezifischen Code via FileStreaming, scannt und identifiziert die geschützten Bereiche bevor er die schon vorhandene Fachlogik in den neuen oder modifizierten Implementationsrahmen schreibt.

objectiF

Für die unterschiedlichen Sprach- und Technologie-Präferenzen bietet microTOOL mit objectiF jetzt übrigens in zwei Tool-Varianten an: Die gerade freigegebene objectiF Visual Studio .NET Edition ist nahtlos mit Visual Studio .NET integriert und für C# und VB .NET optimiert. Sie ist damit speziell auf die .NET-Entwicklung zugeschnitten. Die „Vollversion“ von objectiF – sie heißt Enterprise Edition – wird in Kürze ebenfalls freigegeben (objectiF 5.0). Sie bietet alles, was objectiF Visual Studio .NET Edition mitbringt und unterstützt neben C# und VB .NET natürlich weiterhin auch Java, C++ und Delphi.

Zur Funktionalität der microTOOL Suite .NET gehört außerdem das Konfigurationsmanagement, also die Verwaltung aller im Projektverlauf entstehenden Ergebnisse. Für die agilen Techniken des Unit-Tests und der kontinuierlichen Integration wurden die Open Source-Produkte NUnit und NAnt in die microTOOL Suite .NET eingebunden. Man findet – im Unterschied zu anderen Suites – in der microTOOL Suite .NET aber nicht nur optimal aufeinander abgestimmte Werkzeuge, sondern auch konkrete Unterstützung beim Vorgehen in Form von actiF, einem Prozess für die agile Entwicklung. actiF schafft die Grundlage für die maschinelle Planung und Steuerung von Projekten mit in-Step. Als nächstes wird sich in dieser Richtung etwas bei objectiF tun. Eine Integration mit Eclipse ist bereits in Arbeit. Sie wird einen ähnlich hohen Integrationsgrad wie objectiF Visual Studio .NET Edition haben (www.microtool.de/).

Innovator 8.1

Mit einer ausgefeilten Integration der IBM WebSphere-Entwicklungsumgebung bietet Innovator 8.1 eine leistungsfähige Plattform für die Modellierung und Realisierung komplexer Anwendungen im Umfeld von EJB, Web Services und MDA. Auch von Eclipse aus können Sie nun nahtlos auf die Innovator-Modelle zugreifen. Zusätzlich erweitert die Anbindung von PVCS Dimensions die Möglichkeiten im Konfigurations- und Versionsmanagement.

Innovator 8.1 eignet sich hervorragend für die Definition und Erstellung von Anwendungen im Umfeld von Web Services und MDA. Die strikte Ausrichtung und Anpassung an Standards sichert auch in Projekten nach herkömmlichen Methoden und Verfahren getätigte Investitionen. Abschließend kann man sagen, dass man Innovator eigentlich nicht gerecht wird, wenn man den Einsatz allein auf UML bezieht. Die verschiedenen Tools kommen am besten in einem Umfeld zum Einsatz, wo es eine Durchmischung aus strukturierter und OO-Entwicklung gibt. Wo immer es möglich ist unterstützt Innovator bestehende Industriestandards, was das Tool gerade auch für große und, sage ich mal, schwerfällige Projekte interessant macht. Mit fünf Tools unterstützt Innovator sowohl objektorientierte sowie klassische, strukturierte Technologien (SA/SD und ERM/SERM) in Erstellung und Wartung komplizierter Softwaresysteme bis hin zur Geschäftsprozessmodellierung (www.mid.de/de/news/profile/).

Links & Literatur

[1] Max Kleiner: „Patterns konkret“, Software & Support Verlag, 2003
[2] MDA Specifications, www.omg.org/mda/
[3] b+m Generator Framework www.architectureware.de, b+m Informatik AG
[4] Statemate, www.ilogix.com/products/magnum/index.cfm
[5] Max Kleiner, Designermodell; *Der Entwickler* 1/2001
[6] DelphiWebStart: sourceforge.net/projects/delphiwebstart
[7] www.csharp-info.de
[8] Struts: jakarta.apache.org/struts/

MDD mit Bold für Delphi

Modellgetriebene Anwendungen mit Delphi entwickeln

Lange Zeit hat Borland Delphi als reines RAD-Tool positioniert und Entwickler, die UML-Tools mit Delphi einsetzen waren Mangelware. Auch andere Anbieter, wie Rational Rose haben Ihre Delphi-Anbindungen (RoseLink für Delphi) nur gut versteckt angeboten und erst recht nicht auf Messen gezeigt. Mit Delphi 7 hat sich dies zumindest von Seiten Borlands geändert. Die Enterprise Edition enthält bereits das UML-Tool ModelMaker und die Architect Edition steuert native Delphi-Komponenten bei, mit deren Hilfe sich eine Anwendung direkt aus dem UML-Modell heraus betreiben lässt. Wie Sie mit diesen Komponenten UML und RAD unter einen Hut bringen, verrät Ihnen dieser Artikel.

von Bernd Ua

Die nur in der Architect Edition enthaltenen Bold-Komponenten waren zu der Zeit als Delphi 7 erschienen ist, noch das Hauptprodukt der kleinen Firma BoldSoft aus Stockholm. Wenige Monate später allerdings hat Borland die Firma übernommen, sodass es sich inzwischen nicht mehr um „Fremdkomponenten“ handelt, sondern um Borland-eigene Komponenten. Bereits bevor die Komponenten in die Architect Edition aufgenommen wurden, konnte BoldSoft einige erfolgreiche größere Projekte, unter anderem für das schwedische Parlament, mit den Komponenten vorweisen.

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter

www.derentwickler.de

Wozu Bold-Komponenten verwenden ?

Gerade bei klassischen Datenbank-Projekten tut sich oft eine Lücke auf, zwischen Analyse und Design einer Anwendung in UML und der Entwicklung der Anwendung auf der anderen Seite. Die Klassen und Strukturen des UML-Modells werden in relationale Tabellen einer Datenbank „übersetzt“ und ein beträchtlicher Anteil des Programmcodes beschäftigt sich damit, die Lücke zwischen relationalen Tabellen und Businessobjekten wieder zu schließen. An dieser Stelle setzen die Bold-Komponenten an. Die Komponenten betten das UML-Modell der Business-Objekte in die Anwendung ein und „betreiben“ die Anwendung aus dem Modell heraus. Das Objekt-relationale Mapping der Klassen auf die Tabellen einer relationalen Datenbank kapseln die Komponenten komplett, sodass sich der Entwickler im günstigsten Fall um diesen Punkt gar nicht mehr kümmern muss. Alle gängigen Engines von Delphi, angefangen von BDE und dbExpress bis hin zu IBX und ADO, werden von den Bold-Komponenten unterstützt.

Das Bold Framework

Das Bold Framework selbst setzt sich aus mehr als 80 Komponenten und zahlreichen zusätzlichen Klassen zusammen, die in etwa 15 Packages enthalten sind. Glücklicherweise braucht nicht jede Anwendung alle vorhandenen Komponenten. Die Klassen des Frameworks bilden eine mehrschichtige Architektur, die in ein Layer für die Persistenz, ein Layer für die Business-Objekte, und zwei Layer für die GUI-Repräsentation sowie das GUI-Mapping aufgeteilt ist.

Kern einer Bold-Anwendung ist die Komponente *TBoldModel*. Diese Komponente enthält das UML-Modell der Business-Klassen. Mit Hilfe einer Link-Komponente kann ein Klassenmodell aus ModelMaker, Rational Rose oder einer XMI-Datei importiert werden. Über den Komponenteneditor von *TBoldModel* steht zudem ein nicht-grafischer UML-Editor zur Verfügung um ein importiertes Modell nach zu bearbeiten oder anzupassen.

Theoretisch kann mit diesem Editor auch ohne zusätzliche UML-Tools das Klassenmodell der Anwendung entworfen werden. Bei komplexen Modellen ist davon allerdings abzuraten, allenfalls für einen ersten Test ist der Entwurf mit diesem Editor geeignet.

Für die Darstellung und Speicherung der Klassen des Modells sind weitere Komponenten des Frameworks erforderlich. Statt normaler datensensitiver Controls setzen Sie spezielle Bold-Komponenten ein, die sich auf der Palettenseite Bold Controls befinden und den üblichen Bereich von Navigator und Grid bis zum Editierfeld abdecken. Den Zugang zu den darzustellenden Daten erhalten diese Controls über Komponenten der Seite Bold Handles. Für die erste Bold-Anwendung ist es sinnvoll, die Standardvorlage Default Bold Application aus der Objektablage zu verwenden. Diese enthält neben einem Hauptformular, einen Startup-Dia-

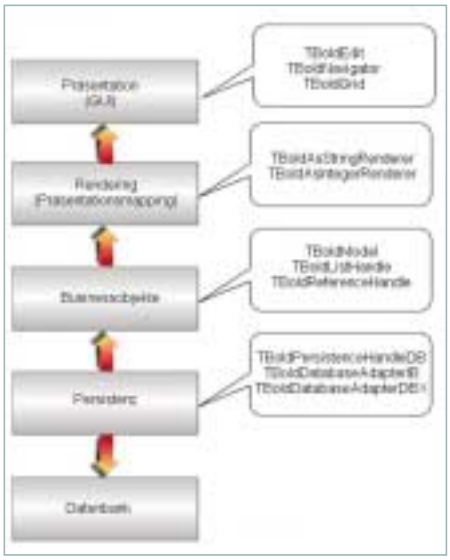


Abb. 1: Schichtenarchitektur im Bold Framework

log zur Datenbankgenerierung sowie eine Datenmodul mit der notwendigen Grundausstattung an Komponenten.

Entwicklung einer Bold-Anwendung

Der Ansatz für die Entwicklung einer Bold-Anwendung unterscheidet sich fundamental vom klassischen Vorgehen in Delphi. Startpunkt ist hier nicht ein bestehendes ER-Diagramm einer relationalen Datenbank sondern der Anwendungsentwurf in UML. Als UML-Tool kommen prinzipiell alle Tools in Frage, die XMI exportieren können. Da im Lieferumfang der Architect-Edition ModelMaker enthalten ist, bietet sich zunächst dieses Werkzeug für die Modellierung an. Wenn Sie bereits mit ModelMaker gearbeitet haben um Code zu generieren oder Refactoring von Klassen zu betreiben, sollten Sie sich allerdings in der Arbeitsweise etwas umstellen. Für Klassen, die Sie mit den Bold-Komponenten verwenden wollen, gelten einige Einschränkungen bezüglich der Attribute und Methoden auf die ich später im Artikel noch genauer eingehe. Das Zusammenspiel von ModelMaker und Bold ist an einigen Stellen nicht ganz reibungslos, da ModelMaker sehr stark auf die Code-Erzeugung ausgerichtet ist. Die Bold-Entwickler neigen sogar dazu ModelMaker nicht als UML-Modellierungstool sondern als Pascal-Code-Generator mit UML-Oberfläche zu bezeichnen (Abb. 2).

Die Beziehung zwischen Kunden und Aufträgen ist in diesem Diagramm als Komposition modelliert, ebenso wie die Beziehung zwischen Auftrag und Auftragspositionen. Eine kleine Bold-Anwendung soll dieses Klassendiagramm verwenden und eine Datenbank für die Klassen des Diagramms erzeugen. In der oben genannten Standardvorlage für eine Bold-Applikation ist bereits eine Linkkomponente *TBoldUMLMMLink* für ModelMaker enthalten. Bei dieser Komponente wird die ModelMaker-Datei, die das Diagramm enthält, in der Eigenschaft *FileName* eingetragen. Die Verbindung zur Modellkomponente erfolgt über die Eigenschaft *BoldModel* dieser Komponente. Ist der Dateiname eingetragen, können Sie das Klassenmodell über den Bold-UML-Editor von *TBoldModel* importieren oder aus dem Popup-Menü der Komponente einfach den Eintrag *Import via Link* wählen.

Zugriff auf die Klassen und Instanzen der Businessobjekte erhalten Sie in Bold über *Handle*-Komponenten. Zwei dieser *Handle*-Komponenten sind bereits in der Standardapplikation enthalten und mit der Modellkomponente verbunden. Bei der einen Komponente *stiMain* handelt es sich um die Klasse *TBoldSystemTypeInfoHandle*, welche die Typinformationen des Modells verwaltet und zur Verfügung stellt. Die zweite Komponente *bshMain* vom Typ *TBoldSystemHandle* ist mit *stiMain* verknüpft und bietet zur Laufzeit Zugriff auf die Instanzen der Businessobjekte. In realen Applikationen werden Sie ohne generierten Code für die Klassen kaum auskommen, deshalb ist die Codeerzeugung in der Komponente *stiMain* über die Eigenschaft *UseGeneratedCode* standardmäßig eingeschaltet. Prinzipiell sind die Bold-Komponenten jedoch in der Lage die Klassen des Modells auch ohne generierten Code zu verwenden. Für einen ersten Testlauf kann die Code-Erzeugung deshalb abgeschaltet werden.

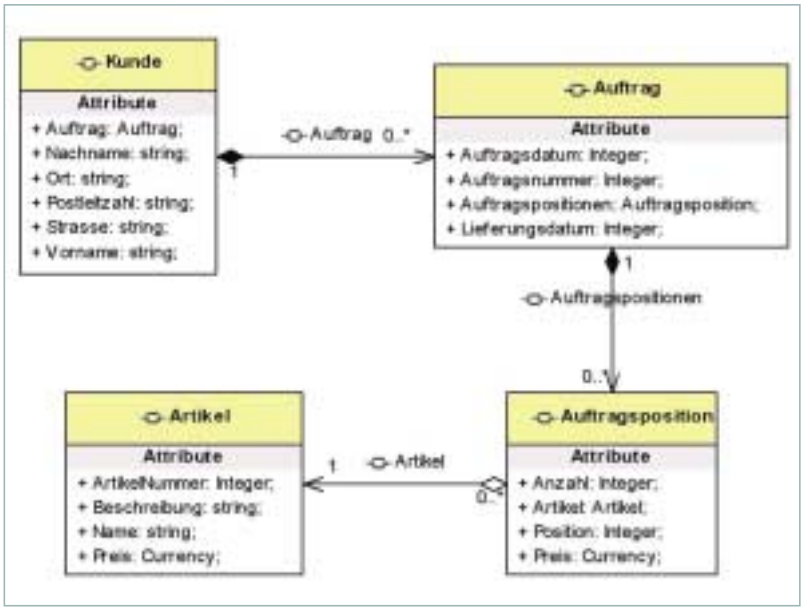


Abb. 2: Beziehungen zwischen Kunden, Aufträgen, Auftragspositionen und Artikeln

Installation von Bold und ModelMaker

Bevor Sie ModelMaker und Bold installieren, sollten Sie Delphi registriert haben. Findet ModelMaker keine Registrierungsinformationen, läuft die Anwendung nur im Demo-Mode. Findet Bold bei der Installation das Programm ModelMaker, wird automatisch ein Bold-Plugin für ModelMaker installiert und registriert. Installieren Sie ModelMaker erst nachträglich, müssen Sie das Plugin *BoldMMPlugin.dll* selber in das Verzeichnis *ModelMaker\Plugins* kopieren und via *regsvr32.exe* registrieren. Für die Bold-Komponenten gibt es zudem ein Update im Internet, welches Sie auf jeden Fall verwenden sollten. Da das Update die vorherige Deinstallation des Originals fordert, starten Sie am besten gleich mit dem Update.

Die Speicherung der Objekte übernehmen in der Standardvorlage die Komponenten *BoldPersistenceHandleDb1* (*TBoldPersistenceHandleDb*), *BoldDataBaseAdapterIB* (*TBoldDataBaseAdapterIB*) und eine IBX-Datenbankkomponente. Die Diagrammansicht der Abbildung 3 verdeutlicht noch einmal die Beziehungen zwischen den Komponenten der Bold-Standardvorlage. Mit den Bold-Komponenten werden einige neue Standardaktionen installiert, von denen drei in der Aktionsliste *ActionList1* der Bold-Standardvorlage zugefügt sind. Der Startupdialog der Anwendung verwendet diese Aktionen um die Datenbank zu erzeugen und das System zu öffnen.

Dieselben Funktionen stehen auch zur Designzeit zur Verfügung. Wollen Sie eine Datenbank zur Designzeit erzeugen und einrichten, tragen Sie zunächst den Namen der Datenbankdatei in der IBX-Datenbankkomponente ein. Anschließend wählen Sie im Popup-Menü des Datenbankadapters den Menüpunkt *Create-Interbase Database File* aus. Dieser Menüpunkt erzeugt allerdings lediglich die

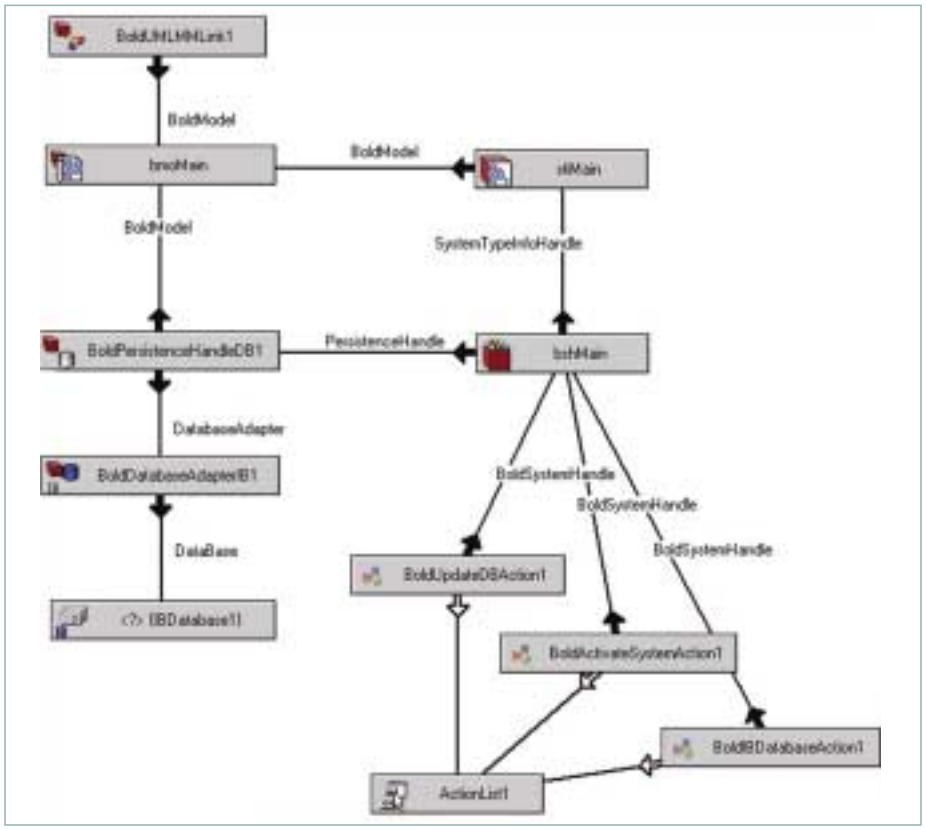


Abb. 3: Die Komponentenverknüpfungen der Standardvorlage

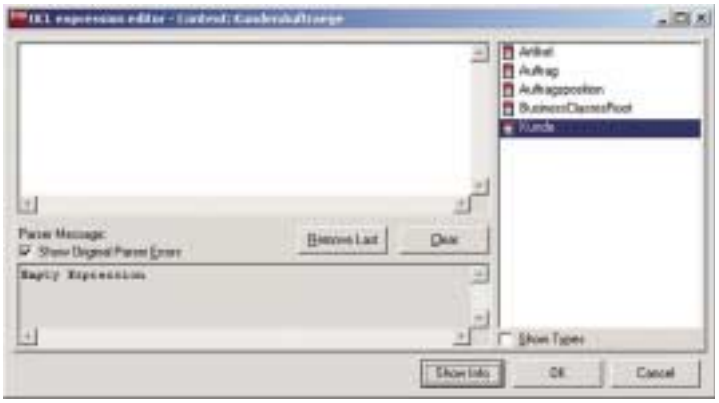


Abb. 4: Der OCL-Eigenschaftseditor

Datei. Über das Popup-Menü der Modellkomponente erzeugen Sie die Tabellenstruktur mit dem Menüpunkt *Generate Database*.

Businessklassen visualisieren

Mit den bisherigen Einstellungen kann die Applikation bereits eine Datenbank zur Speicherung der Klassen erzeugen. Allerdings fehlt noch Code um die Klassen zu verwenden, Instanzen der Klassen anzulegen oder zu verknüpfen. Wie bereits eingangs erwähnt, stellt Bold spezielle Controls für die Visualisierung von Businessobjekten zur Verfügung. Für die erste Anwendung sollen zunächst Grid-Komponenten (*TBoldGrid*) und ein Navigatorkomponenten(*TBoldNavigator*) reichen. Analog der DataSource-Eigenschaft von klassischen datensensitiven Komponenten, besitzen die Komponenten eine Eigenschaft *BoldHandle*, mit deren Hilfe sie an Businessobjekte angebunden werden. Für ein Grid oder einen Navigator ist ein Handle erforderlich, das Zugriff auf eine Kollektion von Objekten bietet. Die geeignete Komponente dafür ist die Klasse *TBoldListHandle*.

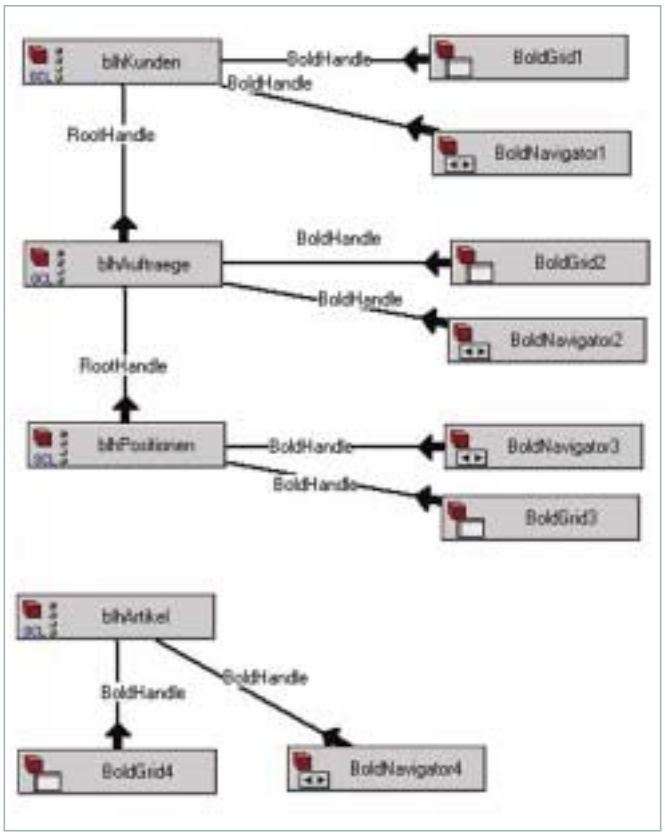


Abb. 5: Die Komponentenverknüpfungen des Hauptformulars

Die Verknüpfung dieser Komponente mit dem Modell der Anwendung erfolgt über die Eigenschaft *RootHandle*. Hier wird das SystemHandle des Datenmoduls *bsiMain* eingetragen. Welche Instanzen in dieser Liste angezeigt werden sollen, muss dem System natürlich auch noch mitgeteilt werden. So wie Sie in einer klassischen Anwendung mit SQL bestimmte Datensätze auswählen, verwenden Sie mit den Bold-Komponenten stattdessen die Sprache OCL (Object Constraint Language), die ein Bestandteil der UML Definition ist, um die Objektauswahl festzulegen.

Die Object Constraint Language ist fast schon eine eigene Programmiersprache welche die Selektion und Navigation von Objekten sowie den Zugriff auf Attribute von Objekten erlaubt. Glücklicherweise erleichtert Bold den Einstieg durch die eingebauten Eigenschaftseditoren für OCL. Für das erste Boldlisthandle, das die Kunden im System anzeigen soll ist der Ausdruck denkbar einfach. In der Eigenschaft Expression des Listhandles wird der Ausdruck *Kunde.allInstances* eingetragen. Welche Ausdrücke möglich und verfügbar sind, zeigt der OCL-Eigenschaftseditor aus Abbildung 4 in einer Listenansicht an. Der OCL-Ausdruck wird bei der Eingabe validiert und der Datentyp der Auswahl sowie gegebenenfalls bei auftretende Fehler bei der Auswertung werden direkt unterhalb des Ausdruckseditors angezeigt. Nachdem die Auswahl in der Eigenschaft Expression eingetragen ist, können Sie im verknüpften Bold-Grid über die rechte Maustaste die Standardspalten für die Auswahl erzeugen lassen.

Für die Anzeige der weiteren Klassen kann wieder eine Kombination von Grid, Navigator und Boldlisthandle verwendet werden. Sollen lediglich die Aufträge des ausgewählten Kunden und die Positionen des ausgewählten Auftrags angezeigt werden, so ist es allerdings sinnvoll die Eigenschaft *Roothandle* bei diesen beiden Handle-Komponenten nicht auf das Systemhandle aus dem Datenmodul zu setzen, sondern auf das Handle der Kundenliste bzw. das Handle der Auftragsliste. Die Ausdrücke für die Eigenschaft *Expression* vereinfachen sich damit zu den

Anzeige

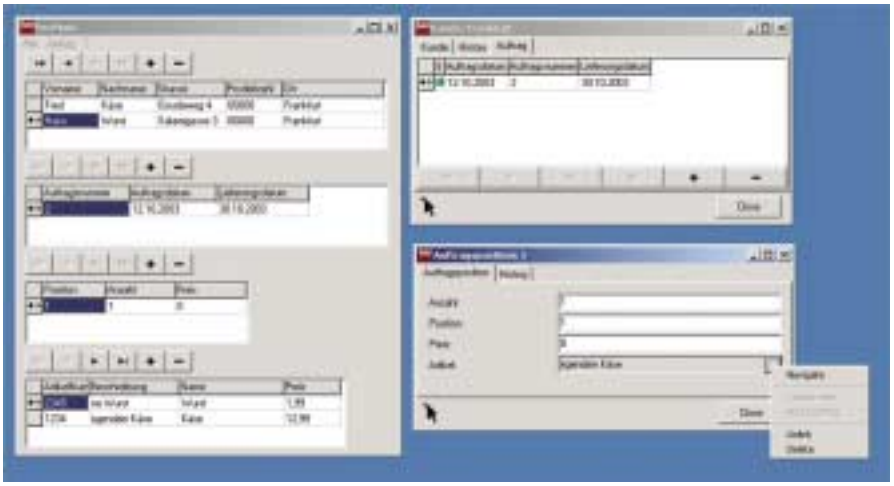


Abb. 6: Anwendung mit Eingabefenstern

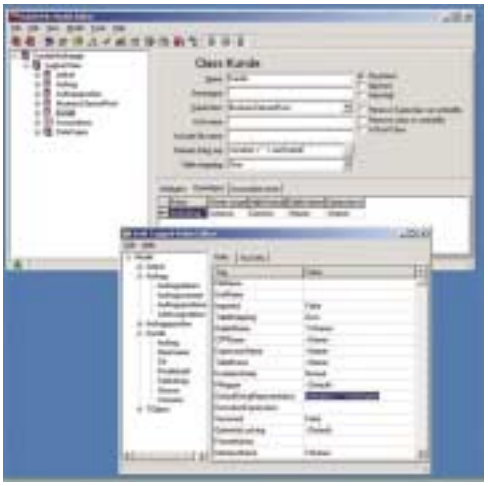


Abb. 7: Bold UML-/Tagged Value-Editor im ModelMaker

Namen der Verknüpfungsattribute *auftrag* und *auftragspositionen* (Abb. 5).

Ohne dass in irgendeiner Form Quelltext für die Klassen erzeugt wird, ist das System jetzt schon lauffähig und die Anwendung kann gestartet werden. Beim ersten Start der Anwendung lassen Sie zunächst die Datenbank durch den Startdialog erzeugen und öffnen das System. Über die Navigatorkomponenten können Sie bereits neue Kunden und Aufträge anlegen, Auftragspositionen und Artikel erfassen und diese via Menü *File | UpdateDB* in die Datenbank speichern. Hier glänzen die Bold-Komponenten durch eine Vielzahl vollautomatischer Features. Durch Doppelklick auf eine Spalte öffnet sich ein neues Eingabefenster für die aktuelle Instanz, das verknüpfte Objekte automatisch mit anzeigt. Sowohl die neuen Eingabefenster als auch die Grid-Komponenten unterstützen von sich aus Drag´n Drop, sodass Aufträge mit der Maus zwischen Kunden verschoben werden können. Die 1:1 Verbindung von Auftragspositionen zu Artikeln führt automatisch zur Einblendung eines Auswahlbutton im Bearbeitungsfenster der Auftragspositionen, sodass von hier aus Artikel angelegt oder ausgewählt werden können (Abb. 6).

Modell und Anwendung

Die Titelzeilen der automatischen Eingabefenster offenbaren, dass die Stringanzeige der Klassen noch überarbeitungsbedürftig ist. Der Ort ist sicherlich als Name für die Klasse ungeeignet. Diese Änderung können Sie im Modell-Editor vornehmen, indem Sie das Feld *Default String Rep* der Klasse mit einem gültigen OCL Ausdruck versorgen wie in Abbildung 7 zu sehen . Im UML-Editor von *TBoldModel* steht Ihnen hierfür der OCL-Editor für die bequeme Eingabe zur Verfügung. Für den Kunden bietet sich der Ausdruck „*vorname* + ' ' + *name*“ an. Neben der Stringrepräsentation lassen sich noch weitere Einstellungen auf Klassen und Attributebene im Modelleditor vornehmen, mit denen Sie unter anderen Spaltennamen, Tabellennamen und Klassennamen modifizieren können.

Die bisherige Funktionalität und Logik der Applikation von der Eingabe bis zur Speicherung wird komplett aus dem Modell heraus betrieben. Ändern sich die Klassen im Modell, weil beispielsweise Attribute hinzugefügt werden, importieren Sie das Modell erneut aus dem Modellierungstool und lassen die Datenbankstruktur von Bold anpassen. Für diese Anpassung stellen die Komponente *TBoldModel* den Menüpunkt *Evolve Database* zur Verfügung. Bei Auswahl dieses Menüpunkts wird die Datenbankstruktur unter Beibehaltung der bestehenden Daten an die Änderungen angepasst. Das einzige was beim aktuellen Stand zusätzlich in der Applikation angepasst werden muss, sind gegebenenfalls zusätzliche Spalten im Grid. Beim Re-Import des Modells gehen allerdings auch alle Änderungen verloren, die Sie im Modelleditor von Bold vorgenommen haben. In diesem Fall zum Beispiel die Stringrepräsentation der Klasse Kunde. Da der Export nach ModelMaker nicht funktioniert, sollten Sie Änderungen im Modelleditor nur dann vornehmen, wenn Sie wissen, dass sich die Applikation nicht mehr ändert (also nie!). Auch wenn es durch fehlende OCL-Editoren etwas unbequemer ist, sollten Sie in jedem Fall so konsequent sein, und diese Informationen im Modellierungstool abgelegen. Um diese Werte UML-konform anzugeben, verwendet Bold sogenannte Tagged Values. Diese können Sie im ModelMaker über das Bold-Plugin editieren, dass Sie im Menü *Tools* unter *Bold tagged value editor* finden.

Wie erwähnt, werden Sie in der Realität kaum ohne zusätzlichen Code in den Klassen auskommen. Bisher enthalten die Beispielklassen nicht einmal Operationen beziehungsweise Methoden. Dennoch können die Bold-Komponenten durch die konsequente Kapselung viel Arbeit und Aufwand beim Refactoring ersparen. Das Ziel bei der Modellierung für Bold ist es, möglichst viel der Anwendungslogik bereits im Modell zu erfassen und nicht als Code zu hinterlegen. In der bestehenden Applikation wurde die OCL-Sprache nur zur Anzeige verwendet. Mit Hilfe der OCL lassen sich aber durchaus auch Eigenschaften oder Beschränkungen von Klassen direkt im Modell definieren, sodass der Anteil an Logik im Code minimiert wird.

Schmuckstück

Ein Blick auf den Microsoft SQL Server 2000 aus der Sicht des Entwicklers

Die Frage „Welche Datenbank ist die Beste?“ wird von einem Entwickler ganz anders beantwortet als von einem Kunden, der diese Datenbank im Alltag verwendet. Denn während der Entwicklungszeit einer Datenbankanwendung werden wir regelmäßig mit Problemen konfrontiert, deren Ursachen sich erst dann zu erkennen geben, wenn wir einen prüfenden Blick unter die Motorhaube werfen. In diesem Fall kann die Datenbank im Entwicklungs-Alltag punkten, die einen gut bestückten Werkzeugkasten mitbringt und die sich gut in das Betriebssystem integriert.

von Andreas Kosch

Wenn man sich die Entstehungsgeschichte dieser Datenbank anschaut, beginnt es auch hier wie so oft im Leben mit dem Satz „Es war einmal“. Im Jahr 1992 hat Microsoft den SQL Server der Firma Sybase lizenziert, um diesen unter dem eigenen Label Microsoft SQL Server zu vermarkten. Allerdings war das Ergebnis bis zur Version 6.5 nicht gerade berauschend – denn die von Sybase lizenzierte Architektur hatte gravierende Nachteile. Zum Beispiel gab es bis zur Version 6.5 keine Datensatzsperren, sondern nur Seiten- und Tabellensperren. Dies hatte zur Folge, dass die Datenbank nur 2 kByte große Datenbankseiten nutzte und auch der vom SQL-Standard vorgesehene Transaktionen-Isolationsgrad *REPEATABLE READ* nicht unterstützt werden konnte (da dieser Modus verheerende Folge auf die Sperren gehabt hätte). Zu dieser Zeit wurde daher dieser SQL Server – zu Recht – von Vielen links liegen gelassen. Microsoft kann aber aufgrund der gut gefüllten Portokasse derartige Probleme auf eine einzigartige Weise lösen. So sagt die Legende, dass eines Tages Bill Gates sein Scheckheft unter den Arm klemmte, um die Top 100 der Datenbank-Branche aus den Konkurrenz-Firmen herauszukaufen (Sie sehen, dass Borland nicht das einzige Opfer dieses Prinzips war). Diese Jungs bekamen ein eigenes Gebäude, scheinbar unbegrenzte Mittel aber keinen strengen Terminplan, sodass sie in Ruhe einen SQL Server von Grund auf neu zusammenbauen konnten, den sie sich schon immer erträumt hatten. Während dieser Zeit wurde der alte SQL Server von der alten Mannschaft weiterentwickelt und als Version 6.0 (Juni 1995) und 6.5 (Juni 1996) ausgeliefert. Erst im November 1998 war es soweit – der auf einer grundlegend neuen Architektur basierende Microsoft SQL Server 7.0 erblickte die Welt und der Source Code der alten Sybase-Fassung 6.5 wurde in die Tonne gehauen. Um zu meinem Beispiel von vorhin zurückzukommen, unterstützt der neue Server nun echte Datensatzsperren, sodass man zu 8 kByte großen Datenbankseiten wechseln konnte und auch alle vom SQL-Standard geforderten Isolationsgrade für Transaktionen verfügbar sind. Mit dem im August 2000 erschienen Microsoft SQL Server 2000 sind einige der noch verbliebenen Lücken (Benutzerdefinierte Funktionen, kaskadierte referenzielle Integrität, INSTEAD OF-Trigger usw.) geschlossen worden. Ein schmucker SQL Server liegt vor, der sich selbstbewusst jedem Vergleich stellen kann.

Welche Gründe gibt es für einen Entwickler, seine Datenbankanwendung so zu konzipieren, dass sie mit dem Microsoft SQL Server 2000 ideal zusammenarbeitet? Aus meiner Sicht gibt es fünf markante Gründe:

- Hoher Marktanteil.
- Hohe Performance.
- Winzige Beschaffungskosten für einen Entwickler.
- Gute Treiber und gute Dokumentation.
- Ausgezeichnete Zukunftsaussichten.

Der erste Grund ist der hohe Marktanteil vom Microsoft SQL Server 2000, denn je höher die Verbreitung, umso größer ist auch die Wahrscheinlichkeit, dass das eigene Programm den „richtigen“ SQL Server voraussetzen kann. Für das Jahr 2002 liegen gleich zwei Zahlen von Gartner Dataquest vor. Während der MS SQL Server im Bereich Windows- und Unix-Datenbanken mit 23 Prozent Marktanteil nach IBM und ORACLE auf den dritten Platz kommt, sieht das in der reinen Windows-Umgebung mit 40 Prozent Marktanteil noch besser aus.

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter
www.derentwickler.de

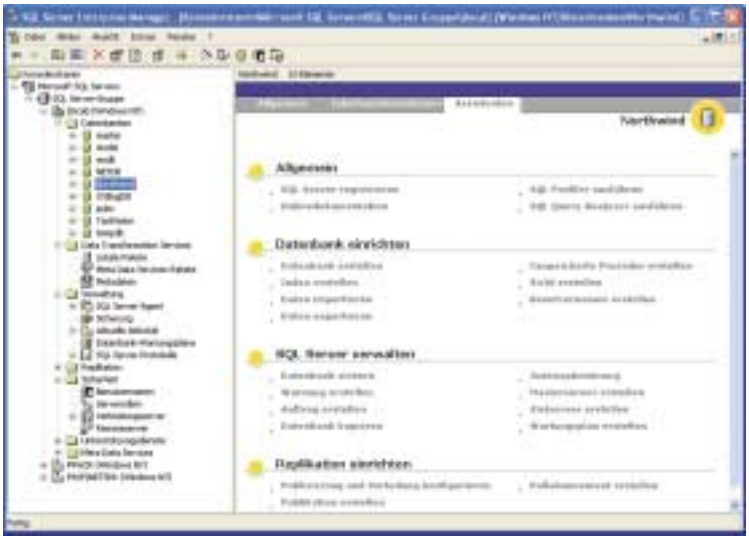


Abb. 1: Der Enterprise Manager kontrolliert alle im Netzwerk erreichbaren MS SQL Server

Der zweite Grund ist die ausgezeichnete Performance des MS SQL Server 2000. Während die Datenbank die Spitzenplätze beim TPC-C Benchmark Test in der Kategorie der Clustered Servern seit Jahren scheinbar geerbt hat, wird dieser Spitzenplatz seit kurzem auch in der Kategorie Non-clustered Server eingenommen. Die Entwicklung ist auch in diesem Punkt beeindruckend – während das Sybase-Original auf 14 000 Transaktionen pro Minute kam, schraubt der MS SQL Server 2000 diese Zahl auf 700 000 hoch, wobei auch die Entwicklung der Hardware einen großen Anteil daran hat. An dieser Stelle muss gesagt werden, dass Microsoft die neue Architektur des SQL Server dem eigenen Betriebssystem auf dem Leib geschneidert hat – wobei aber auch Windows 2000 so erweitert wurde, dass der SQL Server optimal unterstützt wird. Diese Integrationstiefe wurde im Fall von Windows 2003 Server nochmals gesteigert, sodass auch die Datenbankleistung unter dem neuesten Windows erneut ansteigt.

Kommen wir zum dritten Grund – was kostet der Spaß? Für uns als Entwickler wird diese Frage anders beantwortet als für einen Kunden, der die Datenbank in seiner Rechnerumgebung nutzt. Im besten Fall kostet einem Entwickler der Microsoft SQL Server 2000 Developer Edition (der ausgewachsene Server, der sich im Gegensatz zur Standard Edition auch unter den Professional-Versionen von Windows 2000 oder XP installieren lässt) gar nichts. Sowohl im MSDN-Abo als auch beim neuen Borland C#Builder ist dieses Paket bereits enthalten. Borland liefert den C#Builder sowohl mit dem eigenen InterBase 7 als auch mit dem MS SQL Server 2000 aus. Der Grund dafür liegt darin, dass beide SQL-Datenbanken zu unterschiedlichen Gewichtsklassen gehören und somit der MS SQL Server nicht in jedem Fall ein Konkurrent für den InterBase 7 ist. Wenn Sie keine neue Entwicklungsumgebung kaufen möchten, aber trotzdem sparen wollen, müssen Sie warten, bis Microsoft seine Ankündigung wahr macht und den MS SQL Server 2000 Developer Edition für den Preis von 49,-USD verkauft. Aber auch der Endkunde ist vom Preis her gesehen gut bedient – denn der MS SQL Server ist bezogen auf die Transaktionsleistung die preisgünstigste Alternative. Da er so schnell ist, kann die Hardware im Vergleich zu Mitbewerbern auch eine Nummer kleiner gekauft werden, sodass in einer typischen Umgebung nur 20 bis 50 Prozent der sonst üblichen Kosten (Hardware und Software) anfallen. Außerdem gibt es ja auch noch die kostenfrei mit der eigenen Anwendung verteilbare MSDE 2.0 (Microsoft SQL Server 2000 Desktop Engine), die immer dann ausreicht, wenn weniger als 5 gleichzeitige Anwender auf eine Datenbank kleiner als 2 GByte zugreifen.

Zum vierten Grund muss ich Ihnen sicherlich nichts mehr erzählen – jeder Entwickler hat schon selbst die Erfahrung gemacht, welche entscheidenden Auswirkungen der Datenbank-Treiber sowie die vorliegende Dokumentation des SQL Servers und des Treibers auf das Gelingen eines Projekts haben. Gerade bei diesem Punkt zeigte zum Beispiel der Borland InterBase bisher eine Schwäche.

Der fünfte und letzte Grund in meiner Auflistung betrifft die ausgezeichneten Zukunftsaussichten. Der unter dem Codenamen „Yukon“ angekündigte Nachfolger des MS SQL Server 2000 soll im Jahr 2004 auch offiziell das Licht der Welt erblicken. Bis dahin nehmen zur Zeit im Beta-Programm rund 1000 Partnerunternehmen diesen neuen Server unter die Lupe – allein diese Zahl und der lange Zeitraum belegen bereits, das erneut grundlegende Änderungen anstehen. Denn mit Yukon kommt das Thema .NET auch auf die Agenda von Datenbank-Administratoren. Zusätzlich zu T-SQL (hinter Transact-SQL verbirgt sich eine Erweiterung des Sprachumfangs gegenüber dem SQL Standard) und den sonstigen Erweiterungen wird der neue SQL Server jede .NET-Sprache innerhalb einer Stored Procedure oder eines Triggers unterstützen. Somit stellt der neue SQL Server auch im Punkt der Programmierbarkeit alle anderen Konkurrenten in den Schatten, zumal mit .NET im Gegensatz zu Java kein Leistungseinbruch verbunden ist.

Schauen wir uns nun an, welche Tools in der Werkzeugkiste enthalten sind. Al-

Listing 1

```
type
 TDTSCopyObj = class(TAutoObject, IDTSCopyObj,
                                        CustomTask)
 private
  FName     : WideString;
  FDescription : WideString;
 protected
  function Get_Description: WideString; safecall;
  function Get_Name: WideString; safecall;
  function Get_Properties: Properties; safecall;
  procedure Execute(const pPackage,
   pPackageEvents, pPackageLog: IDispatch;
   var pTaskResult: DTSTaskExecResult); safecall;
  procedure Set_Description(
   const pRetVal: WideString); safecall;
  procedure Set_Name(
   const pRetVal: WideString); safecall;
 public
  procedure Initialize; override;
 end;
```

lerdings ist das nur eine Seite der Medaille, denn Microsoft hat selbstverständlich die Chancen genutzt, die sich allein daraus ergeben, dass die Einzelteile MS SQL Server 2000, MDAC (OLE DB), TCP/IP-Implementierung und Betriebssystem aus einer Hand stammen. Neben der Werkzeugen des Servers gibt es somit auch auf der Betriebssystem-Ebene einige Ansatzpunkte.

Enterprise Manager

Die Schaltzentrale des MS SQL Server 2000 verbirgt sich hinter dem Enterprise Manager. Mit diesem Tool übernehmen Sie – vorausgesetzt, die notwendigen Berechtigungen liegen vor – die Kontrolle über alle im Netzwerk erreichbaren MS SQL Server. In der Abbildung 1 ist im linken *TreeView*-Fenser zu sehen, dass Sie den MS SQL Server auch mehrfach auf einem Rechner installieren können – in meinem Fall ist auf dem lokalen Rechner sowohl der MS SQL Server 2000 Developer Edition als auch die vom .NET Framework SDK ins Spiel gebrachte MSDE 2.0 installiert. Damit sich beide Instanzen voneinander unterscheiden, erhält die zweite Version einen anderen Namen (in meinem Beispiel *(local))\NetSDK*).

Die Abbildung 1 zeigt auch, dass Microsoft beim Enterprise Manager sehr viel Wert auf benutzerfreundliche Experten und Assistenten gelegt hat. Als Beispiel möchte ich hier den Indexoptimierungs-Assistenten nennen – dieser Experte ist selbst in der Lage, die von der Anwendung ausgelösten SQL-Anweisungen zu analysieren und daraus eine Empfehlung für die Umstrukturierung der aktiven Indizes zu generieren. Auf Wunsch werden diese Vorschläge auch in die Datenbank eingearbeitet. Der Vorteil dieses Assistenten liegt auf der Hand – da keine theoretischen Betrachtungen zugrunde gelegt werden, sondern die tatsächlich vom eigenen Programm abgeschickten SQL-Anweisungen mit Berücksichtigung der Häufigkeit die Datenbasis bilden, wird ein Optimierungsergebnis erreicht, dass ohne Assistenten nur mit großem Aufwand verbunden wäre (zumal der Entwickler beim manuellen Optimieren die internen Details der SQL-Datenbank sehr gut kennen muss).

Der Enterprise Manager ist aber auch eine visuelle Entwicklungsumgebung – wenn auch der besonderen Art. Ich meine hier nicht die bestehende Möglichkeit, Stored Procedures oder Trigger anzulegen und dort T-SQL-Anweisungen unterzubringen, sondern die Entwicklungen von Paketen der Data Transformation Services (DTS). Der Enterprise Manager stellt dafür ein visuelles Baukasten-System zur Verfügung, mit dem komplette Ablaufbäume zusammengeklickt werden können. Dieser Baukasten enthält in der Standardinstallation gleich 28 Komponenten – der Clou liegt jedoch darin, dass wir selbst weitere Komponenten hinzufügen dürfen. Diese Komponenten werden als COM-Objekte erwartet, die neben dem *IDispatch*-Interface auch das Interface *CustomTask* implementieren. Listing 1 zeigt den Aufbau eines derartigen COM-Objekts, wobei die eigene Nutzfunktion in der Implementierung der Interface-Methode *Execute* untergebracht wird.

Wie die Abbildung 2 zeigt, wird das in Form einer DLL vorliegende COM-Objekt einmalig im Enterprise Manager als benutzerdefiniertes Task-Objekt registriert. Wenn das erledigt ist, steht in der Komponenten-Palette eine weitere Komponente einsatzbereit zur Verfügung. Aus Platzgründen beschränke ich mich bei meinem Beispiel auf ein einfaches DTS-Paket, das nur ein Backup der Datenbank macht und im Erfolgsfall die Backupdatei auch auf einen anderen Server kopiert. Die tatsächlichen Pfadnamen sind dabei nicht fest im eigenen COM-Objekt codiert, sondern werden über globale Paket-Eigenschaften erst zur Laufzeit ausgelesen. Die Interface-Methode *Execute* erhält dazu beim Aufruf im Parameter *pPackage* einen Interface-Zeiger, der direkt in den Typ *_Package2* eingetauscht werden kann. Und dort ist die Interface-Methode *Get_Value* verfügbar, um den aktuellen Inhalt der gesuchten globalen Variable auszulesen. Somit muss das eigene COM-Objekt nur die Zeichenkette mit dem Namen der Variable fest codieren, aber nicht deren Wert:

```
var
 aP2 :_Package2;
 aGV :GlobalVariable;
```

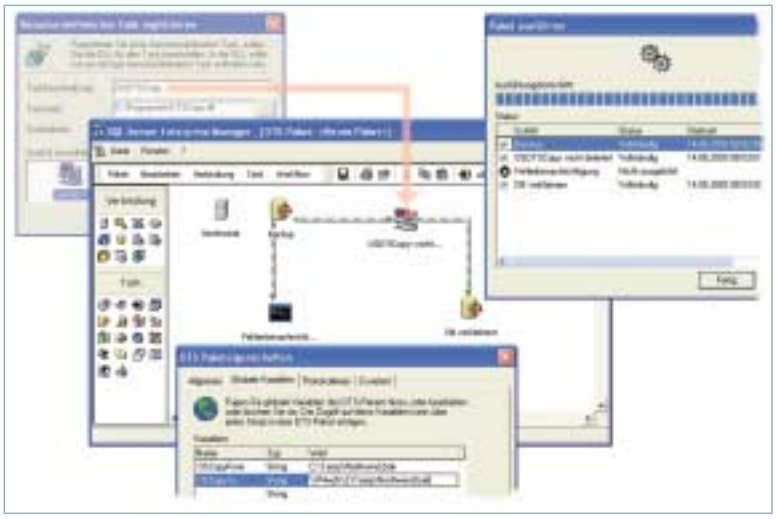


Abb. 2: Das DTS-Paket in Aktion

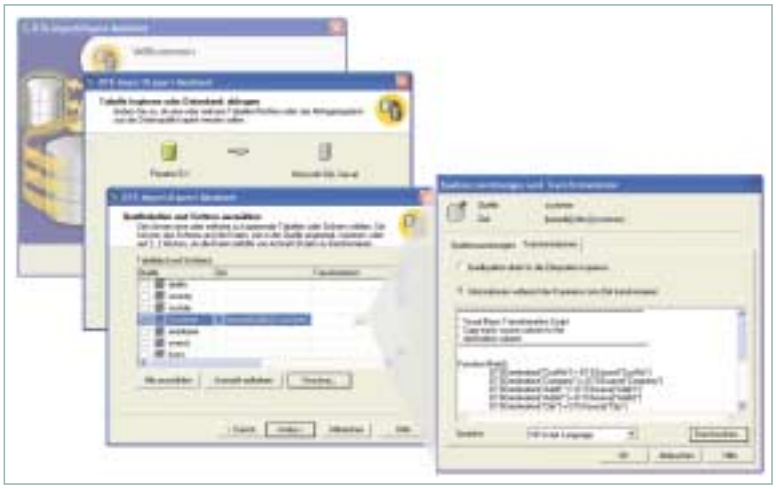


Abb. 3: Die Paradox-Daten landen in der MS SQL Server-Datenbank

```
 sFrom : String;
begin
 aP2 := pPackage as _Package2;
 aGV := aP2.GlobalVariables.Item(cFROM);
 sFrom := aGV.Get_Value;
 ...
```

Wenn das Backup fehlschlägt, arbeitet das DTS-Paket den Fehlerfall ab. In meinem Beispiel auf der *Entwickler* Profi CD habe ich das Paket direkt aus dem Enterprise Manager heraus von Hand gestartet, Sie können diesen Job jedoch auch zeitgesteuert vom MS SQL Server-Agent ausführen lassen, von der Kommandozeile (*dtsrun.exe*) aus starten oder das DTS-Paket über COM-Aufrufe aus der eigenen Delphi-Anwendung heraus ausführen.

Was die Abbildung 2 nicht zeigt, ist die Option im DTS-Paket eigene JScript- oder VBScript-Anweisungen unterzubringen, wobei der Enterprise Manager dazu sogar einen visuellen Funktionsgenerator zur Verfügung stellt. Dies wird immer dann zu einem gravierenden Vorteil, wenn regelmäßige Datenimporte oder Datenexperte inklusive dazu notwendiger Anpassungen auf Ihrer Agenda stehen.

DTS-Import/Export-Assistent

Allerdings ist der Aufwand des DTS-Pakets zu hoch, wenn es nur um einen einmaligen Datenimport oder Datenexport geht. In diesem Fall ist der direkte Weg über den DTS-Import/Export-Assistenten besser geeignet, wobei die Abbildung 3 auch den Grund dafür zeigt. Über die verschiedenen Dialogseiten dieses Assistenten wählen Sie die Quell- und Zieldatenbank aus (wobei alles das ausgewählt werden kann, wofür ein OLE DB-Provider oder ein ODBC-Treiber zur Verfügung steht). In meinem Beispiel kopiere ich die Daten von der Delphi-Beispieldatenbank *customer.db* in die MS SQL Server-Datenbank *tempdb*. Immer dann, wenn Sie mit dem automatischen Vorschlag für den Datentyp nicht zufrieden sind, können Sie von Hand eingreifen und diesen ändern. Die Abbildung 3 zeigt allerdings auch, dass eine Transformation der Daten selbst unterstützt wird, indem eine Anpassung beziehungsweise Umstrukturierung über die unterstützten Scriptsprachen erfolgt.

Das Ergebnis des Assistenten können Sie dann entweder sofort von Hand ausführen, zu einem bestimmten Termin automatisch ausführen oder gar in Gestalt eines DTS-Pakets direkt in der Datenbank ablegen lassen.

SQL Query Analyzer

Wie der Gattungsname SQL Server schon sagt, spielt das Hantieren mit SQL-Anweisungen im Alltagsleben eines Entwicklers eine sehr große Rolle. Zwar stellen die Entwicklungsumgebungen wie zum Beispiel Borland Delphi, Borland C#Builder oder Microsoft Visual Studio .NET eigene Experten zur Verfügung, die sich hinter den Kulissen automatisch um das Generieren von SQL-Anweisungen kümmern, aber dies dient nur dem Komfort. Ein vollwertiger Ersatz für den SQL Query Analyzer – dem Werkzeug für die interaktive Kommunikation mit der SQL- Datenbank – sind diese Experten nicht. Der SQL Query Analyzer lässt keine Wünsche offen, die Abbildung 4 zeigt nur einen winzigen Ausschnitt aus der unterstützten Funktionalität. Über spezielle Vorlagen können Sie zum Beispiel das Grundgerüst von Datenbankobjekten wie zum Beispiel einer Stored Procedure über Drag´n Drop in das Editorfenster ziehen. Über einen speziellen Dialog lassen Sie dann alle dort untergebrachten Platzhalter durch die eigenen Bezeichnernamen ersetzen, sodass am Ende das fertige Grundgerüst vorliegt. Selbstverständlich kann diese gespeicherte Prozedur dann innerhalb des Query Analyzer auch im Debugger schrittweise ausgeführt werden, um die Wert der Parameter oder der lokalen Variablen zu untersuchen.

SQL Profiler

Immer dann, wenn in der eigenen Anwendung irgendwelche Objekte im Hintergrund automatisch SQL-Anweisungen generieren und zur Datenbank schicken, ist ein Werkzeug sehr hilfreich, dass einen Blick hinter die Kulissen erlaubt. Was ich neutral als Objekt bezeichnet habe, kann auch konkret mit dem Namen *Recordset*-Objekt (ADO), *TADODataSet* (ADO Express alias dbGo) oder *TSQLDataSet* (dbExpress) genannt werden. Das Gleiche gilt, wenn der tatsächlich an die Datenbank übergebene Wert eines Parameters untersucht werden soll. Damit das alles nicht so theoretisch bleibt, stelle ich Ihnen ein Beispiel aus der Praxis vor. Angenommen, Sie entwickeln mit dem nagelneuen Borland C#Builder eine Datenbankanwendung für den MS SQL Server 2000, wobei Sie auf die *BDP.NET*-Komponenten (Borland Data Provider for .NET) zurückgreifen. Damit das Beispiel so übersichtlich wie nur möglich bleibt, enthält meine Tabelle nur drei Spalten:

```
CREATE TABLE Vergleich (
 RecID    INTEGER      NOT NULL IDENTITY PRIMARY KEY,
 Wert     VARCHAR(10)  NOT NULL,
 Datum    DATETIME     NOT NULL )
```

Die Spalte *RecID* wird als Primärschlüssel gekennzeichnet, wobei das Schlüsselwort *IDENTITY* festlegt, dass nur der MS SQL Server selbst den *INTEGER*-Wert für dieses Feld festlegt. Der MS SQL Server 2000 unterstützt zwei Datentypen für die Speicherung von Datum- und Uhrzeitwerten: *DATETIME* und *SMALLDATETIME*. Der Datentyp *DATETIME* speichert den Bereich vom 1.1.1753 bis 31.12.9999 mit einer Genauigkeit von 3,33 Millisekunden in Form von 8 Bytes (2 x 4-Byte Integer). Die ersten 4 Bytes entsprechen der Anzahl von Tagen seit dem Basisdatum und die zweiten 4 Bytes speichern die Anzahl von Windows-Ticks, die seit Mitternacht vergangen sind (eine Sekunde besteht aus 300 Windows-Ticks, so ein Tick ca. 3,3 ms entspricht). Der Experte vom C#Builder hat für das Aktualisieren eines Datensatzes die folgende *UPDATE*-Anweisung generiert, wobei die Fragezeichen als Platzhalter für die erst zur Laufzeit zugewiesenen Parameter-Werte genutzt werden:

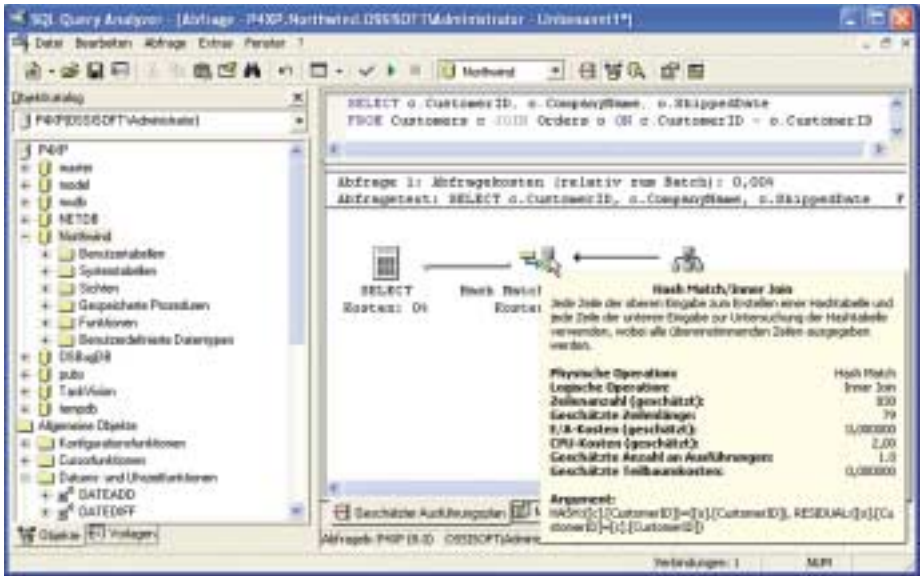


Abb. 4: Die Effektivität einer SQL-Abfrage wird untersucht

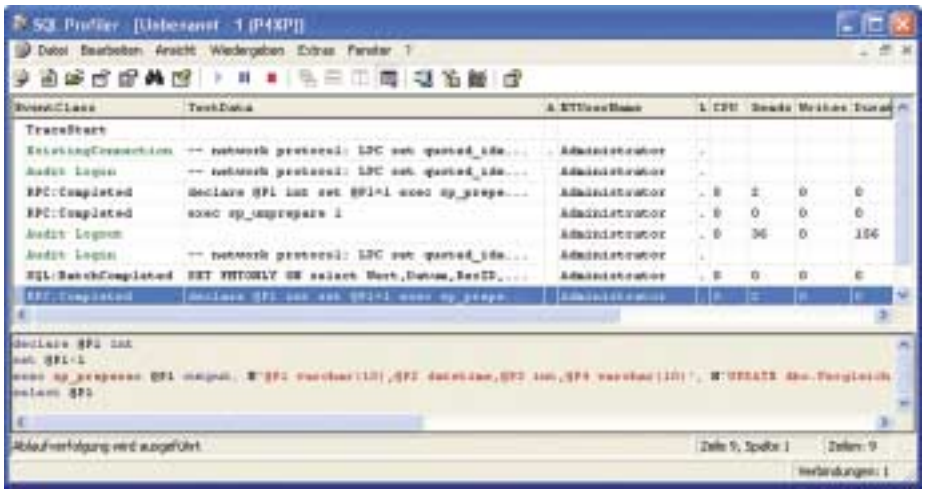


Abb. 5: Der SQL Profiler zeigt die Ursache für dieses Problem an

```
UPDATE dbo.Vergleich
 SET Wert = ?,
 Datum = ?
WHERE RecID = ?
 AND Wert = ?
 AND Datum = ?
```

Sowohl die Tabelle als auch die *UPDATE*-Anweisung sehen so harmlos aus, das niemand mit einem Problem rechnen würde. Allerdings nutzt BDP.NET intern datenbankunabhängige Datentypen (Prinzip des kleinsten gemeinsamen Nenners) und im Fall der Spalte *Datum* wird der native MS SQL Server-Datentyp *DATETIME* von *BDP.NET* durch den eigenen *BdpType*-Datentyp *DateTime* nachgebildet. Zur Laufzeit ist eine Konvertierung unumgänglich – und jede Konvertierung ist eine potenzielle Quelle von unerwünschten Rundungsfehlern. Dies trifft auch in unserem Fall zu, denn jeder Update-Versuch des Datensatzes provoziert trotz korrekt gefüllter Parameters-Kollektion eine Exception mit dem Fehlertext „Concurrency violation: the Updatecommand affected 0 records“. Der *BdpDataAdapter* hat erkannt, dass von der abgeschickten *UPDATE*-Anweisung kein einziger Da-

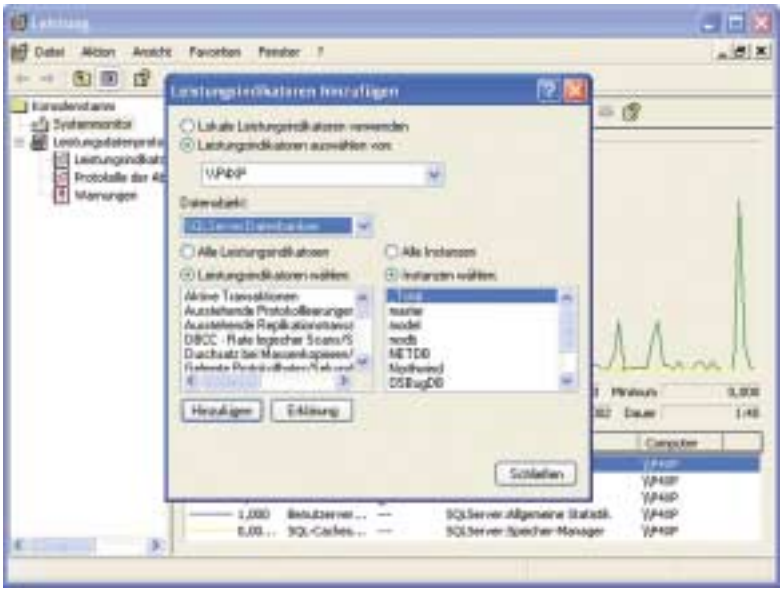


Abb. 6: Die Leistungsanzeige von Windows überwacht den Ressourcenverbrauch

tensatz betroffen war, weil der MS SQL Server keinen zum *WHERE*-Kriterium passenden Datensatz gefunden hat. Normalerweise passiert so etwas, wenn in der Zwischenzeit ein anderer Benutzer diesen Datensatz geändert hat. Die *Exception*-Meldung bezieht sich auf diesen Standardfall und spricht daher von einem Mehrbenutzer-Konflikt. Erst dann, wenn der Entwickler den SQL Profiler des MS SQL Server auf die Lauer legt und die tatsächlich übergebenen Werte anzeigen lässt, wird der tatsächliche Grund für dieses Problem gefunden (Abb. 5).

Wie der folgende Auszug aus dem mitprotokollierten Log zeigt, wird das Datum von *BDP.NET* als Zeichenkette übergeben, wobei die Anzahl der Millisekunden generell mit dem Wert 000 belegt wird:

```
declare @P1 int
set @P1=1
exec sp_prepexec @P1 output, N'@P1 varchar(10),
                                        @P2 datetime,
 @P3 int,@P4 varchar(10),@P5 datetime',
                                  N'UPDATE dbo.Vergleich
  SET Wert = @P1, Datum = @P2
 WHERE RecID = @P3 AND Wert = @P4 AND Datum = @P5',
 'NeuX 2', 'Mai 30 2003  4:33:07:000PM', 2, 'Neu 2',
 'Mai 30 2003  4:33:07:000PM'
select @P1
```

Greife ich dann aber zu einem anderen Werkzeug des MS SQL Server 2000 – in diesem Fall dem SQL Query Analyzer – so wird sichtbar, dass die Datenbanktabelle aber den Millisekundenwert speichert. Somit greift der falsche Parameter-Wert bei der Suche in der Datenmenge ins Leere, sodass kein einziger zu aktualisierender Datensatz gefunden wird. Um nun diesen Datensatz erfolgreich speichern zu können, habe ich nur zwei Alternativen. Entweder nutze ich für den Datumswert eine unscharfe *WHERE*-Bedingung, die über zwei Parameter mit größer als und kleiner als ein Zeitfenster von einer Sekunde für die Suche nach dem zu aktualisierenden Datensatz verwendet, oder ich entferne die Datums-Spalte aus dem *WHERE*-Kriterium.

Integration in das Betriebssystem

Da der SQL Server aus dem gleichen Hause stammt wie das Betriebssystem auch, verwundert es nicht, wie tief der SQL Server in Windows integriert ist. Das wird zum Beispiel bei der Leistungsanzeige (Abb. 6) von Windows sichtbar. Der SQL Server registriert bei seiner Installation eine Vielzahl von Leistungsindikatoren, sodass alle kritischen Ressourcen überwacht werden können. Als Praxisbeispiel nenne ich hier nur die Anzahl der aktiven Datenbankverbindungen – da sowohl bei OLE DB als auch bei .NET im Hintergrund ein automatischer Datenbankverbindungs-Pool werkelt, ist die tatsächliche Anzahl nicht direkt aus dem eigenen Source Code ablesbar.

Als Abschluss meines Streifzuges – bei dem noch längst nicht alles vorgestellt wurde – möchte ich nur eine weitere Fähigkeit des Servers ansprechen. Auch unmittelbar aus T-SQL heraus (wie zum Beispiel einer Stored Procedure) kann ein eigenes COM-Objekt beziehungsweise .NET-Objekt aufgerufen werden, wobei in diesem Fall dieses Objekt noch nicht einmal ein spezielles Interface implementieren muss. Falls Sie im Funktionsumfang von T-SQL etwas vermissen, kann das fehlende also auch mit der eigenen Entwicklungsumgebung nachgerüstet werden.

Resümee

Falls Sie selbst noch keine Erfahrungen mit dem Microsoft SQL Server 2000 gesammelt haben, hat dieser Artikel einen Überblick über diesen Server gegeben. Nun wird auch klar, warum Borland das Konkurrenzprodukt zusammen mit dem C#Builder ausliefert – denn der eigene InterBase 7 ist einige Nummern kleiner. Aber auch hier gilt die Binsenweisheit, dass eine Konfektionsgröße nicht für alle Menschen passt – es ist unsere Aufgabe als Entwickler, die im konkreten Anwendungsfall „richtige“ SQL Datenbank auszuwählen.

Microsoft SQL Server 2000 Developer Edition

Der SQL Server wird von Microsoft in verschiedenen Ausführungen ausgeliefert. Für uns als Entwickler sind allerdings nur zwei davon von Interesse. Zum einen die mit gewissen Einschränkungen kostenlos verteilbare Microsoft SQL Server 2000 Desktop Engine (MSDE 2.0) und zum anderen die Microsoft SQL Server 2000 Developer Edition. Während die kostenlose MSDE 2.0 auf alle visuellen Tools verzichtet und nur über die Kommandozeilen-Ebene oder aus der eigenen Anwendung heraus konfiguriert werden kann, ist die Developer Edition ein vollwertiger Server. Der Unterschied zwischen der Standard Edition und der Developer Edition besteht darin, dass sich die Standard Edition nur auf einer Server-Version von Windows installieren lässt, aber die Developer Edition auch auf den Professional-Versionen von Windows 2000 und XP.

Unbreakable

Hochverfügbarkeit von Datenbanken

Nicht nur für Aktienhandelssysteme und Online Shopping-Unternehmen gewinnt das Thema Hochverfügbarkeit zunehmend an Gewicht. Ausfallzeiten eines unternehmenskritischen Systems verursachen unter Umständen horrende Kosten. Doch die Realisierung von Ausfallsicherheit ist technisch komplex und oft ein teurer Spaß. Wir greifen einige Konzepte auf und betrachten Vor- und Nachteile der jeweiligen Lösung.

von Andrea Held

Die diesjährige Security-Studie der InformationWeek ergab, dass nur 26 Prozent der rund 4900 befragten Unternehmen im vergangenen Jahr keinen Systemausfall zu verzeichnen hatten. Man kann sich vorstellen, was es für ein Kaufhaus bedeutet, an einem vorweihnachtlichen Samstag einen mehrstündigen Ausfall eines zentralen Datenbankservers und damit des gesamten Kassensystems erdulden zu müssen. De facto kann man den Laden auch direkt schließen, da ein Verkauf der Ware ohne Abwicklung an der Kasse kaum möglich ist. Ähnlich verhält es sich bei Aktienhandels- oder Online Flugbuchungssystemen, bei TV-basiertem Home-Shopping und Anbietern von Katalogwaren, die online bestellt werden können. Doch was kann man tun, um Systemausfällen vorzubeugen?

Hochverfügbarkeit bezeichnet die Fähigkeit eines Systems bei Ausfall einer seiner Komponenten einen uneingeschränkten Betrieb zu gewährleisten. Verfügbarkeit, das bedeutet unterbrechungsfreie Stromversorgung genauso wie ein regelmäßiges Backup, redundante Hardware und ausgeklügelte Speichersysteme, der Einsatz von Standby-Systemen und Software, die sich fehlertolerant verhält. Es gibt keine Pauschallösungen. Jeder Einzelfall muss auf seine Anforderungen hin untersucht werden, um eine adäquates Ergebnis zu erzielen.

Das Herz der Matrix

Kommerzielle Systeme arbeiten in aller Regel mit großen Datenmengen. Ohne ein relationales Datenbankensystem im Backend geht da meist nichts: Datenbanken sind das Herz kommerzieller Systeme. In verteilten Systemen sind Datenbanken eine der kritischsten Komponenten. Kontinuierliche Verfügbarkeit des Gesamtsystems bedarf daher auch einer Strategie der Umsetzung von Verfügbarkeit im Backend.

Im Dezember erscheint das neue Oracle 10g, dessen Grid Architektur den Oracle Real Application Cluster wesentlich erweitern wird. Doch auch andere Hersteller bieten interessante Möglichkeiten, Systemausfällen im Backend vorzubeugen. Geo-Spiegelung, Aktiv/Passiv-Cluster und Replikation des Transaktionslogs sind nur einige Beispiele.

Physischen Fehler wie der Ausfall einer Harddisk, einer Netzwerkkarte aber auch eines gesamten Rechners wird mittels Redundanz entgegen getreten. Durch Geo-Spiegelung kann der Ausfall eines gesamten Standortes abgefangen werden. Replikationsmechanismen, die Zeitverzögerung nutzen, können im Fall eines logischen Fehlers eine große Hilfe sein. Architekturen und Konzepte zur Sicherung von Verfügbarkeit sind alles andere als trivial. Bei der Konzeptionierung müssen zum Teil recht unterschiedliche Szenarien in Betracht gezogen werden, die auch in unterschiedlicher Weise abgefangen werden müssen.

Systemausfälle kategorisieren

Generell unterscheidet man zwischen geplanten und ungeplanten Downtimes. Geplante Downtimes sind vorgesehene Wartungsfenster, in denen Upgrades und Patches eingespielt, Backups gezogen und Daten reorganisiert werden. Ungeplante Downtimes dagegen können unvorhersehbar durch recht unterschiedliche Auslöser verursacht werden. Eine der häufigsten Ursachen für einen Systemausfall sind physische Hardwarefehler: Der Ausfall von Festplatten, Netzwerkkarten oder Net-Segmenten.

Solche physischen Hardwarefehler können durch Redundanz, also mehrfache Auslegung der Komponenten abgefangen werden. EMC-Speicherboxen bieten beispielsweise die Möglichkeit von Hot Spare Disks. Diese springen bei Ausfall einer Platte ein und rekonstruieren bzw. übernehmen – meist durch Raid 1, 3 oder 5 gedeckt – deren Daten. Neben Hardware-Raid kann auch Software-Raid bzw. Disk Mirroring eingesetzt werden. Veritas Volume Manager oder Suns Volume Manager bieten hier erweitertes Disk Management mit Spiegelungsfunktionen bis hin zu Geo-Spiegeln.

Doch der Datenerhalt einer Datenbank nutzt wenig, sofern der Datenbankserver ausfällt. Hier gibt es einige zum Teil recht unterschiedliche Konzepte, die ihre jeweils

Unternehmen	Kosten
Brokerage houses	$6.4 million
Credit card sales/ authorization	$2.6 million
Media/ Pay-per-view	$2.6 million
Home shopping (TV)	$113,000
Catalog sales	$90,000
Airline Reservations	$89,000
Package shipping	$28,000

Tab. 1: Kosten für eine Stunde Ausfallzeit, Angaben der „Contingency Planning Research", 1999

Verfügbar in Prozent	Downtime im Jahr		
	Tage	Stunden	Minuten
95%	18	6	0
99%	3	15	36
99,9%	0	8	46
99,99%	0	0	53
99,999%	0	0	5
99.9999%	0	0	<1

Tab. 2: Jährliche Ausfallzeit nach Verfügbarkeit

eigenen Vor- und Nachteile aufweisen: Replikation, Cluster Systeme und Grids.

Cluster und Grid Computing

Gerade Cluster erfreuen sich zunehmender Beliebtheit. Der Nachfolger des Oracle 9i Real Application Clusters erscheint voraussichtlich im Dezember und das unter dem Namen 10g. Das „g“ steht für Grid und soll nach „i” wie Internet und „e” wie eBusiness einen neuen Hype einleiten, der im Gegensatz zu seinen Vorgängern vielleicht auch halten wird, was er verspricht. Oracle verfolgt dabei weiterhin gezielt seine bereits mit dem OPS (Oracle Parallel Server) eingeführte Aktiv/Aktiv-Architektur auf Shared Disk. Fällt ein Datenbankknoten im Grid aus, springt einer der verbleibenden Knoten ein und übernimmt für den Benutzer transparent (im Sinne von „nicht sichtbar”) dessen Anfragen. Dabei kommt TAF (Transparent Application Failover), ein recht eleganter Failover-Mechanismus zum Einsatz. Dieser sorgt dafür, dass bei einem Ausfall des Datenbankservers lesende Zugriffe durch einen anderen Server komplett übernommen werden. Werden von einem Benutzer also gerade Daten abgefragt und auf dem Bildschirm ausgegeben, so bleibt für einen kurzen Moment die Ausgabe stehen, und zwar solange, bis eine andere Datenbankinstanz im laufenden Betrieb übernommen hat und die Daten weiter ausgibt. Ein solcher Mechanismus innerhalb eines Parallel Cluster Systems wird zur Zeit von (noch) keinem anderen Datenbankhersteller geboten.

Failover Cluster dagegen nutzen Aktiv/Passiv-Architekturen. Dabei ist zu einer Zeit nur ein Knoten aktiv. Der andere steht als Standby zur Verfügung und springt nur ein, wenn der primäre Knoten ausfällt. DB2 UDB bietet genau wie Oracle, Sybase, Informix und SAP DB in Kombination mit einer Clusterware solche Failover-Konstellationen an. Die Konfiguration ist im Vergleich zum Oracle Real Application Cluster wesentlich einfacher. Jedoch benötigt der Failover auch in der Regel etwas länger, da die Datenbank bei einem Systemausfall vom Standby-System erst noch gemountet werden muss.

Unterstützt werden Aktiv/Passiv-Architekturen meist durch Drittthersteller. So bietet der Sun Cluster 3.x Data Services für IBM DB2, Informix, Oracle und Sybase. HACMP für AIX (High Availability Cluster Multiprocessing) auf IBM Maschinen, HP Serviceguard und der Microsoft Cluster bieten ähnliche Funktionalitäten. Nicht zu vergessen: Der Veritas Cluster, der für alle gängigen Plattformen verfügbar bist.

Das Prinzip, nach dem die Clusterware arbeitet, ähnelt sich bei allen Herstellern. Die am Cluster beteiligten Rechner kommunizieren untereinander in der Regel über einen *private Interconnect*, auch *private Network* genannt. Hierbei handelt es sich um ein Netz, das für die Kommunikation zwischen den am Cluster beteiligten Rechnern reserviert ist und auf das von außen sichtbar ist. Über ein zusätzliches *public Network* kann von außen auf den Cluster zugegriffen werden. Jeder Rechner hat also mindestens zwei Netzwerkkarten: Je eine für das private und eine für das öffentliche Netz. In der Praxis legt man natürlich auch hier die Hardware redundant aus und verwendet je Rechnerknoten zwei Netzwerkkarten für das private und zwei weitere für das public Network.

Über das private Network wird unter anderem der *Cluster Heartbeat* gesandt: Jeder der beteiligten Cluster-Rechner zeigt hierüber an, dass er noch am Leben ist. Fällt der Heartbeat des aktiven Systems aus, geht das Standby-System davon aus, übernehmen zu müssen. Um Anwendern und Entwicklern zu ersparen, sich mit technischen Details des Clusters auseinandersetzen zu müssen, werden auch hier verschiedene Eigenschaften von der Clusterware transparent gehandhabt. So werden Dienste wie Datenbankservices, aber auch Ressourcen wie logische IP-Adressen und logische Hostnamen im Fall eines Failovers vom Standby-System automatisiert übernommen. Der Anwender connectet sich nach der Übernahme einfach wie gewohnt mit demselben Hostnamen bzw. mit derselben logischen IP-Adresse zum Server und das Standby-System antwortet, ohne dass der Ausfall des Primär Systems nach außen hin sichtbar wäre.

Problematisch ist der Ausfall des private Networks. Da in diesem Fall der Cluster Heartbeat verstummt, geht jeder der Rechner in dieser als *Split Brain* bezeichneten Situation davon aus, das einzig überlebende System zu sein. Die Folge: Jeder der Cluster-Rechner glaubt, übernehmen zu müs-

Terminologie

Cluster: Zwei oder mehr Systeme, die im Verbund Systemressourcen, Daten und Applikationen zur Verfügung stellen.
Cluster Node: Ein einzelner Rechner, der Teil eines Cluster ist. Auch als Cluster-Knoten oder physical Host bezeichnet.
Data Service: Eine Applikation, die Client Services innerhalb eines Cluster zur Verfügung stellt.
Disk Group: Eine Gruppe von Disk Ressourcen in einer Multihost Umgebung
Fault Monitor: Daemon zur Überprüfung der Funktionsfähigkeit eines Data Services.
High Availability (HA): Systeme die eine Verfügbarkeit von mehr als 99,999 Prozent gewährleisten, deren (geplante und ungeplante) Downtime geringer als 5 Minuten jährlich beträgt, werden als „hoch verfügbar” bezeichnet. Jedoch ist diese Grenze nicht in Stein gemeißelt. Manche Anbieter sprechen auch bei Systemen mind. zu 99,99 Prozent (max. 53 Minuten Downtime jährlich) oder gar nur zu 99,9 Prozent (bis zu 8,5 Stunden Downtime jährlich) verfügbar sind.
Logical Host: Beherbergt Ressourcen wie Disk Groups, logische Hostnamen, logische IP-Adressen. Clients sprechen Cluster-Ressourcen über den logischen Host an. Über dieses Konzept verhält es sich für den Client transparent, welcher physische Knoten des Clusters tatsächlich die Ressource zur Verfügung stellt.
Multihost Disk: Disk, die von mehreren physischen Knoten parallel angesprochen werden kann. Auch Shared Disk Subsystem genannt.
Failover: Aufgrund eines Ausfalls oder Fehlers (z.B. von Server Hardware, Harddisk, Netzwerkkarte, des Betriebssystems, eines Data Services etc.) verursachter Wechsel des physischen Knotens im Cluster.
Failback: Geplanter Wechsel auf den Original-Knoten.
Geo-Spiegelung: Der Begriff Geo-Spiegelung bzw. Geo-Mirroring lehnt sich an das Wort „geografisch“ an. Gemeint ist ein Spiegel über geographische Grenzen hinweg. Typisch sind Konstellation mit einem Hauptrechenzentrum, in dem sich der zentrale Datenbankserver für die Produktion befindet und einem dezentralen Rechenzentrum, in dem eine gespiegelte Datenbank als Standby-System gehalten wird.

Anzeige

sen und versucht die Datenbank zu mounten. Ein Aktiv/Passiv-Cluster ist jedoch nicht für konkurrierendes Schreiben ausgelegt. Wird nun in die Datafiles einer Datenbank von verschiedener Seite geschrieben, könnten interne Inkonsistenzen dazu führen, dass die Datenbank komplett unbrauchbar wird.

Doch auch hierfür gibt es Mechanismen: Die Abstimmung der Cluster-Knoten über Quorum bzw. Voting Devices. Ohne Zugriff auf das gemeinsam genutzte Plattensubsystem kann die Datenbank nicht gemountet werden. Solange aber Zugriff auf diese Speicherbereiche besteht, kann – zumindest notfalls – auch über diesen Storage kommuniziert werden. Die Kommunikation im Quorum bzw. Voting-Verfahren ist zwar verglichen mit jener über das private Network unglaublich langsam. Dafür bietet diese alternative Abstimmungsmethode eine zusätzliche Sicherheit.

Alles in allem handelt es sich bei den heutigen Cluster-Architekturen um gut ausgereifte Systeme, die zwar teurer in Anschaffung und Wartung als Single Instanz Systeme sind, dafür aber sehr viel mehr an Ausfallsicherheit und Komfort bieten.

Was kann wer?

Oracle fährt eine recht zweigleisige Strategie: Aktiv/Aktiv-Cluster mit Shared Disk-Architektur werden ebenso angeboten wie Aktiv/Passiv-Konstellationen, die jedoch durch Dritthersteller (Stichwort: Clusterware) unterstützt werden müssen. Bis Oracle 9i muss allerdings ebenfalls die Clusterware des Vendors zum Einsatz gebracht werden. Mit 10g entfällt dies: Oracle bietet nun für den Real Application Cluster eine eigene Clusterware für alle gängigen Plattformen mit.

DB2 zeigt – je nach Plattform – ganz unterschiedliche Ansätze. Im Mainframe Bereich erreicht DB2 nach wie vor 99,999-prozentige Verfügbarkeit durch die Hard- und Software. In Client/Server-Umgebungen kommen eher Cluster-Lösungen zum Einsatz. Die Aktiv/Aktiv-Konstellation von DB2 im Cluster adressiert durch ihre Shared Nothing-Architektur primär Skalierbarkeit und weniger Hochverfügbarkeit. Dafür ist sie leichter zu implementieren als Oracles RAC: Eine Aktiv/Aktiv-Lösung z.B. unter AIX benötigt zwei Knoten und ein einfaches Disk Subsystem. Die Anforderungen an das Disk Subsystem sind gering, da hier – anders als beim Shared Disk Cluster – nur exklusiv von einem Knoten zugegriffen wird. Zusätzlich zur Datenbanksoftware benötigt DB2 ebenfalls eine Cluster-Software wie HACMP bei AIX, im Windows-Umfeld den Microsoft Cluster Server, im Fall eines Einsatzes von Sun der Sun Cluster bzw. Veritas Cluster. Mit Linux kann der Steele Eye Livekeeper eingesetzt werden. Jeder der Cluster-Knoten verwaltet nur einen Teil der Daten. Damit im Ernstfall trotzdem noch alle Daten im Zugriff sind, sollten das Plattensubsystem so verkabelt sein, dass auch ein zweiter Knoten darauf zugreifen und im Falle eines Ausfalles des Primärknotens die Datenverwaltung übernehmen kann. Allerdings sind Aktiv/Passiv-Lösungen auch mit DB2 meist kostengünstiger. Generell ist – das gilt für jeden Datenbankhersteller – die Failover Time von Aufkommen und Art der Transaktionen abhängig. Einem Whitepaper von IBM zufolge konnte der Takeover eines DB2-Systems bei einem namhaftem deutschen Unternehmen in 10 Sekunden erreicht werden. Die Erfordernisse an Hard- und Software sind weitestgehend identisch. Jedoch ist diese Konstellation weniger Komplex und die Kosten für Hardware sind in der Regel wesentlich günstiger. Als Hot Standby kann hier auch eine Entwicklungsmaschine verwendet werden, die im Falle eines Ausfalls automatisiert als Produktivsystem zum Einsatz gebracht wird.

Sybase und der Adabas D Nachfolger SAP DB erlauben ebenso eine HA-Option, die als Aktiv/Passiv-Konstellation ebenfalls die Dataservices einer Cluster Software nutzt. Auch hier kann HACMP, Sun Cluster, Veritas Cluster oder HP ServiceGuard zum Einsatz gebracht werden. Auch hier werden die Sevices der Clusterware ganz ähnlich wie bei Oracle oder DB2 genutzt.

Spieglein, Spieglein an der Wand ...

Gegen Ausfälle ganz anderer Art schützt die Geo-Spiegelung. Während Cluster Systeme oftmals im selben Rechenzentrum stehen und somit vor einem Desaster wie Erdbeben, Feuer oder einstürzenden Neubauten letztlich wenig schützt, sorgt der Geo-Spiegel dafür, dass die Datenbank komplett an einen geographisch anderen Ort gespiegelt wird. Gerade Großbanken, aber auch einige touristische Anbieter nutzen dieses Verfahren gern, um auch im Notfall über Ausweichrechenzentren den Betrieb aufrecht erhalten zu können. Der Dritthersteller Libelle unterstützt mit seiner Software einen solchen Geo-Spiegel: Libelle DBShadow.

Cluster-Architekturen

Aktiv/Passiv: Bei Aktiv/Passiv-Architekturen ist zu einer Zeit immer nur ein Cluster-Knoten aktiv. Fällt der aktive Knoten aus, springt ein Standby-System ein. Vorteil eines solchen Clusters ist, das der Ausfall eines Rechners durch das passive Standby-System abgefangen wird. In aller Regel ist innerhalb weniger Minuten die Systemverfügbarkeit wiederhergestellt.

Aktiv/Aktiv: Bei Aktiv/Aktiv-Clustern unterscheidet man zwei grundlegend verschiedene Architekturen: Shared Nothing und Shared Disk. Während bei der Shared Nothing-Architekturen die Datenbank partitioniert wird und nur einer der am Cluster beteiligten Knoten die jeweilige Partition bedient, wird bei der Shared Disk Architektur von allen am Cluster beteiligten Knoten auf den gesamten Datenbestand zugegriffen. Shared Disk hat daher den Nachteil, dass zusätzlicher Kommunikationsoverhead bei der Koordination konkurrierender Zugriffe von mehreren Knoten auf dieselben Datenbank-Pages entsteht. Shared Disk-Architekturen skalieren daher auch nur mit einem Faktor von 1.8, das heißt, jeder zusätzliche Knoten bringt nicht die vollen 100 Prozent, sondern nur weitere etwa 80 Prozent an Mehr-Leistung. Dafür gewähren Shared Disk-Architekturen zusätzlich zur Skalierung auch Ausfallsicherheit: Bricht ein Knoten weg, steht der komplette Datenbestand über die verbleibenden Knoten zur Verfügung. Anders ist das beim Shared Nothing Cluster: Bricht hier ein Knoten weg, so stehen die Daten der gesamten zugehörigen Datenpartition nicht mehr zur Verfügung. Es gibt auch Mischformen: Aktiv/Aktiv-Cluster mit Shared Nothing Architektur, deren Verfügbarkeit über einen Hot Standby in Form einer zusätzlichen Aktiv/Passiv-Konstellation gewährleistet wird.

Typischer Vertreter der Shared Nothing Architektur ist DB2 von IBM, während Oracle mit dem Real Application Cluster den Shared Disk Zweig vertritt.

Die gesamte Datenbank wird im laufenden Betrieb auf das Spiegelsystem kopiert. Dabei sind je nach Systemumgebung Kopierleistungen von bis zu 200Gbyte pro Stunde möglich. Basis der Kommunikation zwischen den System und Prozessen ist eine TCP/IP Socket-Kommunikation, die unter anderem durch Nutzung von Shared Memory und durch paralleles, komprimiertes Kopieren eine hohe Performance erreicht. Veränderungen in der Echt-Datenbank können zeitversetzt nachgefahren werden, wobei der Zeitversatz angepasst und im laufenden Betrieb verändert werden kann. Während des Normalbetriebs stehen die DBShadow-Prozesse ständig in Kontakt mit dem Echt- und Spiegelsystem. Es erfolgt ein kontinuierlicher Informationsabgleich. Alle durchgeführten Aktionen werden in Log-Dateien protokolliert und redundant gespeichert. Bei einem Ausfall des Echt-Systems kann die Datenbank umgeschaltet werden. Sofern die Logfiles verfügbar sind, bietet sich bei Hardwarefehlern die Möglichkeit, bis zur letzten Transaktion zu recovern. Bei Auftreten eines logischen Problems wie einem Anwender- oder Softwarefehler kann auf den Zeitpunkt vor dem Fehler umgeschaltet werden.

Das Spiegelsystem steht nach dem Umschalten als Produktivsystem zur Verfügung. Die Dauer des Umschaltprozesses hängt von der Größe des Zeittrichters und der Performance des Spiegelsystems ab.

Mit DBShadow Long Distance Edition lassen sich entfernungsunabhängige Notfallrechenzentren realisieren – und über Kontinente hinweg. Mittels *VLP (Very Large Packages)*, optimierten Komprimieralgorithmen und massiv Parallelen Prozessen wird hier versucht, eine optimale Ausnutzung der begrenzten Bandbreite zu gewährleisten.

Die Steuerung von Libelle DBShadow erfolgt über ein graphisches Tool, das alle Aufgaben wie Anlegen des Spiegels, Überwachung und Notfallsteuerung übernimmt. Für Unix-Systeme steht zusätzlich ein Command Line Tool zur Verfügung. DB Shadow lässt sich in alle gängigen Systemmanagement Tools integrieren.

Doch auch andere Hersteller bieten Lösungen: SteelEye LifeKeeper zum Beispiel ist eine einfach zu bedienende, preiswerte Cluster-Lösung für Applikationen, Datenbanken, Server und Storage. Er unterstützt unter anderem DB2, Informix, SAP DB, MS SQL Server, Oracle, PostgreSQL und Sybase. Aber auch Applikationen wie MS Exchange, SAP R/3 oder der Apache lassen sich mit LifeKeeper clustern. Mit SteelEyes` Extended Mirroring kann ebenfalls gespiegelt werden.

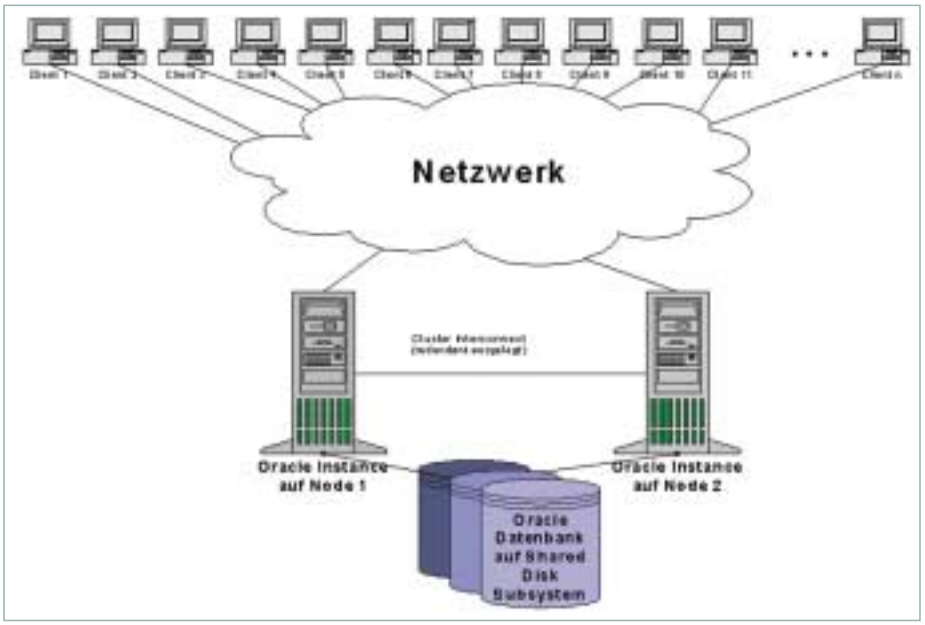


Abb. 1: Aktiv/Aktiv-Clusters mit Shared Disk-Architektur: Oracle Real Application Clusters

Quest Shareplex bietet Replikation von Tabellen und Sequences auf Basis der Redolog-Informationen einer Oracle Datenbank. Dabei liest ein Capture-Prozess auf Basis von Redolog-Informationen einer Oracle Datenbank die entsprechenden DML-Operationen aus den Redolog-Dateien. Ein Reader-Prozess übernimmt diese Informationen und liest anschließend die entsprechenden Datensätze der Primärdatenbank aus. Die entsprechende DML-Operation wird anschließend auf der Sekundärdatenbank nachgezogen.

Standby-Datenbanken

Eine recht einfache Architektur bieten Standby-Datenbanken. Diese machen sich jenen Mechanismus zunutze, der eigentlich für ein Recovery des Systems vorgesehen ist: Über die Transaktionslogs des Primärsystems werden vom Standby-System einfach alle Transaktionen recovert. Die Implementation ist recht einfach. Im Grunde wird eine Vollsicherung im Offline-Modus der Datenbank auf ein zweites System kopiert. Nach dem Startup des Primärsystems werden die Transaktionslogs regelmäßig auf das Standby-System übertragen. Das Standby-System befindet sich dabei in einer Art dauerhaftem Recovery: Es liest stetig die archivierten Transaktionslogs des Primärsystem und reproduziert hierdurch jede einzelne Transaktion. Der Failover ist allerdings nicht allzu komfortabel: Der Administrator muss in aller Regel manuell Umschalten und Vorteil eines Clusters, dass mit der Übernahme einer Ressourcengruppe auch eine logische IP-Adresse und ein logischer Hostname übernommen werden kann, gibt es hier erst mal nicht. Allerdings lässt sich hier im Allgemeinen auch wiederum manches über Skripte automatisieren. Anzuraten ist eine Standby-DB nur, wenn das System nicht 24*7 Verfügbarkeit benötigt. Im Falle sehr hoher Verfügbarkeitsanforderungen empfiehlt sich die Standby-DB auch als Ergänzung zu einem Cluster System. DB2 bietet Standby-Datenbanken ebenso an wie Oracle und Sybase.

Replikation

Sybase bietet im Warm-Standby Setup einen Real Time Replikation an, die alle Datenänderungen auch über entfernte Strecken überträgt. Dabei werden sowohl Datenmanipulation, aber auch Schema-Änderungen wie das Anlegen einer Tabelle repliziert. Sybase repliziert dabei sogar

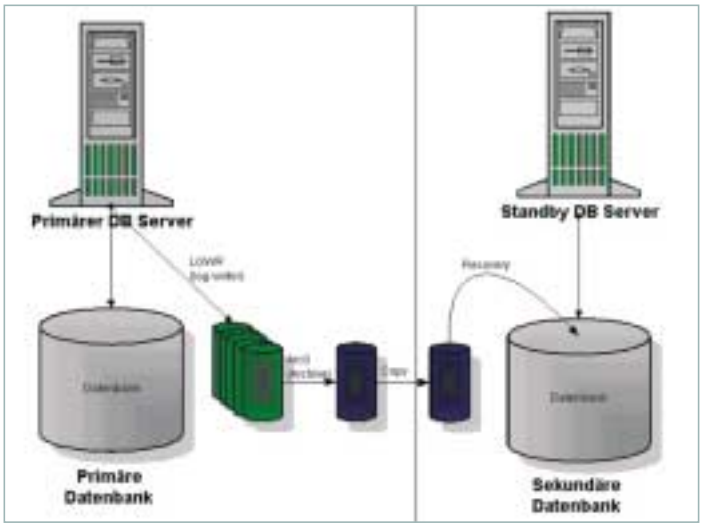


Abb. 2: Konzept einer Standby-Database mit replizierten Transaktionslogs

aus den Transaktionslogs, benötigt daher keine internen Shadow Tables und vermeidet damit auch Reihenfolge-Probleme. Letztere dürften nahezu jedem, der sich mit größeren replizierten Umgebungen herumschlagen musste, bekannt sein. Sybase gestattet dabei sogar unterschiedliche Plattformen. So könnte eine Sybase-Datenbank auf Windows-Server mit einer Warm Standby-Replikation über ein Sun System abgesichert werden. Der Failover. Open Switch ist ein Add On von Sybase, das bei der Replikation mit eingesetzt werden kann. Ohne Open Switch muss der Client sich nach einem Ausfall explizit neu verbinden. Open Switch liegt zwischen Datenbank und Applikation und routet alle Verbindungen des Clients um auf den Datenbankserver. Sobald der Datenbankserver ausfällt, hält Open Switch die Verbindung zum Client und konfiguriert sich intern so um, dass automatisch das bisherige Standby-System als Primärsystem verwendet wird.

Logische Fehler

Man glaubt es kaum, doch es ist wahr: Einer der häufigsten Fehler ist das „versehentliche" Löschen von Daten. Neben dem klassischen *Delete* ohne *Where*-Klausel greift auch das Drop Table schnell mal in die Vollen. Klassische Wiederherstellungsstrategie ist das Recovery der letzten Vollsicherung. Doch kann, sofern nicht auch die Transaktionslogs gesichert wurden, hiermit nicht bis zum letzten Datensatz recovert werden. Die genannte Spiegel-Lösungen bieten hier Schutz, kosten aber zusätzliche Lizenzgebühren. Doch auch mit Bord-eigenen Mittel, Tools also, die durch den jeweiligen Datenbankhersteller mit ausgeliefert werden, können solche „Human Errors" abgefangen werden. Genutzt wird hier meist die ein oder andere Form zeitverzögerter Replikation. Auf ein Standby-System wird eine vollständige Datensicherung kopiert. Die Datenänderungen werden vom Primärsystem via Replikation auf das Standby-System übertragen. Je nach Konzept wird hier entweder logisch auf Satzebene oder über ein Nachfahren der Transaktionslogs repliziert. Dies kann – ähnlich dem Geo-Spiegeln – mit einer Verzögerung realisiert werden. Wurden nun Daten versehentlich gelöscht, kann aus der zeitversetzt replizierten Datenbank der alte Datenbestand extrahiert werden.

Oracle bietet hier einen weiteren Ansatz: Über Undo Tablespaces können Flashback Queries abgesetzt werden. Hier wird einfach angegeben, welchen Zeitpunkt die Ausgabe der Daten wiederspiegeln soll. Wurde beispielweise um 13:07 Uhr eine Tabelle gelöscht, kann mit Flashback Query forciert werden, dass die Daten in Ihrem Datenbestand von 13:00 Uhr angezeigt werden. Dieses Feature kann dazu genutzt werden, gelöschte Datensätze wiederherzustellen.

Fazit

Die Kosten eines ausfallsicheren Systems sind recht hoch. Laut einer Studie der Gartner Group steigen die Kosten, je näher man der 100 Prozent Marke kommt, weit überproportional. Betrachtet man die korrelierten Ausfallzeiten, wird der Grund hierfür auch schnell deutlich: Hat man bei einer Verfügbarkeit von 95 Prozent noch eine mögliche maximale Ausfallzeit von 18 Tagen und 6 Stunden im Jahr, so schrumpft diese bei einer 99-prozentigen Verfügbarkeit auf nunmehr rund 3 Tage und 16 Stunden. Richtig interessant wird es, sobald man sich deutlich an 100 Prozent Uptime annähert: 99,9 Prozent Verfügbarkeit gestattet eine maximale jährliche Ausfallzeit von 8 Stunden und 46 Minuten. Das ist schon verdammt wenig, bedenkt man, dass in aller Regel geplante Wartungszeiten wie das Einspielen von Patches und Upgrades diesen Zeitraum ganz locker verschlingen.

Bei 99,99 Prozent sind schon nur noch 53 Minuten jährliche Downtime zulässig und bei 99,999 Prozent ist man schon unterhalb von 5 Minuten im Jahr. Hier bewegt man sich bereits in Bereichen, die in komplexen Client/Server Systemen kaum noch zu erreichen sind.

Die Kosten für den Einsatz redundanter Hardware, von Geo-Spiegel und Cluster Systemen schrecken viele IT-Verantwortliche. Doch man sollte hier keinesfalls die Kosten für Ausfallzeiten während der Geschäftszeiten außer Acht lassen. Diese beiden Kostenfaktoren sollten stets einander gegenübergestellt werden. Der geschätzte Break Even ist ein guter Anhaltspunkt zur Auswahl des Ausfallkonzeptes. Doch neben den Kosten sollten mögliche Image-Schäden, die schwer in Zahlen zu bemessen sind, bedacht werden.

Links & Literatur

- www.networkingnext.com/clusteringarc.html
- www.sun.com/software/cluster/wp-failover/clusters-failover.pdf
- h18000.www1.hp.com/
- slc.sybase.com/documents/edb513.pdf
- P. Dadam: Verteilte Datenbanken und Client/Server Systeme, Springer Verlag
- M.T. Ozsu und P. Valduriez: Principles of Distributed Database Systems, Prentice Hall
- E. Rahm: Mehrrechner-Datenbanksysteme, Addison Wesley
- S. Ceri und G. Pelagatti: Distributed Databases – Principles and Systems. Mc Graw Hill
- A. Kemper und A. Eickler: Datenbanksysteme – eine Einführung, Oldenbourg Verlag
- S. Abiteboul, P. Buneman und D. Suciu: Data on the Web, Morgan Kaufmann
- Özsu, P. Valduriez: Principles of Distributed Database Systems, Prentice Hall

Anzeige

MySQL wird erwachsen

Status quo – schafft MySQL den Sprung in die Enterprise-Liga?

Im Gegensatz zu anderen DBMS folgt MySQL AB konsequent seiner Unternehmensphilosophie und versucht, Features und Innovationen eher behutsam einzugliedern. Mit der Version 4.1 ist MySQL nicht nur auf dem richtigen Weg angelangt, sondern wird zunehmend ein ernst zu nehmender Konkurrent für kommerzielle Datenbankhersteller. Zudem scheint das Ziel von MySQL klar anvisiert zu sein – der Sprung in die Enterprise-Liga soll mit den kommenden Versionsreihen 4.1 und 5.0 gelingen.

von Andre Gildemeister

Ein Blick zurück

Neben der jüngst abgeschlossenen Partnerschaft von MySQL [1] und SAP zur Entwicklung eines gemeinsamen DBMS für Unternehmensanwendungen sorgte vor allem die Veröffentlichung der ersten Alpha-Version der nächsten MySQL-Versionsreihe 4.1 für Schlagzeilen. Mit dieser Version bleibt MySQL seiner Linie treu und versucht sukzessive die von Anwendern geforderten Features zu implementieren, um somit einen Schritt näher an den SQL-Standard heranzurücken und den gestiegenen Ansprüchen zu entsprechen. Doch zuerst eine kleine Rückblende in die Geschichte von MySQL.

In der Vergangenheit hat My SQL vor allem diejenigen überzeugt, die Wert auf Performanz und Geschwindigkeit legten. Ein enorm wichtiger Aspekt im Webumfeld, wo es bekanntermaßen auf schnelle Antwortzeiten und beschleunigte Verarbeitung ankommt. Dazu war MySQL ein Inbegriff für Stabilität und Einfachheit in der Bedienung von Datenbanken. Weiterführend kam noch die absolut freie Verfügbarkeit (GPL-Lizenz) hinzu, welche bis heute für einen wahnsinnigen Popularitätsschub sorgt und maßgeblich am Erfolg beteiligt ist.

Die fehlenden Features bzw. Fähigkeiten, welche eine Datenbank heutzutage von Haus aus mitbringen sollte, verhinderten bis jetzt den ganz großen Durchbruch. MySQL war eine Datenbank, die man für Webanwendungen hervorragend einsetzen konnte, aber es war keine universell einsetzbare Datenbank. Bei den fehlenden Eigenschaften handelte es sich vornehmlich um Transaktionen und referenzielle Integrität, welche beide maßgebend zur Datenkonsistenz und Integrität beitragen. Dies war unter anderem auch ein Hindernis für den Einsatz im Unternehmen, da man sich in puncto Datensicherheit eher nach anderen Datenbanken umsehen musste.

Somit war der schwedische Datenbankhersteller gegen Ende der Versionsreihe 3.23 gezwungen einzugreifen. Er musste seinen schnellen Datenbankserver den Ansprüchen gemäß an ein zeitgemäßes RDBMS anpassen. MySQL AB befand sich dabei aber gewissermaßen in einer Zwickmühle, denn auf der einen Seite sollte die Performance nicht beeinflusst werden, aber auf der anderen Seite wiederum die angesprochenen Verbesserungsvorschläge implementiert werden.

Mit dem InnoDB-Tabellen-Handler, der in einer späteren 3.23-Version Einzug gehalten hat, ist MySQL jetzt mittlerweile gut ein Jahr ausgestattet und erlaubt somit Transaktionen nach dem ACID-Prinzip unter der Voraussetzung, Tabellen vom Typ InnoDB zu verwenden. Dieses Tabellenformat legte ab Version 4.0 mit dem neuen *.frm*-Dateiformat intern den Grundstein für die Erweiterungen, welche in Version 4.1 zur Verfügung stehen werden. Die meisten der folgenden Verbesserungen oder hinzugefügten Fähigkeiten sind im Laufe der Versionsreihen 3.23 über 4.0 in 4.1 eingeflossen, ohne dass man sie fest einer Versionsnummer zuweisen kann.

MySQL und Standards

Ein anderer wichtiger Aspekt zur Verbesserung war das Thema Standards, welcher bei MySQL bislang nicht unbedingt als so wichtig eingestuft wurde. Bei MySQL konnte man bis jetzt nicht davon sprechen, dass es standardkonform war, zu groß war bislang die Abweichung. Sicherlich ist dies ein allgemeines Problem bei Datenbanken, bei denen entweder der verfügbare Sprachumfang des SQL-Standards gebrochen und darüber hinaus entwickelt wird, oder die Forderungen der SQL-Norm nicht erfüllt werden. Neben den drei aktuellen ANSI-Normen SQL-92, SQL-99 und SQL: 2003 kann man MySQL momentan am ehesten dem SQL-92-Standard zuordnen, obwohl dieser auch nur zum Teil unterstützt wird. Langfristig und vor allem im Hinblick auf die kommenden Versionen, ist die volle SQL-99-Unterstützung angepeilt.

Transaktionsunterstützung dank InnoDB

Entscheidet man sich für einen transaktionsbasierten Tabellentyp, kommt man an InnoDB nicht mehr vorbei. Dieser hat

sich mittlerweile gegenüber seinen Konkurrenten (DBD, Gemini) etabliert und abgesetzt. Im Gegensatz zu nicht transaktionsbasierten Tabellentypen ist bei InnoDB nach wie vor ein kleiner Performance-Verlust zu verzeichnen, den man wohl oder übel in Kauf nehmen muss.

Die Intention hinter Transaktionen ist, dass die Datenkonsistenz in jedem Fall aufrecht erhalten bleiben soll. Es schützt zudem davor, dass nicht mehrere Anwender Datensätze zeitgleich manipulieren können und somit für Inkonsistenzen im Datenmodell sorgen. Ein typisches Szenario wäre, wenn ein Anwender mittels *BEGIN* eine Transaktion startet und innerhalb dieser Phase einen Datensatz manipuliert (*UPDATE*, *INSERT*, *DELETE*) sowie alle notwendigen Operationen darin ausführt. Während dieses Prozesses ist es anderen Anwendern untersagt, zeitgleich diesen Datensatz zu bearbeiten. Erst wenn der Anwender seine Transaktion erfolgreich mit *COMMIT* abschließt, wird der Sperrmechanismus aufgehoben. Sollte der Fall eintreten, in dem die Transaktion nicht abgeschlossen werden soll und der alte Zustand wieder herzustellen ist, so wird *ROLLBACK* ausgeführt.

Sollte eine Transaktion aus verschiedenen Gründen (Stromausfall, Software- oder Hardwarefehler) nicht erfolgreich abgeschlossen werden können, so setzt eine weitere nützliche Eigenschaft ein, dass sogenannte Crash Recovery. Dieser Mechanismus stellt den Zustand zu Beginn der Transaktion wieder her und sorgt für die notwendige Konsistenz der Datenbank.

In Anlehnung an andere RDBMS wurden verschiedene SQL-Statements in Verbindung zur Datensicherheit implementiert. Unter anderem die Ergänzung des *SELECT*-Statements um *FOR UPDATE*, welches innerhalb einer Transaktion einzelne Datensätze vor Änderungen blockiert.

Innerhalb einer gestarteten Transaktion mit *BEGIN* wird mittels *SELECT username, password FROM customer WHERE customer_id = 1024 FOR UPDATE* der Datensatz solange gesperrt, bis die Transaktion erfolgreich durch *COMMIT* abgeschlossen wurde.

Fremdschlüssel und referenzielle Integrität

Lange musste man darauf warten, seit der Version 3.23.44 ist es nun endlich fester Bestandteil von MySQL. Die Rede ist von Fremdschlüsseln (Foreign Keys) und referenzieller Integrität, mit dem MySQL einen weiteren Schritt hin zu einem vollwertigeren RDBMS geht. Bei einer Verknüpfung von mehreren Tabellen war es bislang nicht möglich, den Fremdschlüssel festzulegen, welcher eindeutig die Beziehung zu einer anderen Tabelle herstellt. Dies ist jetzt möglich, allerdings ist es zwingend notwendig, dass für die Schlüsselspalten in den referenzierten Tabellen ein Index angelegt wird, ansonsten erkennt MySQL den Fremdschlüssel nicht an. Eine weitere Voraussetzung ist, dass beide Tabellen vom Typ InnoDB sein müssen und den gleichen Datentyp besitzen sollten. Zudem entfernen jegliche *ALTER TABLE*-Statements sämtliche Fremdschlüsselbeschränkungen, die für die Tabellen galten.

Zurück zur referenziellen Integrität, die ist dann gewährleistet ist, wenn jeder Wert eines Fremdschlüssels (Foreign Key) auch als Primärschlüssel (Primary Key) in der referenzierten Tabelle, auf die verwiesen wurde, vorkommt. Das Ziel ist zudem, die Beziehung zwischen mehreren Tabellen bei *DELETE*-, *INSERT*- und *UPDATE*-Befehlen konsistent und intakt zu halten. Um diese Thematik zu veranschaulichen, ist nachfolgend ein *CREATE TABLE*-Statement zu sehen, womit zwei Tabellen aus einem Shopsystem angelegt werden:

```
CREATE TABLE customer (
 customer_id INT(11),
 firstname VARCHAR(30),
 lastname VARCHAR(40),
 email VARCHAR(40),
 KEY customer_id (customer_id),
) TYPE = InnoDB

CREATE TABLE order (
 order_id INT(11),
 customer_id INT(11),
 order_timestamp INT(11),
 KEY order_id (order_id),
 KEY customer_id (customer_id),
 FOREIGN KEY (customer_id) REFERENCES customer
                          (customer_id) ON DELETE CASCADE
) TYPE = InnoDB
```

In der Tabelle *customer* werden sämtliche Kundendetails gespeichert, wobei *customer_id* als Primärschlüssel definiert wird. Die referenzierte Tabelle *order* nimmt sämtliche Kundenbestellungen ent- gegen. In dieser Tabelle stellt *order_id* und *customer_id* eine Beziehung zu einem Kunden her, wobei *customer_id* als Fremdschlüssel definiert wurde.

Durch die Verwendung von Fremdschlüsseln existieren sogenannte Integritätsregeln, welche die Konsistenz der Datenbank aufrecht erhalten wollen. Es werden Fehler verhindert, welche im laufenden Betrieb sehr oft unbewusst auftreten. Sollte ein Eintrag in *order* mit einer *customer_id* erfolgen, welche in *customer* nicht oder dazu kein zugehöriger Kunde existiert, so schlägt dies genauso fehl, als würde *customer_id* in *customer* geändert werden. Sollte eine Kunde in *customer* gelöscht werden, zu dem mehrere Bestellungen in *order* gehören, sorgt *ON DELETE CASCADE* dafür, dass der Kunden inklusive all seiner Bestellungen gelöscht wird. Zurzeit beschränkt sich die Fremdschlüsselunterstützung allerdings ausschließlich nur auf den InnoDB-Tabellentyp. Für eine der nächsten Versionen ist aber die volle Unterstützung von Fremdschlüsseln für alle Tabellentypen geplant.

Query Cache

Bislang verarbeitete MySQL-Anfragen immer nach dem gleichen Schema, erst wurde der jeweilige Query geparst und anschließend ausgeführt. Dies war bis dato die einzige Möglichkeit, auch bei komplizierten Queries mit langen Abfragezeiten. Ab Version 4.01 kann der Query Cache hier Abhilfe schaffen und die Abfragen beschleunigen. Bei aktiviertem Query Cache speichert der MySQL-Server den *SELECT*-Befehl inklusive Ergebnis in einem Zwischenspeicher. Wenn jetzt eine identische Anfrage an den Server geschickt wird, so wird das Ergebnis direkt aus dem Query Cache entnommen, ohne die Anfrage erneut vom MySQL-Server ausführen zu lassen. Dabei ist allerdings zu beachten, dass der MySQL-Server einen ausgeführten Query nur dann als identisch betrachtet, wenn er in absolut gleicher Form vorliegt. Das gilt insbesondere für Groß- und Kleinschrei-

bung, auf die der MySQL-Server sehr sensibel reagiert.

Diese Methode ist besonders empfehlenswert, wenn sie bei Anwendungen angewandt wird, bei denen der Datenbestand nicht häufig wechselt, wie z. B. bei dynamischen Inhalt, wo der MySQL-Server somit vor redundanten Abfragen bewahrt werden kann. Sollte der entsprechende Datensatz aber durch ein *INSERT*-, *UPDATE*- oder *DELETE*-Statement verändert werden, so wird das Ergebnis des Query Cache mit sofortiger Wirkung geleert.

Die Konfiguration des Query Cache ist relativ einfach zu steuern. Der Parameter *query_cache_type* bestimmt, ob der Status des Query Cache aktiv *ON* oder inaktiv *OFF* ist. Mit dem Parameter *DEMAND* werden gezielt ausgewählte *SELECT*-Abfragen zwischengespeichert. Die Anweisung *query_cache_limit* sagt aus, dass die Ergebnismenge nicht größer als das festgelegte Cache-Limit sein darf. Zu guter Letzt muss noch die Größe des Cache mit der Anweisung *query_cache_size* definiert werden, welcher die Ergebnisse zwischenspeichert. Um zu überprüfen, wie effektiv der Query Cache bei einer Anwendung wirklich agiert, reicht ein Blick auf die Statusvariablen mittels *SHOW STATUS LIKE '%qc'*. Ab Version 4.1 soll der Query Cache auch innerhalb von Transaktionen anwendbar sein.

Änderungen an DELETE, UPDATE und INSERT-Statements

War es bisher nur möglich, Daten mittels *DELETE* in einer Tabelle zu löschen, existiert seit MySQL 4.0.2 die Möglichkeit, Daten in mehreren Tabellen zeitgleich zu löschen. Mit der folgenden Anweisung *DELETE FROM order, customer USING order, customer WHERE customer.customer_id = order.customer_id AND customer.customer_id = 1032*, werden alle Bestellungen des Kunden und schließlich er selber gelöscht. Das *UPDATE*-Statement lässt sich fortan auch bequemer einsetzen, da es möglich ist, mehrere Tabellen beim Ändern von Daten zu verwenden. Diese Verbesserungen machen die Verwendung nicht nur einfacher, sondern auch eleganter und ersparen so manchem Programmierer zusätzlichen Programmcode. Endlich kann *INSERT* mit dem Zusatz *ON DUPLICATE KEY* ausgestattet werden, welcher dadurch bspw. zum Zählen von Einträgen verwendet werden kann.

Neuerungen in MySQL 4.1

Das lange Warten auf einige heiß begehrte Features hat mit der Version 4.1 ein Ende gefunden. Die zurzeit in der Alpha-Version [2] vorzufindende Datenbank wird als ein weiterer Meilenstein in die MySQL-Geschichte eingehen. Aber nicht nur die wichtigen Neuerungen (Subselects, OpenGIS, Prepared Statements) sind hervorzuheben, vielmehr wurde versucht, MySQL weiter zu optimieren, um für eine bessere Performanz zu sorgen. Wie erwähnt, befindet sich MySQL 4.1 erst im Alpha-Stadium und enthält einige Kleinigkeiten, welche noch nicht voll funktionsfähig sind. Denken Sie vor allem daran, diese Version momentan nur zum Testen zu verwenden und auf keinen Fall in Produktionsumgebungen einzubinden.

In den folgenden Passagen werde ich auf die einzelnen unten aufgeführten Features von MySQL 4.1 eingehen, welche in der Übersichtstabelle zu sehen sind.

Endlich Subselects

Um Daten aus verknüpften Tabellen abzufragen, war man bis dato an *JOIN*s gebunden. Endlich wurden die sehnlich vermissten Subselects (Unterabfragen) implementiert, die bei fast allen Anwendern ganz oben auf der Wunschliste standen. Während das SQL-Statement *UNION*-Abfragen auf einer Ebene zusammenfasst, sind mittels Subselects verschiedene verschachtelte Unterabfragen möglich. Ein Part von MySQL sind sie ab der Version 4.1.0. Der zurzeit vorzufindende Alpha-Subselect-Code befindet sich momentan noch in der Stabilisations- und Optimierungsphase.

Die Implementierung von Subselects verlief nicht so reibungslos, wie einst vermutet. Nach wie vor musste MySQL erst seinen Geschwindigkeitsansprüchen gerecht werden, die wie bei allen anderen hinzugefügten Neuerungen nicht beeinflusst werden sollte. Dies stellte die MySQL-Entwickler vor eine schwierig zu lösende Aufgabe, welche im Februar 2002 endlich gestartet wurde. Bislang konnte man seine Abfragen zwar des Öfteren als *JOIN* formulieren, allerdings waren diese Konstrukte oftmals zu umständlich oder zu komplex konstruiert. Zudem kommt man bei manchen Querys nicht an Subselects vorbei, welche sich dazu in der Anwendung allgemein als viel komfortabler und flexibler ausgezeichnet haben. Sicherlich ist das aber auch eher eine Gewohnheits- bzw. Geschmacksfrage, obwohl oftmals bestimmte Abfragen als Subselects einfacher zu verstehen sind. Ähnlich wie bei *JOIN*s fragt man bei Subselects auch Daten ab, welche zueinander in relationaler Abhängigkeit stehen. Die Besonderheit besteht darin, dass einem ein Subquery erlaubt, dass Ergebnis eines Query innerhalb eines anderen Query zu verwenden. Innerhalb unserer Shop-Datenbank würde das wie folgt aussehen:

```
SELECT firstname, lastname FROM customer
  WHERE customer_id = (SELECT customer_id FROM order
WHERE order_timestamp = 1057010400)
```

Sollten Sie sich jetzt fragen, ob dies auch mit einem *JOIN* funktioniert, so ist das in diesem Fall noch ohne Probleme möglich:

```
SELECT firstname, lastname FROM c.customer, o.order
  WHERE c.customer_id = o.customer_id AND o.order_
                                timestamp = 1057010400
```

Wenn wir aber den Vor- und Nachnamen des Kunden innerhalb einer Ab-

Features von MySQL 4.1

- Subselects (Unterabfragen)
- Optimiertes Client/Server-Protokoll (Prepared Statements, Multi-Query-Ausführung)
- Verschlüsselte Datenübertragung (SSL) zwischen Client und Server
- Geometrische Berechnungen auf Basis des OpenGIS-Standards
- Multi-Master-Replikation
- Unterstützung internationaler Zeichensätze UCS2 und UTF8

Anzeige

Anzeige

Anzeige

Anzeige

frage herausbekommen möchten, welcher die erste Bestellung aufgegeben hat, ist dies ohne Subselect nicht so einfach möglich:

```
SELECT firstname, lastname FROM customer
  WHERE customer_id = (SELECT customer_id FROM order
    WHERE order_timestamp = MIN(order_timestamp))
```

Im direkten Vergleich zu Subselects können *JOIN*s nur innerhalb von *SELECT*-Statements und eingeschränkt in *UPDATE*- und *DELETE*-Statements verwendet werden. Subselects hingegen können bei den folgenden Statements (*SELECT*, *UPDATE*, *INSERT*, *DELETE*, *REPLACE*, *DO*, *SET*) angewandt werden. Des Weiteren unterstützt MySQL 4.1 die Subselect Operatoren *IN*, *EXIST*, *ANY*, *SOME* und *ALL*, welche in der *WHERE*-Klausel Anwendung finden:

- *IN* prüft, ob ein Wert in dem Ergebnis einer Unterabfrage enthalten ist.
- *EXIST* stellt fest, ob die Unterabfrage mindestens eine erforderliche Zeile zurückliefert.
- *ANY* prüft, ob die Bedingung für irgendeine Zeile der Unterabfrage zutrifft.
- *SOME* arbeitet äquivalent zu *ANY*.
- *ALL* stellt fest, ob die Bedingung für alle Zeilen der Unterabfrage zutrifft.

OpenGIS-Datentypen

In MySQL 4.1 wurde ein neuer Datentyp *GEOMETRY* hinzugefügt, mit dem es möglich ist, geometrische Daten nach dem OpenGIS-Standard [3] zu verarbeiten. Darin lassen sich geometrische Daten wie Punkte, Linien und Polygone speichern. Dies war zwar bislang auch mit existierenden Spaltentypen wie bspw. mit einem BLOB (Binary Large Object) machbar, denn gespeichert werden ja lediglich Koordinaten. Der Vorteil des geometrischen Datentyps kommt erst dann zur Geltung, wenn eine Reihe neuer Funktionen [4] angewandt werden, welche in MySQL 4.1 Einzug gehalten haben und sich an dem OpenGIS-Projekt orientieren. Damit lassen sich bequem Koordinaten eines Punktes ausgeben, Flächen eines Polygons oder Schnittpunkte von Linien berechnen. Dies ist aber nur ein Teil der Möglichkeiten, welche man mit den Funktionen berechnen kann. Im Laufe der Version 4.1 werden noch weitere Funktionen hinzugefügt, welche MySQL dazu verleiten, geometrischen Daten zu verarbeiten und auszuwerten.

Ausführungsgeschwindigkeit durch Prepared-Statements verbessern

Eine weitere Neuerung welche erstmals in MySQL 4.1 implementiert wurde, sind Prepared-Statements. Im Prinzip ist ein Prepared-Statement ein SQL-Statement mit Platzhaltern. Erst bei der Ausführung werden die zu verwendenden Werte spezifiziert und ausgeführt. Damit können Sie die Ausführung von Queries beschleunigen, da der MySQL-Server nur noch einmal einen Query-Plan erstellen muss. Entwickler kennen diese Methode schon aus Datenbank-Abstraktions-Layer in PHP (*PEAR::DB*) [4] oder aus Perl (DBI) [5] und konnten diese auch schon praktisch nutzen. Da MySQL aber Prepared-Statements bislang noch nicht unterstützte, blieb ein Performance-Gewinn aus. In der Version 4.1 enthält die Client Library erstmals Datentypen und Funktionen (*mysql_prepare()*, *mysql_execute()*), mit denen Prepared-Statements vorbereitet und anschließend mit Daten gefüllt werden können. In der Praxis sieht die Anwendung der so eben genannten Funktionen folgendermaßen aus:

Zeichensätze festlegen

```
§db->prepare("INSERT INTO order (product_type,
order_timestamp, customer_id) values (?,?,?)");
$db->execute('Inline-Skater',103939333,221);
$db->execute('Fahrrad',1039393433,131);
```

Zuerst wird der Query-Plan aufgestellt, welcher anschließend mit unterschiedlichen Werten gefüllt wird. Aus Sicht der Datenbank haben Prepared-Statements Vorteile, weil die Ausführungsstrategie (welche Indizes, welche Reihenfolge bei *JOIN*s etc.) relativ unabhängig von den konkreten Werten des Query ist. Die Datenbank kann also die Prepared Statements cachen und spart sich bei weiteren Ausführungen des Query das Parsen, Analysieren und Optimieren, was natürlich zu deutlichen Performance-Steigerungen führt. Den größten Gewinn kann man mit dem Feature aber erzielen, wenn man viele Queries der gleichen Struktur, aber mit verschiedenen Werten hat. Lohnen tut es sich auch, wenn Sie einen Query besitzen, welcher nicht nur einmalig ausgeführt wird. Allerdings werden momentan noch nicht alle Query-Informationen gecacht, dies soll aber spätestens in Version 5.1 nachgeholt werden.

Backup und Lastverteilung mit Replikationen

Befindet sich MySQL in einer Produktionsumgebung, in der ein einziger Server der Last nicht Stand halten kann, ist man mit Replikation auf der sicheren Seite. Dieser Mechanismus erlaubt es, alle Anfragen der Clients an den jeweiligen Master-Server weiterzuleiten, der die auszuführenden Datenänderungen in eine Datei namens Binary Log schreibt. Der untergeordnete Slave-Server wiederholt diese Operation und befindet sich somit auf dem gleichen Stand wie der Master-Server und besitzt dazu noch ein aktuelles Backup. Sollte der Master-Server unerwartet ausfallen, kann der Slave-Server problemlos in dessen Rolle schlüpfen und dessen Aufgabe übernehmen. Wer Wert auf Ausfallsicherheit, Lastenverteilung (Load Balancing) oder Backups legt, ist mit dem MySQL Replikationsmechanismus bestens beraten. In Version 4.1 soll auch endlich Multi-Master-Replikation enthalten sein.

Geplante Features der MySQL-Version 5.0

- Gespeicherte Prozeduren (Stored Procedures)
- Trigger
- Fremdschlüssel mit kaskadierenden löschen (MyISAM)
- Vollständige Subselect-Unterstützung
- ANSI SQL 99-Unterstützung

Weitere Neuerungen

Neben den vorgestellten Verbesserungen hat MySQL natürlich auch in anderen Bereichen zugelegt, welche ein bisschen im Schatten der Hauptverbesserungen stehen.

- Ab sofort sind Änderungen an den Serverparametern, welche innerhalb von *my.cnf* definiert werden, im laufenden Betrieb möglich. Somit lassen sich Werte für den ganzen Server (*SET GLOBAL*) oder für einzelne Verbindungen (*SET SESSION*) festlegen. Mit *SET GLOBAL query_cache_size = 1000000* wurde bspw. die interne Query Cache-Größe auf 1.000.000 Bytes erhöht.
- Eine weitere interessante Neuerung ist die explizite Festlegung von Zeichensätzen für Tabellen und Spalten einer My-SQL-Datenbank. Auch wenn es Ihnen bis jetzt egal war, aus welchem Zeichensatz eine Tabelle oder gar eine Spalte stammt, ist dies je nach Verwendungszweck nicht immer ganz unwichtig, wenn Sie z. B. abhängig vom Zeichensatz eine Sortierung vornehmen wollen. Mittels *CONVERT()* ist es möglich, die Spaltenkodierung zu wechseln, durch *COLLATE* können Sie eine andere Sortierreihenfolge innerhalb eines *SELECT* vornehmen.
- Die Client/Server-Kommunikation wurde verbessert, indem das Binär-Protokoll hinzugefügt wurde, welches neben dem existierenden Client/Server-Protokoll als Erweiterung fungiert. Es sendet und empfängt Daten vollständig in binärer Form. Es entsteht kein Overhead durch Umwandlung von Integers zu Strings in der Client/Server-Kommunikation, wie es zurzeit der Fall ist. Die Parameter werden im binären Format zum Server geschickt und auch im selben Format zurückgeschickt, was den Datentransfer minimiert und die Geschwindigkeit erhöht.
- Ebenfalls ist es ab sofort möglich, die Kommunikation zwischen Client und Server mittels SSL (Secure Socket Layer) zu verschlüsseln und somit die Datensicherheit in MySQL weiter zu erhöhen. Mit diesem Feature werden Daten zukünftig nicht mehr im Klartext übertragen, sondern in verschlüsselter Form, welches zusätzlich den Datenaustausch zwischen Client und Server bedenkenlos möglich macht. Die Verschlüsselung benötigt ein aktuell installiertes OpenSSL-Paket [6]. Nachdem anschließend die Server und Client Keys mit den dazugehörenden Zertifikaten erstellt wurden, müssen diese nur noch in der globalen Konfigurationsdatei *my.cnf* eingetragen werden.

PHP MySQL 4.1 API

Nach den angekündigten und bereits implementierten Features in Version 4.0 und 4.1 wurde beschlossen, dass die aktuelle MySQL API (*ext/mysql*) nicht mehr weiterentwickelt wird. Aufgrund der vielen Neuerungen wurde eine neue verbesserte Extension entwickelt, welche die Vorteile der neuen Features nutzt, aber wiederum keine strikte Rückwärtskompatibilität gewährleisten muss. Somit ist es bald möglich, Features wie Prepared Statements, Transaktionsunterstützung, Replikationen etc. innerhalb von PHP zu nutzen.

Ein Ausblick auf MySQL 5.0

Noch in diesem Jahr will MySQL die Version 4.1 der Datenbank in Produktionsstatus erheben und außerdem die Version 5.0 zumindest als Alpha-Version herausbringen. Auf der diesjährigen MySQL User Conference wurde sogar schon davon gesprochen, dass dies bereits in den nächsten fünf Monaten erscheinen soll.

Version 5.0 soll dann die von einigen Entwicklern sehnlichst erwartete Fähigkeit zum Programmieren von Stored Procedures und Triggers mitbringen. Nur was sind eigentlich gespeicherte Prozeduren (Stored Procedures), von denen so oft die Rede ist. Es ist eine Reihe von SQL-Statements, die kompiliert auf dem Server gespeichert werden können. So müssen die Anfragen von Clients nicht mehr erneut ausgeführt werden, sondern können sich bequem auf die gespeicherte Prozedur beziehen. Durch diese Methodik wird eine bessere Performance erzielt, da die Anfrage nur einmal geparst werden muss. Trigger sind nichts anderes als gespeicherte Prozeduren, welche je nach Ereignis aufgerufen werden.

MySQL 5.0 befindet sich derzeit im Pre-Alpha-Stadium. Über das Versionsverwaltungsprogramm BitKeeper [8] ist es aber schon jetzt möglich, an die Quelltexte heranzukommen.

Fazit

Nach Evaluierung der meisten Neuerungen innerhalb von MySQL 4.1 stößt man natürlich des Öfteren auf kleinere Fehler, welche aber für eine Alpha-Version mehr als normal sind. Insgesamt macht MySQL 4.1 einen reifen Eindruck und besticht nach wie vor durch seine erstklassige Performance. Sicherlich bleibt jetzt abzuwarten, wann MySQL 4.1 als stabil deklariert wird und sich somit in der Praxis bewähren kann. Trotzdem gewinnt man den Eindruck, dass MySQL nach und nach auch den ausgefallensten Wünschen gerecht wird.

Eines der Hauptziele von MySQL AB wird es langfristig sein, den vollen ANSI SQL 99-Standard zu unterstützen und somit zu kommerziellen Datenbanken wie bspw. Oracle aufschließen zu können. Mit den Eigenschaften der kommenden Versionen 4.1 und 5.0 wird man dem wohl Rechnung tragen. Spätestens mit Version 5.0 beziehungsweise 5.1, in der Views angekündigt sind, sollte MySQL alle Eigenschaften besitzen, die von manchen Anwendern bis heute noch vermisst werden.

Andre Gildemeister (ag@inocreation.com) ist selbstständiger Softwareentwickler (www.inocreation.com/) und befasst sich verstärkt mit der Softwareentwicklung im webbasierten Umfeld und dessen Technologien. Des Weiteren ist er als freier Autor für verschiedene Online- und Print-Magazine rund um PHP und Webtechnologien aktiv tätig.

Links & Literatur

[1] Offizielle MySQL Homepage: www.mysql.com/
[2] MySQL 4.1 Alpha-Release: www.mysql.com/downloads/mysql-4.1.html/
[3] OpenGIS Consortium: www.opengis.org/
[4] PEAR::DB – Abstraktion für Datenbankfunktionen: pear.php.net/package-info.php?pacid=46/
[5] Perl DBI – Generic Database Interface: search.cpan.org/author/TIMB/DBI-1.37/DBI.pm/
[6] OpenSSL – Open Source Toolkit für SSL/TLS: www.openssl.org/
[7] Neue PHP MySQL-Extension: cvs.php.net/cvs.php/php4.fubar/ext/mysqli/
[8] Checkouts aus dem MySQL Development Tree: www.mysql.com/doc/en/ Installing_source_tree.html/

Schnittstelle

Serverseitige Programmierung unter Advantage Database Server 7

Datenintensive Berechnungen gehören rein aus Performancegründen auf den Server ausgelagert. Aber ebenso kann dieser Server aus Gründen der Datenstabilität und -Integrität Aufgaben der Geschäftslogik in Form von Stored Procedures und Triggern übernehmen.

von Joachim Dürr

Im *Entwickler* 6/03 habe ich bereits die neuen Features des Advantage Database Server (ADS) 7 vorgestellt. Dieses Mal sollen wie angekündigt die Themen Advantage Extended Procedures eingehend betrachtet werden. Advantage Extended Procedures (AEPs) wurden auch schon in *Entwickler* 4/03 behandelt und deshalb soll an dieser Stelle nur das Arbeiten mit dem neuen Interface vorgestellt werden. Eine Besonderheit, die das neue Interface leider nicht mehr hat, ist das Bearbeiten der Ausgabetabelle, doch das alte Interface kann weiter benutzt werden.

Was bedeutet das Bearbeiten der Ausgabetabelle eigentlich genau? Nun, das Einfügen von Datensätzen wohl nicht, da ohne diese Aufgabe eine Tabelle eigentlich sinnlos ist. Ich spreche hier vom wirklichen Bearbeiten der Tabelle, also dem Löschen, Neuanlegen und Ändern der Felddefinitionen. Dies war eigentlich ungeplant und selbst die Entwicklungsmannschaft des ADS war beim Bekannt werden dieser Option überrascht. Starten wir also gleich mit einem Beispiel. Die erste AEP nach altem Interface soll je einen Eingabe- und Ausgabeparameter bekommen. Der Eingabeparameter ist vom Tap Integer, der Typ des Ausgabeparameters ist irrelevant. Wir benötigen nur einen Ausgabeparameter, dass die Ausgabetabelle überhaupt erst angelegt wird. Interface 1 sieht wie folgt aus:

```
function MyProcedure( ulConnectionID: UNSIGNED32;
        pusUserName: PChar;
        pucPassword: PChar;
        pucProcName: PChar;
        ulRecNum: UNSIGNED32;
        pucTable1: PChar;
        pucTable2: PChar ): UNSIGNED32;
        {$IFDEF WIN32}stdcall;{$ENDIF}{$IFDEF
                                LINUX}cdecl;{$ENDIF}
```

Was die einzelnen Parameter bedeuten, möchte ich an dieser Stelle nicht wiederholen. Nur auf einen möchte ich hinweisen: Die Ausgabetabelle wird hier per Namen in *pucTable2* übergeben. Dieser Tabellenname beinhaltet den kompletten Pfad zu dieser Tabelle (Listing 1).

Wird nun diese AEP in der Datenbank mit SQL Skript angelegt und ausgeführt, so wird nicht wie erwartet eine Ausgabe mit nur einem Feld zurückgeliefert, sondern eine mit drei (Abb. 1).

Sie werden sich nun sicherlich fragen, warum dieses undokumentierte (und ungeplante) Feature mir so wichtig erscheint. Bei uns taucht immer wieder die Frage auf, wie denn nun Kreuztabellenabfragen (Pivot-Tabellen) mit dem ADS realisiert werden können. Wenn die Struktur bekannt ist, ist dies mit einem mehr oder weniger aufwändigen SQL-Statement zu erlangen. Gibt es aber nun eine undefinierte Anzahl von Spalten, so kann eben solch eine Stored Procedure genutzt werden, die Abfrage selbst zu berechnen und die Struktur der Ausgabetabelle entsprechend auszugeben. Auch kann diese flexible Gestaltung der Ausgabetabellen dazu genutzt werden, eine Fremddatenbank in die eigene einzubinden. Erstellen Sie hierzu eine AEP, welche als Input-Parameter zwei Strings bekommt. Der erste könnte ein ADO Connection String sein, der zweite ein SQL-Statement. Die AEP verbindet nun über den Connection String zu einer beliebigen Datenbank und führt das übergebene SQL-Statement aus. Wird ein Cursor (also eine Datenmenge) zurückgeliefert, so wird einfach die Ausgabetabelle entsprechend erstellt und gefüllt.

AEP Interface 2

Wenn diese Interface so mächtig ist, warum benötigen wir dann überhaupt eine Modellpflege? Schauen Sie sich hierzu den Prototypen des Interface 2 einmal an und vergleichen es mit dem des Interface 1:

```
function MyProcedure( ulConnectionID: UNSIGNED32;
        hConnection: ADSHANDLE;
        pulNumRowsAffected: PUNSIGNED32 ): UNSIGNED32;
        {$IFDEF WIN32}stdcall;{$ENDIF}{$IFDEF
                                LINUX}cdecl;{$ENDIF}
```

Listing 1

```
strTable2 := StrPas( pucTable2 );
ParamsConn.ConnectPath := ExtractFileDir( strTable2 );
tblOutput.tablename := ExtractFileName( strTable2 );
[...]
//Tabelle löschen
tblOutput.DeleteTable;
//neue Tabellenstruktur definieren
tblOutput.FieldDefs.Clear;
tblOutput.FieldDefs.Add('Id', ftAutoInc);
tblOutput.FieldDefs.Add('text', ftString, 40);
tblOutput.FieldDefs.Add('text2', ftString, 40);
//Tabelle anlegen
tblOutput.CreateTable;
//Tabelle öffnen und mit Inhalt füllen
tblOutput.open;
for i:=1 to iCount do
 begin
  tblOutput.append;
  tblOutput.FieldByName('text').AsString :=
   'Hello World';
  tblOutput.FieldByName('text2').AsString :=
   'This is Record No '+ IntToStr(i);
  tblOutput.post;
 end;
tblOutput.close;
```

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter www.derentwickler.de

Dieser Prototyp benötigt nur noch drei Parameter (anstatt der sieben des Interface 1). In diesen Parametern kommt keinerlei Information über die Temporärtabellen oder den angemeldeten Benutzer vor. Übergeben wird hier in *ulConnectionID* ein eindeutiger Identifier für die Instanz der AEP und ein *AdsHandle*, genauer das Connection-Handle, welches von der Workstation zum ADS verwendet wird, um diese AEP auszuführen. Über dieses Handle stehen alle Benutzer- und Verbindungsinformationen bereit, um im Kontext der Workstation zur Datenbank zu verbinden. Dieser Kontext der Workstation kann ungemein wichtig werden, wenn Sie mit Transaktionen arbeiten (Zur Erinnerung: Der Advantage Local Server hat keinerlei Transaktionsunterstützung). Die kompletten Daten, welche innerhalb der AEP verändert werden, werden so im Transaktionskontext der Workstation verarbeitet. Wird am Client eine Transaktion zurückgenommen (Rollback), so werden auch alle Datenänderungen, welche innerhalb der AEP stattgefunden haben, mit zurückgenommen. Interface 1 bietet diese Möglichkeit nur über mehrere Umwege. Zusammenfassend kann man sagen, dass dieses neue Interface nur eingesetzt wurde, um dem Programmierer das Schreiben von AEP zu vereinfachen. Der dritte Parameter (*pulNumRowsAffected*) kann dazu verwendet werden, dem aufrufenden Programm einen Status zurückzuliefern.

Kommen wir also zu einem einfachen Beispiel. Was bietet sich bei Transaktionen mehr an, als das Beispiel auf einer kleinen Kontenverwaltung aufzubauen. Erstellen Sie sich mit dem Architect ein Data Dictionary (z B über SQL: *CREATE DATABASE entwickler*), verbinden Sie im SQL Utility auf dieses Dictionary und erstellen die benötigten Tabellen mit folgendem Skript:

```
//Tabelle BankAccnt erzeugen
create table bankaccnt(
 accnt char(8),
 name char (40),
 balance money
);

//Tabelle TransAct erzeugen
create table transact(
 accntFrom char(8),
 accntTo char(8),
 balance money,
 username char(50),
 transdate timestamp
);

// 2 Kunden (Konten) angeben
insert into bankaccnt values ('11111111', 'kunde 1',0);
insert into bankaccnt values ('22222222', 'kunde 2',0);
```

Nun erstellen Sie in Delphi über Datei|Neu|Andere eine Advantage Extended Procedure Interface 2. Sollte diese in der Objektablage nicht sichtbar sein, so könne Sie diese über die Eigenschaften dieser Objektablage einer sichtbaren Seite zuordnen. Dem Template fügen wir im Datenmodul zwei *TAdsTable*-Komponenten hinzu, welche in der Methode *MyProcedure* verwendet werden (Listing 2).

Diese AEP kann nun in das Dictionary eingebunden werden und schon können Sie mit wilden Buchungen vom einen zum anderen Konto das Transaktionsverhalten testen.

Links & Literatur

- J. Dürr, Nachgelegt; *Der Entwickler 6/2003*
- J. Dürr, Erweiterte Vorgänge; *Der Entwickler 4/2003*

Listing 2

```
function MyProcedure( ulConnectionID: UNSIGNED32;
          hConnection: ADSHANDLE;
          pulNumRowsAffected: PUNSIGNED32 ):
                                        UNSIGNED32;
          {$IFDEF WIN32}stdcall;{$ENDIF}{$IFDEF LINUX}
               cdecl;{$ENDIF} // Do not change prototype
var
 DM1 : TDM1;
 sAccntFrom: string;
 sAccntTo: string;
 dBalance: Double;
begin

 Result := AE_SUCCESS;

 {* Get this connection's data module from the
                                    session manager. *}
 DM1 := TDM1( AEPSessionMgr.GetDM( ulConnectionID ) );
 try
  with DM1 do
  begin

   //Inputparameter lesen und temporär speichern
   tblInput.open;
   sAccntFrom := tblInput.FieldByName('AccntFrom').
                                          AsString;
   sAccntTo  := tblInput.FieldByName('AccntTo').AsString;
   dBalance  := tblInput.FieldByName('Balance').AsFloat;
   tblInput.close;
   tbAccnt.DatabaseName := 'DataConn';
   tbAccnt.TableName := 'BankAccnt';
   tbAccnt.Open;
   //Betrag abbuchen
   if tbAccnt.Locate('accnt', VarArrayOf([sAccntFrom]), [])
    then begin
     tbAccnt.Edit;
     tbAccnt.FieldByName('balance').AsFloat :=
      tbAccnt.FieldByName('balance').AsFloat - dBalance;
     tbAccnt.Post;
    end
    else begin
     DataConn.Execute( 'INSERT INTO __error VALUES
                ( 1, "Quellkonto nicht vorhanden")');
    end;
   //Betrag zubuchen
   if tbAccnt.Locate('accnt', VarArrayOf([sAccntTo]), [])
    then begin
     tbAccnt.Edit;
     tbAccnt.FieldByName('balance').AsFloat :=
      tbAccnt.FieldByName('balance').AsFloat + dBalance;
     tbAccnt.Post;
    end
    else begin
     DataConn.Execute( 'INSERT INTO __error VALUES
                ( 1, "Zielkonto nicht vorhanden")');
    end;
   tbAccnt.Close;

   //und nun noch die ganze Aktion mitloggen
   tbTransAct.DatabaseName := 'DataConn';
   tbTransAct.TableName := 'TransAct';
   tbTransAct.Open;
   tbTransAct.Append;
   tbTransAct.FieldByName('AccntFrom').
                                  AsString := sAccntFrom;
   tbTransAct.FieldByName('AccntTo').
                                  AsString  := sAccntTo;
   tbTransAct.FieldByName('balance').
                                  AsFloat   := dBalance;
   tbTransAct.Post;
   tbTransAct.Close;
  end;  {* with DM1 *}

 except
  on E : EADSDatabaseError do
   {* ADS-specific error, use ACE error code *}
   DM1.DataConn.Execute( 'INSERT INTO __error
          VALUES ( ' + IntToStr( E.ACEErrorCode ) + ', ' +
                    QuotedStr ( E.Message ) + ')');
  on E : Exception do
   {* other error *}
   DM1.DataConn.Execute( 'INSERT INTO __error VALUES
                ( 1, ' + QuotedStr( E.Message ) + ')');
 end;

end;
```

Ring MY Bell

ISDN-Anwendungen mit Delphi erstellen

In der letzten Zeit ist, im wahrsten Sinne des Wortes, immer mehr von Audiotex zu hören. Televoting, Logo- oder Klingeltonversand oder auch nur einfache Auskunftsdienste haben eines gemeinsam, der gegenüberliegende Teilnehmer ist ein „einfacher" Computer. Genauso „einfach" können Sie nach dem durchlesen des Artikels eine Audiotex-Plattform in Betrieb nehmen.

von Erdal Kücük

Der vorliegende Artikel in drei Bereiche gegliedert. Der erste Teil beschäftigt sich mit den ISDN-Anschlussarten und liefert allgemeine Informationen zu ISDN. Im zweiten Abschnitt wird der Einsatz des PC als ISDN-Endgerät vorgestellt. Neben der erforderlichen Hardware wird die Kommunikation zwischen Applikation und der Hardware über das CAPI näher betrachtet. Nachdem ein ISDN-Anschluss vorhanden und der PC als Endgerät eingerichtet wurde, kann man sich nun in die Programmierung stürzen. Im dritten und letzten Abschnitt, dem Praxisteil, geht es um den Aufbau und die Programmierung eines Bestellsystems via Delphi.

Was ist ISDN?

ISDN steht für Integrated Services Digital Network. Unterschiedliche Dienste (z.B. Fax, Daten) können über ein einheitliches Netz (und gleichzeitig) übertragen werden. In Deutschland wurde bereits das gesamte Leitungsnetz digitalisiert und laut Telekom gibt es schon über 22,4 Millionen verfügbarer ISDN-Kanäle. Die Informationsdaten eines Analoganschlusses werden in der nächsten Orts Vermittlungs Stelle (OVSt) in digitale Signale umgewandelt und vice versa.

Die Deutsche Telekom stellt zwei ISDN-Anschlussvarianten zur Verfügung. Einmal den Basisanschluss (BRI: Basic Rate Interface) und den Primärmultiplexanschluss (PRI: Primary Rate Interface). Nachfolgend werden beide Anschlussvarianten näher erläutert, beginnend mit dem Basisanschluss.

Der Basisanschluss beinhaltet zwei Nutzkanäle auch B-Kanäle (Bearer) genannt mit je 64 KBit/s und einen Steuerkanal (Daten Kanal, D-Kanal) mit 16 KBit/s Übertragungsrate. Die B-Kanäle stellen reine Nutzkanäle dar und die Informationsübertragung findet ungesichert statt. Deshalb werden zur fehlerfreien Übertragung Signale im D-Kanal ausgetauscht. Über den D-Kanal werden ebenso die Rufnummern und Gebühreninformationen übertragen.

Der Basisanschluss kann als Mehrgeräteanschluss oder als Anlagenanschluss verwendet werden. Beim Mehrgeräteanschluss können an den S0 Bus des NTBA (Network Termination Basic Access) bis zu 8 Endgeräte angeschlossen werden. Beim Anlagenanschluss wird nur ein Endgerät und zwar eine ISDN TK-Anlage (Telekommunikationsanlage) angeschlossen. Der Vorteil einer TK-Anlage liegt darin, dass für Verbindungen innerhalb der Anlage keine Gebühren anfallen, da das öffentliche Netz nicht beansprucht wird.

Beim Basisanschluss bekommen Sie drei Rufnummern, sogenannte Multi Subscriber Number (MSN) zugewiesen. Eigentlich 10 aber nur drei davon sind im Basispaket enthalten. Die restlichen können gegen eine einmalige Gebühr beantragt werden. Somit können alle Endgeräte auf eine bestimmte Nummer reagieren. Diese Einstellungen kann man entweder am Endgerät selbst oder an der TK-Anlage vornehmen. Es ist auch möglich eine Nummer zwei verschiedenen Endgeräten zuzuordnen (z.B. Telefon und Fax). Das Endgerät entscheidet dann selber darüber ob es die Verbindung annimmt oder nicht und zwar anhand der Dienstkennung. Bei korrekter Einstellung wird ein Telefon kein Fax und ein Faxgerät kein Telefongespräch annehmen, doch dazu später.

Bei Mehrfachnummern wird zwischen Primär und Sekundär MSN unterschieden. Die Primär MSN wird als Rufnummer bei gehenden Anrufen übertragen und in der OVSt zur Gebührenabrechnung benutzt. Die Sekundär MSN sind bei kommenden Anrufen zur Zuweisung an die verschiedenen Endgeräte von Bedeutung.

Der PC als ISDN-Endgerät

Nachdem Sie sich für eine Anschlussvariante entschieden haben, können Sie die entsprechende Peripherie anschließen. Un-

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter
www.derentwickler.de

Primärmultiplexanschluss

Der Primärmultiplexanschluss besitzt gegenüber dem Basisanschluss 30 B-Kanäle mit je 64 KBit/s und einem D-Kanal mit 64 KBit/s. Das entspricht in etwa 2 MBit/s. Deshalb wird dieser Anschluss auch als S2M (Standardanschluss mit knapp 2 MBit/s) bezeichnet. In Europa wird dieser Anschluss als E1 (gg. T1 in den USA, Japan) bezeichnet. Der Primärmultiplexanschluss ist nur als Anlagenanschluss erhältlich.

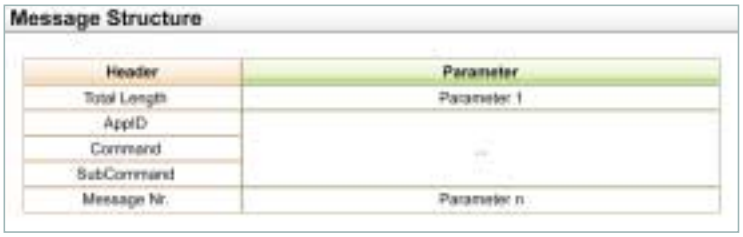


Abb. 1: Aufbau der Nachrichtenstruktur

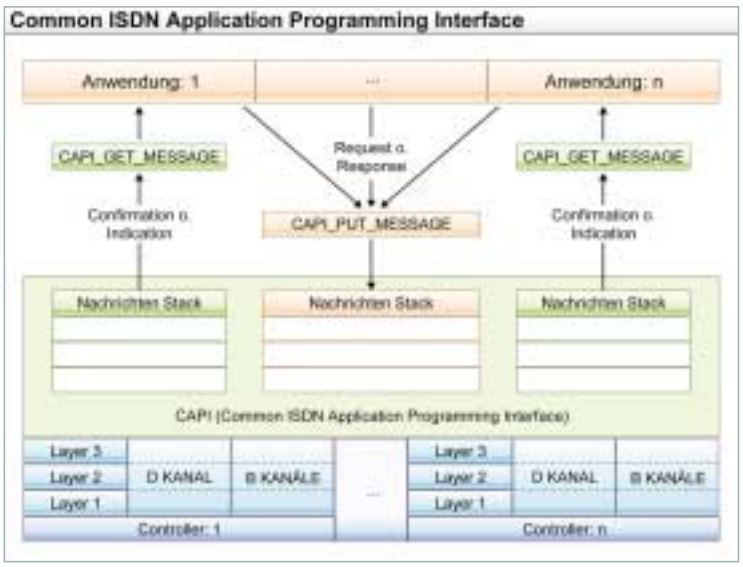


Abb. 2: Nachrichtenaustausch zwischen Anwendung und *CAPI*

ter anderem den PC. Damit der PC die Leistungsmerkmale des ISDN bedienen kann, benötigt es einen ISDN Controller.

ISDN Controller gibt es in den verschiedensten Varianten. Intern oder Extern, Aktiv oder Passiv. Für unseren Aufgabenbereich ist die Interne Variante am besten geeignet. Ob Aktiv oder Passiv entscheidet die Anzahl der Anschlüsse. Bei einem aktiven Controller ist ein Chip auf der Karte integriert, welches die Funktionssteuerung des Controllers übernimmt. Passive Controller haben keinen solchen Chip integriert, weshalb die Funktionssteuerung über die CPU des Hostsystems abgewickelt wird, was eine erhöhte Prozessorlast bedeutet und anderen Rechenoperationen umso weniger zur Verfügung steht.

Bevor Sie sich für einen ISDN Controller entscheiden, achten Sie bitte auch darauf welche ISDN-Protokolle von der Hardware unterstützt werden. Je mehr Protokolle die Hardware unterstützt desto mehr Rechenpower bleibt für die eigentliche Anwendung übrig.

Im Lieferumfang des Controllers befindet sich neben dem Hardware Treiber die CAPI (Common (ISDN) Application Programing Interface). CAPI entstand aus einem Zusammenschluss verschiedener Hersteller aus der Telekommunikationsbranche und der Dt. Telekom um eine einheitliche ISDN Standard Schnittstelle für Anwendungen zu etablieren. Dieser Standard liegt nun in der Version 2.0 in Form einer Bibliothek (32 Bit Windowssysteme – *CAPI2032.DLL*) vor.

Das CAPI sitzt oberhalb der Schicht 3 des OSI Modells und bietet die Dienste der darunter liegenden Schichten an, weshalb es die Schnittstelle den Programmierern erlaubt, ISDN-Funktionalität in Ihre Anwendungen zu integrieren ohne besondere Kenntnisse der ISDN-Protokolle. Auf das CAPI können Sie wie bei der Win32 API auch mittels Funktionen zugreifen. In der Dokumentation zum CAPI wird der Gebrauch dieser Funktionen als Operation bezeichnet. Die Funktionen lassen sich dabei in 5 Gruppen einteilen:

- Informationsabfrage
  - *CAPI_INSTALLED* (je nach verwendetem Betriebssystem)
  - *CAPI_GET_MANUFACTURER*
  - *CAPI_GET_VERSION*
  - *CAPI_GET_SERIAL_NUMBER*
  - *CAPI_GET_PROFILE*
- An- und Abmeldung der Applikation
  - *CAPI_REGISTER*
  - *CAPI_RELEASE*
- Nachrichtenaustausch
  - *CAPI_GET_MESSAGE*
  - *CAPI_PUT_MESSAGE*
- Benachrichtigung
  - *CAPI_SET_SIGNAL* (je nach verwendetem Betriebssystem)
  - *CAPI_WAIT_FOR_SIGNAL* (je nach verwendetem Betriebssystem)
- Herstellerspezifische Dienste
  - *CAPI_MANUFACTURER* (je nach verwendetem Betriebssystem)

Die Kommunikation zwischen der Anwendung und dem CAPI findet in Form von Nachrichten statt (*CAPI_PUT_MESSAGE, CAPI_GET_MESSAGE*). Hierbei werden von der Anwendung zum *CAPI* die OSI Primitivtypen *Request* (Anfrage) oder *Response* (Antwort) verwendet. In umgekehrter Richtung *Confirmation* (Bestätigung) oder *Indication* (Hinweis).

Die Nachrichten bestehen aus einem Header fester Länge und einem Parameterbereich. Die Nachrichten werden fortlaufend nummeriert, womit die richtige Zuordnung der Nachrichten sichergestellt wird. Für den Nachrichtenaustausch stellt das CAPI der Anwendung einen Stack bereit, welches nach dem FIFO (First-In-First-Out) Prinzip organisiert ist. Dabei ist zu beachten, dass alle Anwendungen in Richtung CAPI einen gemeinsamen Stack benutzen aber jede Applikation Ihre entsprechende Nachricht über einen eigenen Stack aus der Richtung CAPI erhält. Nachrichten werden logisch in drei Arten gruppiert:

- Auf- und Abbau von Verbindungen (D-Kanal).
- Nutzung der logischen Kanäle (B- oder D-Kanal).
- Verwaltung, Administration.

Die Verwendung des *CAPI* läuft nach folgendem Schema ab:

- Anmelden der Anwendung am CAPI (*CAPI_REGISTER*)
- CAPI auf Ereignissteuerung schalten (*CAPI_SET_SIGNAL, CAPI_WAIT_FOR_SIGNAL*)

- Nachrichten senden, bzw. abholen (*CAPI_PUT_MESSAGE, CAPI_GET_MESSAGE*)
- Anwendung abmelden (*CAPI_RELEASE*)

Mit *CAPI_GET_PROFILE* können Sie im Vorfeld wichtige Informationen, die Sie für Ihre Anwendung benötigen erhalten, wie z.B. Anzahl der Controller, Anzahl B-Kanäle und welche Protokolle unterstützt werden. Nähere Informationen zum *CAPI* und dessen Einsatz entnehmen Sie bitte der *CAPI*-Dokumentation [1].

ISDN-Anwendungsentwicklung in der Praxis

Bevor Sie sich an die Arbeit machen und eine *CAPI*-Komponente entwickeln wollen, will ich Sie in diesem Abschnitt mit der ISDN-Komponente von ISDN-Objects (www.isdn-objects.de) bekannt machen. Diese Komponente ist leider nicht frei erhältlich, jedoch sind alle Leistungsmerkmale von ISDN implementiert. Bevor Sie sich zum Kauf entschließen, können Sie diese Komponente mit vollem Funktionsumfang ausgiebig testen.

Die Installation der Komponente erfolgt in Delphi 6 über den Menüpunkt KOMPONENTE ? PACKAGES INSTALLIEREN. In dem erscheinenden Dialogfeld klicken Sie auf *Hinzufügen* und suchen sich die Package Bibliothek (**.bpl*) in dem Verzeichnis, in der Sie die Komponente entpackt haben. Nachdem Sie mit *OK* die Installation bestätigt haben, finden Sie in der Komponentenleiste ein neues Register vor, mit der Bezeichnung *ISDN*.

Im Folgenden werden wir mit Hilfe dieser Komponente eine komplett funktionsfähige IVR-Plattform (Interactive Voice Response) programmieren, die folgende Funktionsmerkmale aufweist:

- Wiedergeben von Audiodateien.
- Einlesen von Tastaturfolgen.
- Tastatur Menü.

Wählen Sie dazu in der Komponentenleiste das Register *ISDN* aus. Fügen Sie ihrem Formular die Komponente *ISDN Manager* hinzu. Wollen Sie, dass beim Start ihrer Anwendung sofort auf eingehende Verbindungen reagiert werden soll, so geben Sie im ObjektInspektor für die Eigenschaften *AutoListen* und *AutoRegister* die Werte *True* an. Wenn *AutoTerminate* auf *True* steht und in Ihrem System kein CAPI vorhanden ist, dann bricht die Anwendung mit einer Meldung (Eigenschaft: *TerminationMsg*) ab. Sie können jedoch auch *AutoTerminate* auf *False* setzen und in der Ereignisroutine *OnCapiNotFound* auf das nicht vorhanden sein des CAPI reagieren und die Anwendung trotzdem starten lassen.

Als nächstes fügen Sie Ihrer Anwendung die Komponente *ISDN Line* hinzu. Diese Komponente ist Ihnen bei der Auswahl der B-Protokolle behilflich indem es die Eigenschaft *BProtocolMakro* bereitstellt. In unserem Fall setzen Sie die Eigenschaft auf *bpmTELEPHONY*. Die Protokollauswahl für die Schichten 1-3 wird somit automatisch gesetzt.

Die Eigenschaft *CalledMSN* ist nur bei abgehenden Verbindungen relevant, des-

halb lassen wir sie unberührt. Wichtiger für uns ist die Eigenschaft *CallResponse*. Mit dieser Eigenschaft ist es uns möglich der Anwendung (dem Endgerät) mitzuteilen auf welche Rufnummer und welchen Dienst bei eingehenden Anrufen reagiert werden soll. Somit können Sie zwei Anwendungen (z.B. Telephonie und Fax) parallel betreiben, sodass die richtige Anwendung auf den richtigen Dienst antwortet.

CallResponse.MSN ist vom Typ *TStringList*. Sie können die Rufnummern, auf die Ihre Anwendung reagieren soll zur Laufzeit oder im String-Listen-Editor angeben. Wenn die Liste leer bleibt akzeptiert die Anwendung jeden eingehenden Anruf egal welche Rufnummer gewählt wurde. Es sei denn, die Dienste wurden entsprechend gesetzt. Die jeweiligen Dienste lassen sich unter *CallResponse.Services* einstellen. In unserem Fall setzen Sie *cipSpeech*, *cipAudio_31Khz* und *cipAudio_7Khz* auf *True*.

Die nächste wichtige Eigenschaft wäre *Manager*. Hier können Sie angeben mit welchem *ISDN Manager* die *ISDN Line*-Komponente verknüpft werden soll. Sie haben nämlich die Möglichkeit mehrere *ISDN Manager* gleichzeitig zu verwenden. Jedoch sollten Sie beachten, dass jeder *ISDN Manager* einen neuen Thread startet und die Performance Ihrer Anwendung darunter leiden könnte. Im Normalfall sollte ein *ISDN Manager* pro Anwendung genügen.

Die Eigenschaft *OwnService* sollte, nachdem Sie die B-Protokolle eingestellt haben, ebenfalls automatisch gesetzt sein. Diese Eigenschaft ist ebenfalls nur bei abgehenden Rufen relevant und teilt dem angerufenen System mit, auf welche Art Ihre Anwendung zu kommunizieren wünscht.

Als nächstes fügen Sie der Anwendung eine *ISDN Device*-Komponente hinzu. Die *ISDN Device*-Komponente übernimmt die Aufgabe der eigentlichen Kommunikation, des Datenaustausches. Der *ISDN Device*-Komponente müssen Sie nur noch unter der Eigenschaft *Line* eine *ISDN Line*-Komponente zuordnen.

Wir wollen kurz zusammenfassen: Ihre Anwendung benötigt einen *ISDN Manager*, der für die An- und Abmeldung an dem CAPI zuständig ist. Ein *ISDN Manager* kann beliebig viele *ISDN Line*-Komponenten verwalten. Jede *ISDN Manager*-Komponente startet einen neuen Thread. Eine sinnvolle Situation für den Einsatz mehrerer *ISDN Manager* wäre eine Multi Prozessor-Umgebung.

Ein Merkmal des CAPI ist die Verwendung mehrerer logischer Verbindungen innerhalb einer physikalischen Verbindung.

Modulbeschreibung

Das ganze Programm basiert auf Modulen (Klassen). Das Basisobjekt stellt *TModule* dar. In diesem Projekt kommen die Module *TAudioPlay*, zur Wiedergabe von Audiodateien; *TReadDigit*, für das Einlesen von Zeichenfolgen und *TTMenue*, zum Erstellen eines Tastenmenüs zum Einsatz deren Eigenschaften nachfolgend beschrieben werden. *TModule* stellt folgende Eigenschaften zur Verfügung:

- *ModuleId*: Eine eindeutige Id
- *ModuleName*: Name des Moduls
- *ModuleType*: Typ des Moduls
- *NextModule*: ModuleId des nachfolgenden Moduls
- *onNext*: Ereignis, welches nach Durchlauf des Moduls ausgelöst wird

Hinweis: Statt *ModuleName* und *ModuleType* könnte man genauso gut die Methoden *ClassName* und *ClassType* des Vorfahr Objekts, *TObject* verwenden. Von *TModule* werden alle anderen Module abgeleitet. Die Eigenschaften von *TAudioPlay*:

- *DTMFAbort*: Wenn *True*, werden DTMF Töne erkannt und die Wiedergabe der Audiodatei unterbrochen
- *FileName*: Dateiname der Audiodatei, welches in diesem Modul wiedergegeben wird
- *NextOnPlay*: ModuleId des nachfolgenden Moduls, nach Beendigung der Wiedergabe der Audiodatei

Das Modul *TAudioPlay* stellt noch folgende Methode bereit:

- FileSent: In dieser Methode wird der Eigenschaft *NextModule* der Wert von *NextOnPlay* zugewiesen und anschließend das Ereignis *onNext* ausgelöst

Die Eigenschaften von *TReadDigit*:

- *DigitCount*: Anzahl der Zeichen die per Telefontastatur eingegeben werden können
- *DigitsList*: String Liste, die bei mehrmaliger Verwendung des Moduls alle Strings von DTMFStr speichert
- *DTMFStr*: In diesen String werden die Zeichen der Telefontastatur nacheinander eingelesen
- *NextOnTimeOut*: ModuleId des nachfolgenden Moduls nach einem Timeout
- *NextOnReadDigit*: ModuleId des nachfolgenden Moduls, nachdem die Anzahl in DigitCount erreicht oder das Zeichen in *TerminationString* eingegeben wurde
- *TerminationString*: Nach Eingabe dieses Zeichens wird der Vorgang des Einlesens abgebrochen

Methode von *TReadDigit*:

- *StartListen*: Beim Aufruf dieser Methode wird der Timer initiiert und für den *Timeout*-Prozess gestartet

Weil das Modul *TTMenue* ebenfalls die Eigenschaften *Timeout* und *TerminationString* bereitstellt, leiten wir das Modul nicht von *TModule* ab, sondern von *TReadDigit*. Die Eigenschaften von *TTMenue*:

- *DigitCount*: Diese Eigenschaft wird auf *ReadOnly* und den Defaultwert *1* gesetzt
- *NextOnKey_0 bis NextOnKey_9* und *NextOnKey_s, NextOnKey_r* und *NextOnKey_w*, ModuleId des nachfolgenden Moduls, wenn die entsprechenden Zeichen eingelesen wurden; *NextOnKey_s*: Stern-Taste; *NextOnKey_r*: Raute-Taste; *NextOnKey_w*: Wrong, bei drücken einer falschen Taste

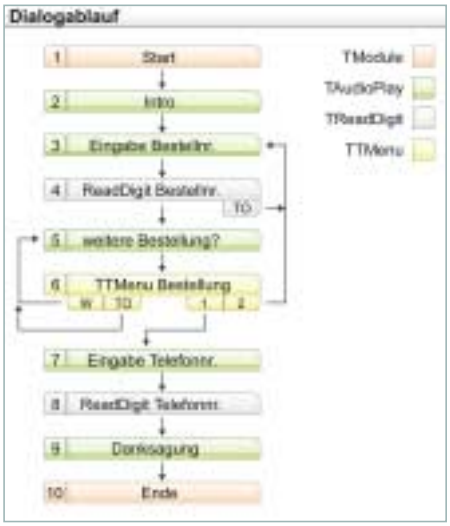


Abb. 3: Programm- bzw. Dialogablauf

ISDN Line-Komponenten stellen die physikalische, *ISDN Device*-Komponenten die logische Verbindungen dar. Einer *ISDN Line*-Komponente können somit beliebig viele *ISDN Device*-Komponenten zugeordnet werden.

Damit Sie den Dialogablauf mitverfolgen können, benötigen wir nur noch drei *Edit*-Boxen, bzw. -Labels, die als Anzeige der angerufenen Rufnummer (Zielrufnummer), die Rufnummer des Anrufenden (Ursprungsnummer) und der Meldungsausgabe, dienen (Abb. 3).

Wie Sie der Grafik vielleicht entnehmen können, kommen diverse Module zum Einsatz. Diese Module sind nicht Teil der ISDN-Komponentensammlung. Sie finden Sie in der Unit *ISDNModule.pas* auf der *Entwickler* Profi CD. Eine Beschreibung der Module können Sie dem Kasten „Modulbeschreibung“ entnehmen.

Sie müssen noch die Unit *ISDNModule.pas* in Ihr Projekt einbinden, indem Sie im *uses*-Abschnitt Ihres Formulars den Eintrag *ISDNModule* vornehmen.

Beginnen wir damit, dass wir die Module zunächst im *OnCreate*-Ereignis des Form Objekts erstellen. Für jedes Modul wird dafür eine Prozedur aufgerufen, welches das Modul mit den erforderlichen Parametern erzeugt und anschließend den Zeiger der auf dieses Modul verweist, in einem *TList* Objekt (*moduleList*) ablegt (Listing 1).

Die Anwendung ist so eingestellt, dass es auf eingehende Anrufe automatisch reagiert. Welche Events dabei ausgelöst werden, soll mit dem nächsten Codeabschnitt erläutert werden. Sobald ein Anruf hereinkommt löst die *ISDN Line*-Komponente das Ereignis *OnIncomingCall* aus. Dem Referenzparameter *Action* wird standardmäßig der Wert *rvAccept* vom Typ *TRejectValue* übergeben, sodass Sie es nicht wie hier explizit angeben müssen. Ich wollte nur darauf hinweisen, dass der Typ *TRejectValue* weitere Elemente zur Verbindungssteuerung bereithält, die es Ihnen ermöglichen die Verbindung anzunehmen, bzw. abzulehnen oder in einen Wartezustand zu versetzen (Listing 2).

Nach einer erfolgreichen Verbindung der *ISDN Device*-Komponente, wird das Ereignis *OnConnect* der *ISDN Device*-Komponente ausgelöst. Dieses Ereignis fangen wir ab, denn das ist der Zeitpunkt unser erstes Modul zu starten:

```
procedure TForm1.ISDNDeviceConnect
                              (Device: TCustomISDNDevice);
begin
 //activeModule ist ein globales Modul,
 //welches auf das derzeit aktive Modul verweist
 activeModule := TModule(moduleList.Items[0]);

 ModuleOnNext(activeModule);
end;
```

Listing 1

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 moduleList := TList.Create;
 AddModule(0, 'Start', 1);
            //Startmodul immer als erstes Modul erzeugen
 AddModule(9, 'End', -1);
              //Endmodul, property NextModule immer -1
 AddModuleAudioPlay(1, 'Intro', 'audio\intro.wav', True, 2);
 ...
 AddModuleReadDigit(3, 'RDBestellNr', '*#', 6, 2, 4);
 ...
 AddModuleTMenue(5, 'TMBestellNr', 4,-1,2,6,-1,-1,-1,
                                        -1,-1,-1,-1,4,4,4);
end;

procedure TForm1.AddModule(id: Integer; name: string;
                                              next: Integer);
var module: TModule;
begin
 module := TModule.Create(self);
 module.ModuleId := id;
 module.ModuleName := name;
 module.ModuleType := 'TModule';
 module.NextModule := next;
 module.onNext := ModuleOnNext;
 moduleList.Add(module);
end;

procedure TForm1.AddModuleAudioPlay(id: Integer;
      name: string; filename: string; dtmfAbort: Boolean;
                                    nextOnPlay: Integer);
var module: TAudioPlay;
begin
 ... -> TModule
 module.FileName := filename;
 module.DTMFAbort := dtmfAbort;
 module.NextOnPlay := nextOnPlay;
end;

procedure TForm1.AddModuleReadDigit(id: Integer;
        name: string; termStr: string; digitCount: Integer;
    nextOnTimeOut: Integer; nextOnReadDigit: Integer);
var module: TReadDigit;
begin
 ... -> TModule
 module.TerminationString := termStr;
 module.DigitCount := digitCount;
 module.TimeOut := 5;
 module.NextOnTimeout := nextOnTimeOut;
 module.NextOnReadDigit := nextOnReadDigit;
end;

procedure TForm1.AddModuleTMenue(id: Integer;
  name: string; nextOnTimeOut: Integer; t0,t1,t2,t3,t4,t5,
                                  t6,t7,t8,t9,ts,tr,tw: Integer);
var module: TTMenue;
begin
 ... -> TModule
 module.NextOnKey_0 := t0;
 module.NextOnKey_1 := t1;
end;
```

Listing 2

```
procedure TForm1.ISDNLineIncomingCall
  (Line: TCustomISDNLine; UUSDataReceived: String128;
                              var Action: TRejectValue);
begin
 Action := rvAccept;

 //welche Rufnummer wurde gewählt
 zielnr.Text := Line.CallerParams.CalledMSN.Number;

 //welche Rufnummer hat der Anrufer
 ursprungsnr.Text := Line.CallerParams.CallingMSN.Number;

 //entsprechendes Device starten; nur dann aufrufen
wenn mehrere Device Objekte mit dem
 entsprechenden ISDN Line Objekt verknüpft sind.
 ISDNDevice.Connect;
end;
```

TModule löst nach dem Durchlaufen des Moduls das Ereignis *onNext* aus. In dieser Prozedur wird anhand der Eigenschaft *NextModule* des derzeit aktiven Moduls (*activeModule*), aus der Modulliste (*moduleList*) das entsprechende Modul ermittelt. Nach Übereinstimmung der ModuleId wird die Objekt Referenz des ermittelten Moduls *activeModule* zugewiesen:

```
for i:=0 to moduleList.Count - 1 do
begin
 module := TModule(moduleList.Items[i]);
 //bei übereinstimmung der IDs
 if activeModule.NextModule = module.ModuleId then
 begin
  //setzen des aktuellen Moduls
  activeModule := module;
  break;
 end;
end;
```

Welches Modul als nächstes durchlaufen werden soll haben wir ermittelt. Als nächstes müssen wir feststellen ob die Eigenschaft *NextModule* des ermittelten Moduls größer -1 ist, falls dies nicht der Fall sein sollte haben wir das Ende des Dialogs erreicht und brauchen den nachfolgenden Code nicht mehr auszuführen. Mit der Methode *Disconnect* der *ISDN Device*-Komponente wird anschließend das Ende der aktiven Verbindung einleiten.

Mit der Eigenschaft *ModuleType* ermitteln wir den Typ des aktuellen Moduls. Handelt es sich um *TAudioPlay*, sagen wir der *ISDN Device*-Komponente, dass es die in der Eigenschaft *FileName* notierte Audiodatei wiedergeben soll. Die Audiodatei muss als Windows Wave-Datei (8 KHz, 16 Bit, Mono) vorliegen:

```
if activeModule.ModuleType = 'TAudioPlay' then
begin
 audioPlay := (activeModule as TAudioPlay);

 if audioPlay.DTMFAbort then ISDNLine.StartDTMFListen
 else ISDNLine.StopDTMFListen;

 ISDNDevice.SendWave(audioPlay.FileName,True);
end;
```

Falls *activeModule* vom Typ *TReadDigit* ist rufen wir die Methode *StartDTMF Listen* der *ISDN Line*-Komponente auf.

```
if activeModule.ModuleType = 'TReadDigit' then
begin
 readDigit := (activeModule as TReadDigit);
 readDigit.StartListen;

 ISDNLine.StartDTMFListen;
end;
```

Analog zu *TReadDigit*, das Modul *TTMenue*:

```
if activeModule.ModuleType = 'TTtMenue' then
begin
 ttMenue := (activeModule as TTtMenue);
 ttMenue.StartListen;

 ISDNLine.StartDTMFListen;
end;
```

Für den Start der Module wäre gesorgt, jetzt müssen wir nur noch die Ereignisse abfangen, welche ausgelöst werden, wenn die entsprechenden (Re-)Aktionen seitens des Anrufers erfolgen.

TReadDigit und *TTMenue* erwarten, dass der Anrufer eine Taste drückt. Dafür stellt die *ISDN Line*-Komponente das Ereignis *OnDTMFReceived* bereit. Dieses Ereignis wird bei jedem Tastendruck aufgerufen, der Parameter *Digits* übergibt den Wert als *String* der aktuell gedrückten Taste auf der Telefontastatur (Listing 3). Nach der Wiedergabe einer Audiodatei löst die *ISDN Device*-Komponente das Ereignis *OnFileSent* aus. In dieser Ereignisprozedur rufen wir nur noch die Methode *FileSent* von *TAudioPlay* auf:

```
procedure TForm1.ISDNDeviceFileSent
                (Device: TCustomISDNDevice; Info: Word);
var audioPlay: TAudioPlay;
begin
 if activeModule.ModuleType = 'TAudioPlay' then
 begin
  audioPlay := (activeModule as TAudioPlay);
  audioPlay.FileSent;
 end;
end;
```

Die Anwendung kann nun Anrufe entgegennehmen und einen Dialog mit dem Anrufer führen. Den kompletten Quellcode finden Sie auf der *Entwickler* Profi CD.

Listing 3

```
procedure TForm1.ISDNLineDTMFReceived
                (Line: TCustomISDNLine; Digits: String);
var audioPlay: TAudioPlay;
  readDigit: TReadDigit;
  ttMenue: TTtMenue;
begin
 //wenn TAudioPlay, dann Wiedergabe beenden
 if activeModule.ModuleType = 'TAudioPlay' then
 begin
  Line.StopDTMFListen;
  ISDNDevice.StopSending(False);

  audioPlay := (activeModule as TAudioPlay);
  audioPlay.FileSent;
 end;

 //wenn TReadDigit, dann Zeichen an TReadDigit
 //weitergeben
 if activeModule.ModuleType = 'TReadDigit' then
 begin
  readDigit := (activeModule as TReadDigit);
  readDigit.DTMFStr := Digits;
 end;

 //wenn TTMenue, dann Zeichen an TTMenue weitergeben
 if activeModule.ModuleType = 'TTtMenue' then
 begin
  ttMenue := (activeModule as TTtMenue);
  ttMenue.DTMFStr := Digits;
 end;
end;
```

Fazit

Mit den hier vorgestellten ISDN-Komponenten ist es ein Einfaches, ISDN-Funktionalität in Anwendungen zu integrieren, da man sich nicht um den Nachrichtenaustausch mit dem CAPI kümmern muss. So kann man sich auf das eigentliche Problem der Applikationsentwicklung konzentrieren, was ja den Sinn und Zweck der Komponentenidee ausmacht. Wer jedoch den Preis scheut, dem bietet dieser Artikel einen kleinen Einstieg in die CAPI-Programmierung. Wenn die Grundlagen erst einmal verstanden sind, dann sollte es nicht allzu schwer sein mit Hilfe des CAPI, gewisse ISDN-Leistungsmerkmale in Ihren Anwendungen bereitzustellen.

Links & Literatur

[1] CAPI-Association: www.capi.org
- www.telekom.de
- AVM: www.avm.de
- Hermstedt: www.hermstedt.de
- Gerdes AG: www.gerdes-ag.de
- ISDN-Objects: www.isdn-objects.de
- Crealog: www.crealog.de
- TE-Systems: www.te-systems.de
- www.auerswald.de/de/support/ isdnlexi/start.htm

Delphi serviert PDF

PDF-Berichte mit Delphi als ISAPI-DLL erstellen

CGI-Anwendungen, die PDF-Berichte dynamisch im Web erstellen, sind für eine kleine Arbeitsgruppe gut und schön. Aber der Lauf der Dinge ist bekannt: Mehr Benutzer rufen die als PDF publizierten Datenbankberichte ab, der Server muss mehr Anfragen bearbeiten und die Datenbankzugriffe nehmen stetig zu. Kurz, die Anforderungen an PDF-Berichte im Web steigen. Kein Märchen aus der Internet-Boom-Zeit, sondern wohl eher ein Blick in die Realität von etablierten Websites und größeren Intranets, die an Umfang und Besucherzahlen zunehmen.

von Timo Göbel

Im *Entwickler* 6/03 standen die Grundanforderungen an dynamisch generierte PDF-Berichte im Vordergrund. Ein kleines Projekt hat demonstriert, wie mit QuickReport, den Gnostice Export-Komponenten und Delphi Enterprise eine CGI-Anwendung relativ einfach erstellt werden konnte [1]. Für überschaubare Anwendungsbereiche mit wenigen Benutzern ist diese Lösung in der Regel ausreichend. Was jedoch, wenn komplexere Berichte von vielen Benutzern abgerufen werden oder einfach die Serverlast reduziert werden muss? Eine Lösung für dieses Szenario mittels ISAPI-Server-Erweiterungen wird im vorliegenden zweiten Teil gezeigt. Dieser Artikel verzichtet größtenteils auf die zuvor verwendeten Step-By-Step-Anweisungen, liefert mehr Hintergrundwissen und geht auf die Tücken der ISAPI-Programmierung ein. Damit richtet sich dieser Teil mehr an fortgeschrittene Programmierer, die durch ISAPI ihre bestehenden Webprojekte ausbauen möchten.

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter
www.derentwickler.de

Probleme und Lösungen

CGIs (Common Gateway Interface) sind Skripte oder Anwendungen, die über einen Skriptinterpreter bzw. als eigenständige EXE ausgeführt werden. Mit Delphi werden grundsätzlich nur CGI-Anwendungen erstellt. Ein Beispiel für CGI-Skripte wäre z.B. ein Perl-Skript, welches erst durch den Perl-Interpreter zu einer ausführbaren Anwendung wird. Ein Blick auf das allgemeine CGI-Modell (Abb. 1) zeigt das grundlegende Problem mit CGIs: Sobald der Webserver eine Anforderung (Request) von einem Benutzer (auf ein CGI) erhält, startet der Server jedes Mal eine neue Instanz der CGI-Anwendung und geht für diese Anfrage in Wartestellung. Jede Instanz der CGI-Anwendung [1] wird somit erneut in den Hauptspeicher geladen und initialisiert. Diese Anwendung baut dann eine Verbindung zur Datenbank auf und bereitet den angefragten Bericht vor. Anschließend sendet die den fertigen Bericht zurück an den Webserver, der diesen dann mit einem entsprechenden HTTP-Header versehen, an den Benutzer ausgibt (Response). Gleichzeitig wird die CGI-Anwendung entladen und der belegte Speicher wird wieder freigegeben.

Greifen nun mehrere Benutzer gleichzeitig mit ihren Requests auf eine CGI-Anwendung zu, werden die Ressourcen des Webserver alles andere als geschont. Jede CGI-Instanz belegt Speicher und fordert Rechenleistung, je nach Komplexität des auszugebenden Berichts, teilweise im zweistelligen MB-Bereich bei entsprechender CPU-Auslastung.

Durch den Einsatz einer verwandten Technologie kann dieses Problem umgangen werden: Die Programmlogik der CGI-Anwendung kann direkt in den Webserver verlegt werden. Der Name dieser Lösung nennt sich Internet Server Application Programming Interface, kurz ISAPI. ISAPI ist eine Programmierschnittstelle (API), welche ursprünglich von Microsoft zur Programmierung von Erweiterungen des Internet Information Servers bzw. Services (IIS) zur Verfügung gestellt wurde. Neben dem IIS gibt es andere Webserver-Produkte, die ebenfalls eine ISAPI-Schnitt- stelle zur Verfügung stellen, so z.B. Sambar, Falcon oder die bereits in [1] genannten Omni-HTTP oder Xitami.

Abb. 1: Das CGI-Modell: *n* Anfragen resultieren in *n* CGI-Instanzen

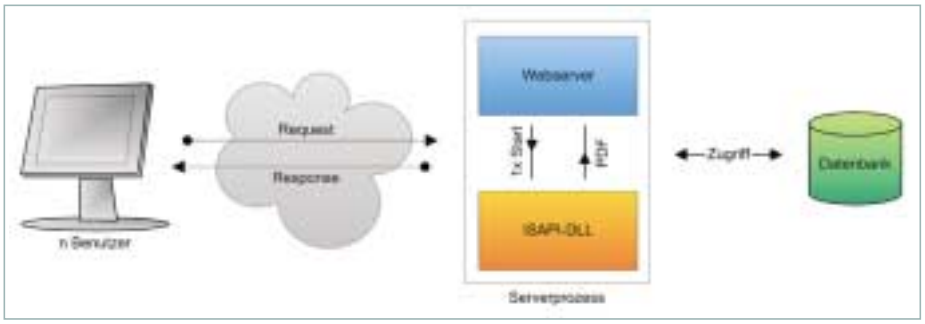


Abb. 2: Das ISAPI-Modell: Eine DLL als Webserver-Erweiterung bearbeitet *n* Anfragen

Ein ISAPI-Programm ist keine eigenständige Anwendung sondern eine Webserver-Erweiterung, die als Dynamic Link Library (DLL) vorliegen muss (Abb. 2) und mit dem Webserver über eine festgelegte Schnittstelle kommuniziert. Eine ISAPI-DLL läuft in der Regel im gleichen Prozess und Speicherraum wie der Webserver. Der große Vorteil ist dabei offensichtlich: Die DLL wird nur ein einziges Mal in den Hauptspeicher geladen, wodurch die Startzeit bei weiteren Anfragen verkürzt und die Speicherbelegung minimiert wird. Jede weitere Anfrage startet lediglich einen neuen Thread in der gleichen Instanz der DLL. Einmal geladen, bleibt die DLL im Speicher, bis der Webserver sie aus dem einen oder anderen Grund selbstständig entlädt oder der komplette Webserver beendet wird. Damit wächst natürlich auch die Verantwortung für den Entwickler, so müssen z.B. Speicherlecks von vornherein ausgeschlossen werden. Speicherlecks in einer ISAPI-DLL, welche permanent im Speicher bleibt und nur sehr selten entladen wird, wirken sich ansonsten verheerend auf die Stabilität des Webservers oder des ganzen Systems aus.

Webserver-Erweiterungen mit Apache

Seit Delphi 6 unterstützt *Delphi Apache Shared Modules* als Erweiterung für den Apache Webserver – quasi die Entsprechung der ISAPI-DLLs. Delphi 7 bietet im Gegensatz zu Delphi 6 auch eine Unterstützung für Apache Version 2.x. Die Entwicklungsschritte für ein Apache MOD sind denen von CGIs oder ISAPI-DLLs dank Delphi-Experten sehr ähnlich. Der Experte für Webserver-Anwendungen liefert auch für Apache MODs wieder den Rumpf der Anwendung. Das zugehörige Webmodul wird genauso programmiert, wie bei ISAPI- oder CGI-Anwendungen. Da Apache auch unter Linux verfügbar ist, können mit Delphi und Kylix Apache-MODs genauso wie CGI-Anwendungen plattformübergreifend entwickelt werden, entsprechende Drittherstellerkomponenten vorausgesetzt.

Von der Theorie zur Praxis

Aus dem ISAPI-Modell lassen sich für die Praxis einige wichtige Anforderungen ableiten:

- ISAPI-Anwendungen liegen als DLL vor.
- Eine standardisierte Schnittstelle regelt die Interaktion zwischen DLL und Webserver.
- Da mehrere Instanzen der DLL als eigener Thread innerhalb des Webservers laufen, muss die Anwendung Multithreading unterstützen und natürlich threadsicher sein.

Um die ersten beiden Anforderungspunkte muss sich ein Delphi-Entwickler keine Sorgen machen. Delphi generiert für ISAPI-Projekte automatisch eine Library mit den benötigten Schnittstellen, namentlich die Exports Funktionen *GetExtensionVersion* und *HttpExtensionProc* (weitere Informationen und Alternativen hierzu unter [2]). Der dritte Punkt wiegt dagegen schon schwerer, da Multithreading-Anwendung entsprechendes Know-how und etwas Planung erfordern.

Ein kurzer Blick auf die zugrunde liegende Projektbeschreibung [1] soll eventuelle kritische Punkte beleuchten: Das Beispielprojekt bereitet aus den in einer Datenbank gespeicherten Kundendaten eine PDF-Liste auf und publiziert diese dynamisch als PDF-Stream direkt in einen Browser. Der Datenbankzugriff auf die Tabelle *customers* aus der Delphi-Beispieldatenbank *dbdemos.mdb* wird mittels ADO realisiert. An Komponenten kommen QuickReport und Gnostice QuickReport Export zum Einsatz.

Die genannten Komponenten sind für Multithreading-Anwendungen geeignet. Sowohl QuickReport als auch Gnostice QuickReport Export sind in einer Multithreading-Umgebung einsetzbar. Die Datenbank (Access) muss Daten nur lesend zur Verfügung stellen und sollte somit auch keine Probleme bereiten. Und auch ADO ist, richtig angewendet, threadsicher.

Die Abkürzung ADO (ActiveX Data Objects) trägt es allerdings bereits im Namen: ADO setzt auf OLE DB und damit COM auf. COM-Objekte müssen mit *CoInitialize* bzw. *CoInitializeEx* ihr Apartment anmelden [4], [5]. So lässt sich der häufig im Zusammenhang mit ADO und ISAPI genannte [6] Interne Serverfehler

Listing 1

***CoInitializeEx* im *OnCreate-Event* des Webmoduls**

```
uses
 ComObj, ActiveX;

procedure TwmMain.WebModuleCreate(Sender: TObject);
begin
 CoInitializeEx (nil, COINIT_MULTITHREADED);
end;

procedure TwmMain.WebModuleDestroy
                                    (Sender: TObject);
begin
 CoUninitialize;
end;
```

Listing 2

***CoInitializeEx* in der *Unit ComInit*, die an erster Stelle im Projekt eingebunden wird**

```
unit ComInit;

interface

uses
 ActiveX;

implementation

initialization
 CoInitializeEx (nil, COINIT_MULTITHREADED);

finalization
 CoUninitialize;
end.
```

500 mit der Fehlermeldung *EOleSysError: CoInitialize has not been called* verhindert.

Wo bzw. wann diese Aufrufe innerhalb einer ISAPI-DLL gemacht werden sollten, ist leider nicht ganz eindeutig. Das Problem wurde zu unterschiedlichen Zeiten in unterschiedlichen Quellen auf die eine oder andere Art gelöst. Es gibt mindestens zwei unterschiedliche Implementierungen für den *CoInitializeEx*-Aufruf innerhalb eines ISAPI-Projektes.

Listing 1 zeigt den Einsatz innerhalb des *OnCreate* Ereignisses des Webmoduls. Diese Methode funktioniert in der Regel zuverlässig, so z.B. mit Delphi 7 und IIS 5.1 unter Windows XP Professional und Windows 2000 Server. Einigen Berichten im Usenet zu Folge hat sich die in Listing 2 beschriebene Methode von [6] z.B. beim IIS 6 des Windows Servers 2003 als die bessere Wahl herausgestellt. Bei diesem Verfahren wird eine kleine Unit *ComInit.pas* erstellt, welche anschließend in der *uses*-Anweisung des ISAPI-Projektes (also in der *pdfberichtdll.dpr*) an erster Stelle eingefügt wird.

Eine abschließende Bemerkung zum Threading innerhalb von ISAPI-DLLs. Auf BPL-Packages sollte unbedingt verzichtet werden. Zwar lässt sich die Dateigröße der erstellten Datei mit Packages teilweise erheblich reduzieren, jedoch kommt es immer wieder zu Fehlern, wenn z.B. zwei ISAPI-DLLs auf einem Server ausgeführt werden oder Virenscanner im Einsatz sind.

Listing 3

***pdfberichtdll.dpr*: Ärger mit *ISAPIThreadPool* (hier Delphi 7)**

```
library pdfberichtdll;

uses
 ActiveX,
 ComObj,
 WebBroker,
// ISAPIThreadPool,  // ist an dieser Stelle zwecklos
 ISAPIApp,
 ISAPIThreadPool,
                // zum Aktivieren nach ISAPIApp aufrufen!
 wMainISAPI in 'src\wMainISAPI.pas'
                              {wmMainISAPI: TWebModule},
 qKundenlisteISAPI in 'src\qKundenlisteISAPI.pas'
                                    {qrKundenlisteISAPI};

{$R *.res}

exports
 GetExtensionVersion,
 HttpExtensionProc,
 TerminateExtension;

begin
 CoInitFlags := COINIT_MULTITHREADED;
 Application.Initialize;
 Application.CreateForm(TwmMainISAPI,
                                          wmMainISAPI);
 Application.Run;
end.
```

Erneut ans Werk

Die Umsetzung des Beispielprojektes ist ansonsten ähnlich mit dem der CGI-Anwendung aus [1]. In der Delphi IDE wird eine neue Webserver-Anwendung generiert, diesmal natürlich als ISAPI/NSAPI-DLL. Der Delphi-Experte erzeugt ein neues Library-Projekt mit dem erforderlichen Exports-Bereich (Listing 3).

Allerdings patzt der Experte leider seit Delphi 6 hierbei gewaltig: Wie an anderer Stelle bereits angemerkt, [8], [9], muss die Reihenfolge der automatisch eingefügten Units geändert werden. Die Unit *ISAPIThreadPool* ist *vor* ihrer Schwesterunit *ISAPIApp* eingebunden, ziemlich wirkungslos. Beide Units enthalten nämlich die ISAPI Export-Funktionen und Delphi verwendet bekanntlich immer nur die Deklarationen der zuletzt aufgerufenen Unit. Die Reihenfolge der Units sollte also tunlichst getauscht werden.

Der ISAPI Thread Pool ist übrigens ein seit IIS 5 unter Windows 2000 verfügbares Sammelbecken für die Thread-Erzeugung. Kommt der Thread Pool zum Einsatz, werden eingehende Anfragen durch den IIS an Hilfthreads aus dem Thread Pool (auch Workerthreads genannt) in eine Warteschlange zur Erzeugung der eigentlichen ISAPI-Instanzen weitergeleitet. Diese Hilfthreads geben dem IIS zur Bearbeitung der nächsten Anfrage eher grünes Licht (die Anfrage an die ISAPI-DLL wird bearbeitet), als dies ohne ihren Einsatz der Fall wäre. Der IIS kann sich somit früher mit der Weiterleitung der nächsten Anfrage an die DLL beschäftigen und verliert keine Zeit durch Warten auf die eigentliche Thread Erzeugung. Wie dies im Detail geschieht, klärt ein Blick in den Quellcode von *ISAPIThreadPool.pas*.

Von Borlands Codecentral kann übrigens unter [9] eine fehlerbereinigte Version dieser Datei für Delphi heruntergeladen werden. In den meisten anderen Webservern ist ein Thread Pool nicht implementiert. In diesem Fall sollte der Verweis auf diese Unit auch ganz aus den betreffenden Projektdateien entfernt werden.

Ist diese Hürde aus dem Weg geräumt, können die nächsten Schritte analog zu denen der CGI-Anwendung durchgeführt werden. Der Ausgabepfad in den Projektoptionen wird wieder auf einen gültigen Webserver-Ordner gesetzt, z.B. *c:\inetpub\scripts*. Listing 1 wird integriert und neben einer Standardaktion für *Default*-Anfragen wird die eigentliche Web-Action für die Ausgabe des PDF-Berichts aus dem CGI-Projekt übernommen (Formular und Code aus [1] können 1:1 kopiert werden). Die Gnostice *TgtQRPDFExport*-Komponente wird noch auf dem Webmodul platziert und konfiguriert. Fertig ist die ISAPI-DLL?

Vor dem ersten Kompilieren sollten noch einige Einstellungen in den *Internetinformationsdiensten* vorgenommen werden. Diese Änderungen an der IIS Konfiguration sind erforderlich, damit die DLL später nach einem weiteren Compilerlauf überschrieben werden kann und ein Debugger Zugriff auf den Serverprozess bekommt. Dazu die Managementkonsole *Internetinformationsdienste* unter Systemsteuerung/Verwaltung starten und die Eigenschaften der Website des Entwicklungsrechners aufrufen. Die benötigten Einstellungen finden sich auf der Registerkarte Basisverzeichnis, wo die Option *Anwendungsschutz* standardmäßig auf Mittel (zusammengefasst) steht. Der Anwendungsschutz bezieht sich auf den Prozess, in dem Webanwendungen ausgeführt werden. Der IIS (*inetinfo. exe*) wird in einem eigenen Prozess ausgeführt. Weitere Webanwendungen, wie z.B. die ISAPI-DLL werden standardmäßig in einem einzelnen, zusammengefassten Prozess (*DLLHost.exe*) gestartet.

Debuggen von ISAPI-DLLs

Das Debuggen von ISAPI-DLLs kann für viele ein Buch mit sieben Siegeln bleiben. Zahlreiche, teilweise recht komplizierte Möglichkeiten finden sich für den IIS 5.x im Web. Seit Delphi 6 gibt es zwar den in die IDE integrierten Webdebugger, das Debuggen in einer annähernden Produktionsumgebung ist allerdings am Besten mit dem IIS möglich.

Ist der IIS Anwendungsschutz mit *Hoch* konfiguriert und die Zwischenspeicherung von ISAPI-Dlls deaktiviert, müssen noch ein paar Schritte gegangen werden, damit ein Debuggen möglich wird. Auf www.msdelphi.com werden diese im Einzelnen unter [11] detailliert beschrieben und werden hier nur kurz wiedergegeben.

Sobald der Anwendungsschutz auf *Hoch* gestellt wurde, findet sich unter Verwaltung/Komponentendienste eine COM+-Anwendung mit dem Namen *IIS-{Standardwebsite//Root}*. Unter den Eigenschaften dieses Com-Objektes lässt sich die Anwendungs-ID *{ID}* ablesen. Sie wird für den Delphi Debugger benötigt. Außerdem muss auf der Registerseite Identität des Eigenschaftendialogs der Benutzer auf *Interaktiver Benutzer* geändert werden.

Zurück in Delphi wird unter Start/Parameter die Host-Anwendung *c:\windows\system32\dllhost.exe* und der Parameter */ProcessID:{ID}* gesetzt (alles ohne Leerzeichen und für *{ID}* muss die Anwendungs-ID des COM+- Objektes eingetragen werden). Unter Delphi 7 und Windows XP SR1 kann auf einigen Entwicklermaschinen der Parameterteil */ProcessID:* weggelassen werden. In diesem Fall wird als Parameter nur die Anwendungs-ID eingetragen. Als Arbeitsverzeichnis kann unter Delphi 7 ferner noch das entsprechende Webserver-Verzeichnis hinterlegt werden. Fertig. Keine Hacks in der Registry und keine Neustarts wie bei einigen anderen Methoden.

Mit F9 kann die ISAPI-DLL (oder vielmehr *DLLHost.exe*) nun in Delphi gestartet werden. Breakpoints und andere Vorteile des gewohnten Debuggings sind ab sofort auch für die ISAPI-DLL möglich.

Großer Nachteil dieser Methode: Der Benutzerkontext (die Identität des COM+-Objektes) wird geändert. Anders als die fertige ISAPI-DLL auf dem Produktionssystem läuft die Anwendung nun nicht mehr unter dem Konto *IWAM_<RECHNERNAME>* sondern im aktiven Benutzerkontext. Diese Änderung kann je nach Aufgabe der ISAPI-DLL und ihrer benötigten Zugriffsrechte allerdings schwerwiegende Auswirkungen haben. Daher sollte der Entwickler diesen Eingriff in die Einstellungen immer im Hinterkopf behalten und diese im Falle eines unerklärlichen Fehlers auf dem Produktionsserver auch auf dem Entwicklungssystem probehalber wieder deaktivieren.

Mit der Einstellung Anwendungsschutz kann festgelegt werden, dass Anwendungen entweder in demselben Prozess wie die des IIS (*inetinfo.exe*) oder in einem von den Web-Diensten getrennten, isolierten Prozess (*DLLHost.exe*) ausgeführt werden. Die seit IIS 5.0 vorhandene dritte Option *Mittel* startet einem zusammengefassten Prozess und damit eine weitere Instanz von *DLLHost.exe*. Die verschiedenen Optionen bieten unterschiedlich hohen Schutz für Situationen, in denen eine fehlerhafte Webanwendung ausgeführt wird und dazu führt, dass der Prozess, in dem die Anwendung ausgeführt wird, abstürzt.

In einer Produktionsumgebung muss zwischen der Leistung und dem Maß an Anwendungsschutz abgewogen werden. Anwendungen, die im IIS-Prozess *inetinfo. exe* ausgeführt werden, bringen eine wesentlich höhere Leistung mit sich, jedoch auch das Risiko, dass es aufgrund einer fehlerhaften Anwendung zu einem Ausfall des gesamten Webservers kommt. Auf dem Entwicklungsrechner sollte die Einstellung *Hoch (isoliert)* gewählt werden. Durch diesen Wert werden Webanwendungen als isolierte Prozesse (neue Instanz von DLLHost) gestartet.

Eine weitere Einstellung sollte an gleicher Stelle geändert werden. Nach einem Klick auf den Button *Konfigurieren…* sollte die Option *ISAPI-Anwendungen zwischenspeichern* auf dem Entwicklungsrechner deaktiviert werden. ISAPI-DLLs werden – wie oben ausgeführt – im IIS geladen und zwischengespeichert, sodass weitere Anfragen ausgeführt werden können, ohne dass die Anwendung erneut angefordert werden muss. Die meisten ISAPI-Anwendungen (einschließlich Active Server Pages) nutzen diesen Zwischenspeicher. Für die Fehlersuche in ISAPI-Anwendungen sollte diese Option deaktiviert bleiben.

Vorteil der geringeren Leistung durch diese Einstellungen in einer Entwicklungsumgebung ist die Tatsache, dass ISAPI-Datei selbst bei einem Absturz nach einem einfachen Neustarten des Webservers (Dauer ca. 1-3 sec) in den *Internetinformationsdienste* überschrieben werden kann. Wird diese Einstellung nicht vorgenommen, müssen sämtliche Webdienste

in der *Dienste*-Managementkonsole neu gestartet werden (Dauer ca. 30-60 sec) bevor die DLL ausgetauscht werden kann – in Härtefällen also nach jedem Compilerlauf. Ferner sind diese Schritte Voraussetzung für das Debuggen von Server-Anwendungen (Kasten „Debuggen von ISAPI-DLLs“).

Ist nun auch dieser, nicht gerade unwichtige Aspekt geklärt, sollte nun endlich dem ersten Testlauf der ISAPI-DLL nicht mehr viel im Wege stehen. Über die Seite *localhost/scripts/pdfberichtdll.dll/* kann die Default-Action getestet werden. Erscheint die Standardausgabe korrekt, kann der eigentliche PDF-Bericht durch Hinzufügen von *kundenbericht/* in der URL aufgerufen werden. Ein erfolgreicher Test sollte mit der Anzeige eines PDF-Berichtes im Browser enden. Für eine detailliertere Anleitung oder mögliche Fehlerquellen und Lösungsansätze sei nochmals auf [1] hingewiesen.

Ausblick: Die Zukunft der ISAPI

Das Entwickeln von ISAPI-DLLs ist wie so vieles im Leben: Einfach, wenn man weiß, wie es geht. Sind die schärfsten Klippen mit Server-Konfiguration, Threading und dem Debugging umschifft, bieten ISAPI-DLLs eine sehr gute Performance – verglichen mit CGI-Anwendungen oder interpretierten Skripten – nicht nur bei der Erzeugung von PDF-Berichten.

Ein Blick in die Zukunft sei gewagt: Wie wird sich ISAPI im Angesicht von ASP.NET & Co. entwickeln? ISAPI ist eine Kerntechnologie, die es erlaubt, den IIS Webserver direkt um Funktionalitäten zu erweitern. ASP.NET und andere Frameworks, wie etwa IntraWeb, nutzen ISAPI intensiv. So ist ASP.NET genauso wie ASP als eine ISAPI Webserver-Erweiterung implementiert. Der Vorteil der genannten Frameworks ist genauso offensichtlich: Visuelles Design von Oberflächen und einfache Entwicklung von komplexen Anwendungen (IntraWeb) oder die Integration in das .NET Framework.

Für spezielle Aufgaben, wie etwa dem vorgestellten PDF-Datenbankbericht, eignet sich eine ISAPI Webserver-Erweiterung nach wie vor hervorragend – wenn auch vielleicht mit anderen Komponenten als dem betagten QuickReport. Weitere Grundlagen und Informationen zu ISAPI (und zu Web Services) findet der geneigte Leser auf den Seiten von Shiv Kumar unter [10].

Delphi CGI-Anwendungen

Delphi CGIs haben leider unter Windows 2003 Server einen schweren Fehler in dem Aufruf von *GetEnvironmentVariable*. Das Listing zeigt einen Fix in der Delphi Unit SysUtils, der dieses Problem behebt. Die geänderte Unit anschließend entweder im Anwendungsordner oder im Suchpfad speichern (Quelle: Usenet):

```
function GetEnvironmentVariable (const Name: string):
string;
var
 Len: integer;
 W : String; // 14.8.03 Server 2003 CGI-fix
begin
 Result := '';
 SetLength(W,1); // 14.8.03 Server 2003 CGI-fix
 Len := GetEnvironmentVariable(PChar(Name),
              PChar(W), 1); // 14.8.03 Server 2003 CGI-fix
// Len := GetEnvironmentVariable(PChar(Name), Nil, 0);
 if Len > 0 then
 begin
  SetLength(Result, Len-1);
  GetEnvironmentVariable(PChar(Name), PChar(Result),
                                              Len);
 end;
end;
```

Links & Literatur

[1] T. Göbel, Delphi serviert PDF, Teil 1; *Der Entwickler* 6/2003
[2] J. Hummel, Hallo ISAPI; *Der Entwickler* 2/2000
[4] A. Kosch, ADO und Delphi, Software & Support Verlag, 2002
[5] A. Kosch, COM/DCOM mit Delphi, Software & Support Verlag, 1999
[6] Borland Newsgroup: news://borland.public.delphi.internet.isapi-webbroker
[7] Unit ComInit.pas: www.devexpress.com/products/vcl/ewf/faq.asp#A8
[8] A. Kosch, Brückenbauer, Web Services mit Borland Delphi 7; *Der Entwickler* 3/2003
[9] S. Trefethen auf codecentral.borland.com/codecentral/ccweb.exe/listing?id=17616
[10] The Delphi Apostle, www.matlus.com
[11] Debugging mit IIS5: www.msdelphi.com/Articles/Web/DebuggingIIS/Debugging_IIS5_the_easy_way.htm

- Falcon: www.blueface.com
- OmniHTTP/Pro: www.omnicron.ca
- Sambar: www.sambar.com
- Xitami: www.xitami.net
- Gnostice QuickReport Export: www.gnostice.com
- QuSoft QuickReport: www.qusoft.com

Bericht erstatten

Rave Reporting Server

In den letzten Rave-Artikeln im *Entwickler* [1] wurde immer das Reporting speziell mit Delphi besprochen. Neben den zwei Borland-Editionen ist noch die sogenannte Standard Edition von Rave5 sowie der Rave Server von der Firma Nevrona Designs verfügbar. Im nachfolgenden Artikel soll insbesondere der Rave Reporting Server näher besprochen werden. Für dieses Produkt ist von Nevrona Designs bisher noch keine Dokumentation in PDF oder HLP-Format erhältlich.

von Thomas Pfister

Der Rave Server ist ein zentraler Speicher bzw. Bibliothek für die Archivierung und Ausführung sämtlicher Rave Reports. Durch eine Report-Kommando-Aufforderung eines autorisierten Nutzers wird der Report innerhalb des Rave Reporting Servers aufbereitet und im HTML, PDF oder dem native Rave Format NDR an den Browser weitergeleitet. Eine Nutzung ist daher sowohl im lokalen Netzwerk, Intranet, Extranet und selbstverständlich auch im Internet möglich. Aufgrund dessen, dass das Serverdesign auf Skalierbarkeit aufgebaut ist, kann er mit zunehmenden Anforderungen von wenigen bis hin zu vielen tausend Nutzern mitwachsen – sogar bis zur Ausdehnung eines Lastausgleichs über mehrere Server ist damit möglich und schon in der Praxis bewährt. Die Lizenzierung erfolgt über die Anzahl der Concurrent User innerhalb einer Minute.

Doch alles der Reihe nach. Am Anfang steht die Idee, einen Report mittels einem Internet Explorer (oder Netscape oder Opera) am Client anzuzeigen. Das bedeutet in einer korrekten Client/Server-Abhängigkeit, dass auf beiden Seiten kein Delphi- oder C++Builder-Programm aktiv ist. Und das ist ein kleiner, aber umso wichtiger Punkt damit schon bei dem Einstieg in diese interessante Materie kein grundlegender Fehler begangen wird.

Beginnen wir zuerst mit der Client-Seite. Hier ist ein Client mit einem Internet-Browser notwendig. Dieser Rechner benötigt eine IP-Verbindung zum Rave Server. Auf der Server-Seite sind die Voraussetzungen etwas enger zu definieren. Als Betriebssystem wird Windows NT4.06 bzw. Windows 2000/2003 (Server und Professional) vorausgesetzt. Windows 9x ist zwar grundsätzlich auch verfügbar, sollte jedoch, wenn überhaupt, nur im Bereich des Entwicklers zum Einsatz kommen. Der Server kann sowohl im Applikations- wie auch Dienstmodus gestartet werden:

- Applikationsmodus: Dieser Modus ist in den aktuellen Version der einzige Modus, der einen Zugriff auf die Konfigurationskonsole bietet. Es muss jedoch immer ein Benutzer angemeldet sein und die Applikation entweder im (globalen) Autostart-Ordner bzw. manuell gestartet werden.
- Dienstmodus: Mit *RaveReportServer.exe –install* bzw. *–uninstall* wird der Dienst entsprechend eingerichtet. Der Rave Server läuft nun unabhängig einer Benutzeranmeldung (nur bei Windows NT/2000/XP).

Nachfolgend sollen die Bestandteile und die Abhängigkeiten untereinander dargestellt werden; danach steigen wir in die Praxis ein. Nach Installation des Rave Servers gibt es neben den (Programm-)Dateien im Verzeichnis selber noch verschiedene Unterverzeichnisse. *RaveReportServer.exe* beinhaltet den eigentlichen Rave Server, welcher über den Applikations- oder Dienstmodus gestartet wird. Wie bereits erwähnt, ist der Konsolenaufruf derzeit nur im Applikationsmodus möglich.

Hier werden Informationen, im weiteren Verlauf noch erläuterte, über die Konfiguration angezeigt. *RaveServerConfigu-*

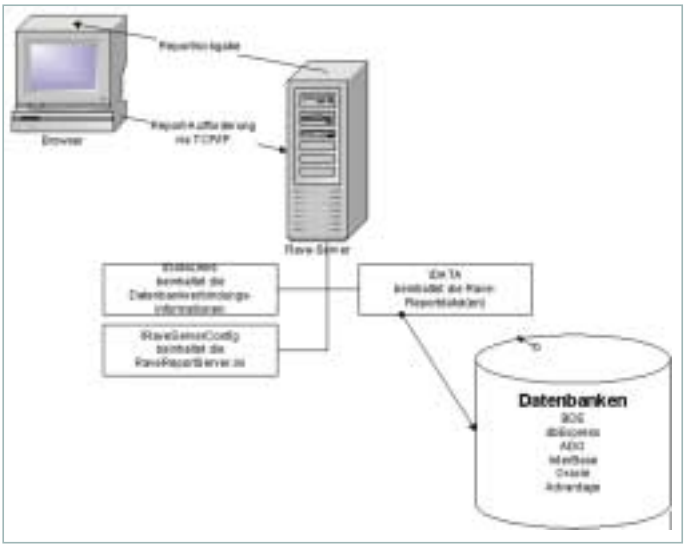


Abb. 1: Prozessablauf

rator.exe dient der Grundkonfiguration des Rave Servers. Er unterteilt sich in zwei Register: Auf der ersten Registerseite (General) wird der HTTP-Port definiert. Grundsätzlich sollte die Vorgabe von 4330 erhalten bleiben, wenn jedoch aufgrund von Firmenkonzepten oder aus Sicherheitsgründen (z.B. Firewall) andere Portbereich nur zulässig sind, so kann hier die Änderung erfolgen.

Im zweiten Bereich wird die Lizenznummer mit dem sogenannte Unlock-Code eingegeben. Die korrekte Schreibweise, d.h. ohne Leerzeichen ist notwendig, da bei einer Fehleingaben keine Fehlermeldung erscheint. Mit dem Aufruf der Konsole kann die Anzahl der maximal zulässigen Verbindungen abgefragt werden (Abb. 2), hier wären es 25 Verbindungen.

Auf der zweiten Registerseite (Packages) können neben den Standard Packages auch weitere definiert werden. Denkbar wäre beispielsweise die Installation von einem Rave-Addon von www.nevrona.com/rave/addons.shtml wie dem *GreenBarRectangle*. Diese *BPL*-Datei muss dann auf dem Rave Server kopiert und über diese Registerseite hinzugefügt werden. Sinnvollerweise sollte es im gleichen Verzeichnis wie die Standard-BPLs (d.h. im *RaveServer*-Verzeichnis direkt) abgelegt werden.

Neben diesen zwei ausführbaren Dateien befinden sich die bereits erwähnten Packages sowie eine *RaveServerAccess.Log* in dem Root-Verzeichnis des Rave Serves. Diese *Log*-Datei speichert jeden IP-Zugriff auf den Rave Server inklusive den Reportinformation. An Verzeichnissen existieren derzeit vier Stück:

- *Cache*: In diesem Verzeichnis werden temporäre Daten für die Erstellung der HTML-Seiten gespeichert.
- *Data*: In diesem Verzeichnis werden die Rave Report-Dateien (**.rav*) abgespeichert. In der Konsole können die RAV-Dateien ausgelesen werden (Abb. 2). Hier sind zwei RAV-Dateien (*Sample* und *Project1*) im Data-Verzeichnis enthalten. Die RAV-Datei *Project1* besteht aus drei Reports (rechte Seite der Abb. 2).
- *DataLinks*: In diesem Verzeichnis sind die notwendigen Dateien für die Datenbankverbindung gespeichert. Standardmäßig werden bei der Installation für folgende Verbindung die sogenannten DataLinks-Driver eingespielt: BDE, ADO, dbExpress. Auf der AddOn-Seite www.nevrona.com/rave/addons.shtml gibt es weitere kostenfreie DataLink-Driver, aktuell für Oracle, InterBase, Advantage. Es handelt sich hierbei um DLL-Dateien mit der RVD-Extension. Die Erstellung von diesen RVD-Dateien ist seitens Nevrona in einer PDF-Datei mit dem Namen *DataLinkSpec.pdf* genau beschrieben. Hierdurch können weitere Datenbankenverbindung selber entwickelt werden. Der Sinn bzw. die Notwendigkeit von diesen DataLink-Dateien wird im nachfolgend klarer.
- *RaveServerConfig*: In diesem Verzeichnis wird die Konfigurationsdatei für den Rave Server gespeichert. Auch hier werden wir im weiteren Verlauf Abhängigkeiten klar darstellen.

Hier wird auf den Artikel mit *Entwickler* 4/2003 verwiesen, der sich unter anderem die Möglichkeiten von Rave und Datenbankanbindungen befasst. Rave in der be- bzw. beX-Version ist bekanntlich mit den Borland-RAD-Tools wie Delphi, C++Builder und Kylix erhältlich. Die Verbindung zwischen Rave und den Daten erfolgt grundsätzlich über zweierlei Wege: Direct Dataview und Driver Dataview (Abb. 3 und 4).

Abb. 2: Konsole des Rave Server

Bei Direct Dataview wird immer eine Dataset-kompatible Verbindung innerhalb der Delphi-Anwendung benötigt. Dies ist bei einem Rave Server prinzipiell nicht zulässig. Wir wollen bekanntlich keine Delphi- oder C++Builder-Applikation am Server oder Client laufen haben.

Daher ist die zweite Anbindungsmöglichkeit der einzig mögliche Weg. Bei der Driver Dataview erfolgt direkt mit der jeweiligen Datenbank eine Verbindung. Konkret bedeutet dies, dass Rave selbständig mit der jeweiligen Datenbank „korrespondiert" und von dort die entsprechenden Daten erhält. Hierfür ist in der Rave-IDE die Database Connection und der Driver Dataview zuständig. Zuerst wird eine Database Connection definiert. Und genau hier sind wir im DataLinks-Verzeichnis, welche die RVD-Dateien für die jeweiligen Datenbanken enthält. Sowohl auf der Entwicklermaschine wie auch auf der Server-Maschine müssen die (möglichst versions-

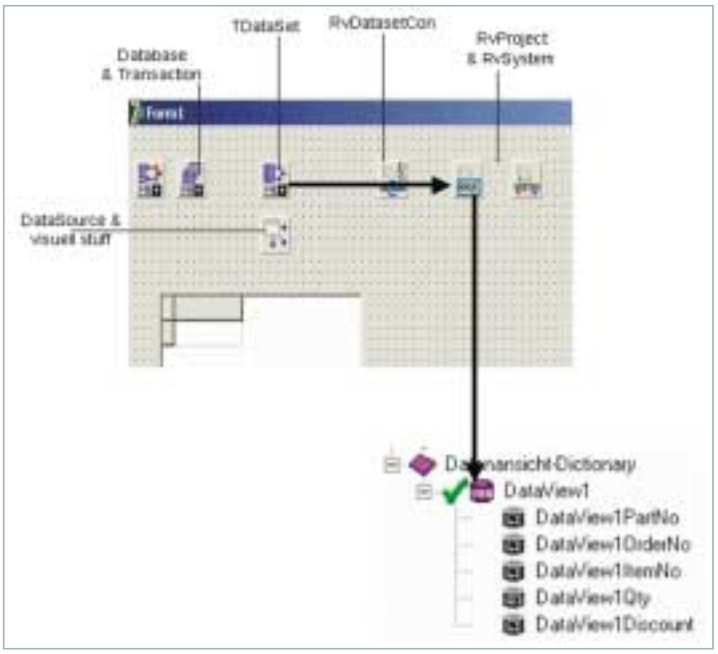


Abb. 3: Direct Dataview

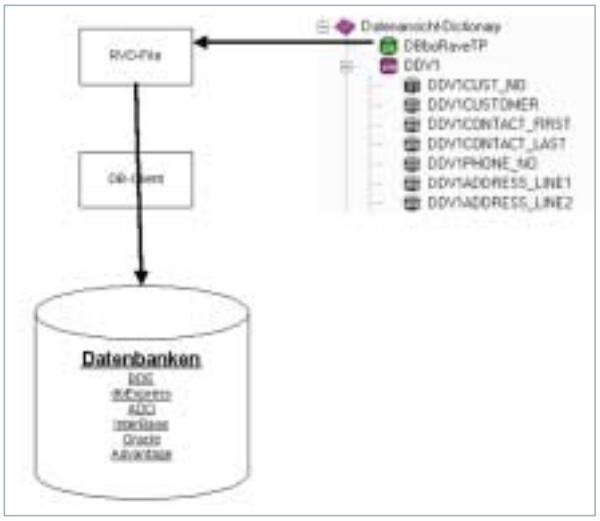


Abb. 4: Driver Dataview

gleichen) RVD-Dateien vorhanden sein. Auf der Nevrona-Seite www.nevrona.com/rave/download.html ist der Rave Server in einer Demoversion verfügbar.

Praxis

Ziel der ersten Übung soll ein Report sein, der für alle Kunden sämtliche Bestellungen mit deren Status ausdruckt. Als Datenbank möchte ich die in InterBase-Kreisen bekannte *Employee.gdb* nehmen, welche die Tabellen *Customer* und *Sales* beinhaltet. Die SQL-Abfrage soll hierbei noch sehr einfach sein:

```
SELECT T1.*, T2.*
FROM CUSTOMER T1, SALES T2
WHERE T2.CUST_NO = T1.CUST_NO
ORDER BY T1.CUST_NO, T2.ORDER_STATUS
```

Um in die Rave-IDE ohne ein in unserem Fall nicht benötigtes *RvProject* zu gelangen, ist der Designer im Pulldown-Menü *Tools* zu starten.

Im Gegensatz zu der Direct Dataview muss bekanntlich die Driver Dataview als Datenbankverbindung gewählt werden. Der Einstieg ist über die Database-Komponente zu definieren. Nach Installation der InterBase-RVD-Datei sowohl auf dem Rave Server wie auch auf dem Entwicklerrechner erscheint mit dem Auswählen der Database-Komponente neben den Standarddatenbanken BDE, ADO und dbExpress auch InterBase / IBX. Mit der Auswahl von der InterBase-Datenbank gelangt man nun in die InterBase-Verbindungseinstellungen.

Über die dritte Registerseite wird die korrekte Verbindung überprüft. Nun wird diese Database-Verbindung als Zwischenstück zwischen dem Datenbank-Client und der sogenannten Treiberdatenansicht bzw. Driver Dataview genutzt. Bei den Eigenschaften dieser Database-Komponente werden die oben getroffenen Verbindungsinformationen zweifach genutzt und zwar für den Entwicklungwie auch Ausführungsmodus. Ab sofort können die zwei Einstellungsinformationen getrennt voneinander modifiziert werden.

Selbst mit der Aufteilung zwischen Entwicklung- und Ausführungsmodus ist hier die geforderte Flexibilität nicht erkennbar da eventuell der Intranet- bzw. Webserver die Einstellungen ändern kann. Nun müsste der Entwickler die Datenbank-Information (*e:\ib_db\employee.gdb*) innerhalb der Rave-Datei modifizieren. Die oben kurz vorgestellte *raveserverconfig.ini* muss um einen zusätzlichen Abschnitt wie folgt ergänzt werden:

```
[DBRaveTP]
DataSource=192.168.200.2:e:\ib_db\employee.gdb
Options=
UserName=sysdba
Password=masterkey
```

Der Name des Abschnittes oder der Section muss mit dem Namen der Database-Komponente in Rave identisch sein. Rave prüft vor Nutzung der in *AuthRun*-Eingetragen Informationen die *ini*-Datei und nutzt zuerst die Informationen aus der *ini*-Datei. Damit kann der Server-Administrator notwendige Einstellungen in der *ini*-Datei selber treffen.

Nun muss noch die bereits im letzten Abschnitt angesprochene Driver Dataview erstellt werden. Nach dessen Auswahl wird mit der Database-Komponente (*DBRaveTP*) die Verbindung definiert und danach gelangt der Entwickler in den Query Advanced Designer. Hier kann der SQL-Befehl auf gewohnte Weise definiert werden und die entsprechenden Joins bei Nutzung mehrerer Tabellen festgelegt. Um den optimalen SQL-Aufbau bestimmen zu können, kann mit dem Button *Editor* in den Textmodus gewechselt werden und dort die Änderungen durchführen.

Nun sind die ersten Arbeiten erledigt und in dem rechten Datenbereich sollten bei der *Driver Dataview1* die gesamten Felder angezeigt werden. Als Report soll folgendes definiert werden:

- Band1
  BandStyle *GroupHeader*
  ControllerBand *DataBand1*
  GroupKey *Cust_No*
  DataText *Cust_No*
  DataText *Company*
- Band2
  BandStyle *GroupHeader*
  ControllerBand *DataBand1*
  GroupKey *Order_Status*
  DataText *Order_Status*
- DataBand1
  DataText *Date's*

Mit der F9-Taste wird der Report in der Rave-IDE gestartet und bei korrekter Report-Erstellung sollten die Daten wie gewünscht angezeigt werden. Dieser Report soll nun über das Intranet publiziert werden. Serverseitig sind nunmehr die folgenden Schritte notwendig:

- Kopieren der InterBase-RVD-Datei in das DataLinks-Verzeichnis.
- Installieren bzw. Prüfen des InterBase-Clients auf dem Server (ggf. InterBase Server auf dem Rave Server).
- Kopieren der Rave-Datei *(.rav)* in das Data-Verzeichnis.
- Aufrufen der Konsole (bei Rave Server Modus *Applikation*) und Auswahl von

Project1 in dem Project-Bereich sowie des Berichts *Report1* im Reportsbereich und abschließend Execute.
- Anzeige des Reports im Internet Explorer im HTML-Format.

Auf der Server-Seite soll bekanntlich keine Delphi-Applikation installiert sein, d.h. die Kommunikation zwischen dem Rave Server und dem Client verläuft nur im Internet Explorer. Diese Reportanforderung muss selbstverständlich bestimmten Vorgaben entsprechen. Clientseitig sind folgende Schritte notwendig:

- IP-Verbindung zwischen Client und Rave Server testen.
- Eingabe der URL des Rave Servers mit den entsprechenden Parametern.

Es gibt zwei zulässige Syntaxmöglichkeiten für die Reportanforderung:

- *http://{ip_address}:{port_number}/{[ER|GR]}/{project}/{report}[/HTML][/PDF]*
- *http://{ip_address}:{port_number}/ExecuteReport/?ND_Project={project}&ND_Report={report}[&ND_Output={[HTML|PDF|NDR]}]*

Der GR-Weg sollte für den HTML-Bereich (cached generation) und der ER-Weg für die anderen Formate (no caching) genommen werden. Derzeit werden neben HTML noch NDR und PDF unterstützt, in Zukunft werden weitere Render-Komponenten zur Verfügung stehen. Folgende Parameter nutzt der Rave Server in der URL:

- *ND_Project* – Project to generate
- *ND_Report* – Report to generate
- *ND_PageNo* – Page number to generate All or number
- *ND_Output* – Output format (PDF or HTML currently)
- *ND_Template* – HTML template to use

Template-Befehle:

- *{%Report%}* – Location in HTML where report will be inserted
- *{%PageNo%}* – Page number will be inserted
- *{%TotalPages%}* – Total pages will be inserted
- *{%MaxY%}* – Height of report in pixels will be inserted
- *{%PrevPageURL%}* – URL to go to previous page
- *{%NextPageURL%}* – URL to go to next page
- *{%FirstPageURL%}* – URL to go to first page
- *{%LastPageURL%}* – URL to go to last page

Nach diesen Vorarbeiten sollte ein Report im Internet Explorer mit den Daten entsprechend der Vorschau am Rave Server angezeigt werden.

Parametrisierte Abfrage

In dem Beispiel wurde die gesamte Tabelle angezeigt, d.h. es erfolgte keine Abgrenzung nach Kundennummer, d.h. die Abfrage soll nunmehr parametrisiert werden. Dies ist in der Rave-IDE bei der Abfrage zu definieren. Der Query-Parameter wird mehr oder weniger analog von Delphi bzw. C++-Builder eingesetzt. Ich habe das SQL-Statement wie folgt ergänzt:

```
Select
 T1.*,
 T2.*
From
 CUSTOMER T1,
 SALES T2
where
 T2.CUST_NO = T1.CUST_NO
and
T1.CUST_NO = :CUSTNO
ORDER BY
 T1.CUST_NO, T2.ORDER_STATUS
```

Der *:CUSTNO*-Parameter wird dann in der Driver Dataview im *QueryParam* wie folgt definiert:

```
CUSTNO=Param.CUSTNO('1012')
```

Bei der Ausführung in der Rave-IDE wird die Kundennummer *1012* genommen und die entsprechend selektierten Datensätze angezeigt. Die Übertragung des Parameters in der URL ist wie folgt realisiert: *192.168.200.2:4330/ExecuteReport?ND_Project=Project1&ND_Report=Report1&ND_Output=HTML&ND_PageNo=All&CUSTNO=1008&&Submit=Run+Report*. Hierdurch werden an den Rave Server die Queryparameter übergeben und der Server für die SQL-Abfrage mit den Parameter gegen die Datenbank durch und generiert die (HTML bzw. PDF)-Bericht.

Sicherheit auf Reportebene

Bei den letzten Beispielen wurde immer davon ausgegangen, dass der Client auch berechtigt ist, die Daten zu nutzen. Eine Authentifizierung fand nur zwischen dem Rave Server und dem Datenbankserver statt. Die Kommunikation zwischen Client und Server unterlag keiner Einschränkung. Ich höre jetzt schon den ein oder anderen Datenschutzbeauftragten mit dem Aufruf, den Einsatz von Rave Server zu verbieten. Dem ist zum Glück nicht so. Denn Nevrona bietet im Bereich der Datenobjekte den *SimpleSecurityController*. Hier kann in der Eigenschaft *UserList* eine Liste von Benutzernamen und deren Passwort gepflegt werden. Die Syntax hierfür lautet: *Benutzername=Passwort*. Nunmehr muss noch die Verknüpfung zwischen dem Report und dem *SecurityController* erfolgen. Diese Eigenschaft findet sich bei *Report.SecurityControl*; Damit kann für jeden Report ein eigener *SecurityController* mit jeweils eigener Benutzerübersicht definiert werden. Bei dem Aufruf des entsprechenden Reports erfolgt die Authentifizierung über eine Eingabemaske des Internet Explorers. Erst nach Eingabe einer gültigen Kombination von Benutzernamen und Passwort erfolgt die Generierung des Reports und die Anzeige im Browser.

Ergebnis

Der Rave Server stellt eine sehr gute Lösung insbesondere im Bereich des Webreporting dar. Insbesondere die Rave5-Entwickler können sich sehr schnell in diese Art von Reporting einarbeiten und damit schnell hochwertige Berichte erstellen. Eine Webapplikation bestehend aus den zwei Produkten IntraWeb und Rave Reporting Server ist eine sehr produktive Kombination (für einige sicherlich neu: Chad Hower und Hariri Hadi waren vor zwei Jahren Mitarbeiter bei Nevrona Designs).

Links & Literatur

[1] *Der Entwickler*. 1/2003, 2/2003 und 4/2003

... aus dem *Entwickler-Forum*

Liebe Leserinnen und Leser,

In der Rubrik Tipps & Tricks stellen wir ihnen regelmäßig die interessantesten Fragen und Antworten aus unserem *Entwickler*-Forum (www.entwickler.com/forum) vor. Sie finden dort Diskussionen und Beiträge zu den verschiedensten Themen, darunter Delphi, C#, C++, Datenbanken, aber auch Internet, spezielle Tools und allgemeine Fragen werden besprochen. Und wie Sie in dieser Rubrik sehen können: Für Fragen und Probleme findet sich immer ein guter Rat!

von Andreas Kosch

Kopierstation

Frage: Ich möchte gerne in einer Stored Procedure einer MS SQL Server-Datenbank ein Image-Feld von einem Datensatz in einen anderen kopieren. Mein erster Versuch provozierte nur die Fehlermeldung „Die Datentypen *text*, *ntext* und *image* sind für lokale Variablen ungültig“. Kann mir jemand sagen, wie es richtig geht?

Antwort: Für das Umkopieren der BLOb-Daten muss man auf eine direkte SQL-Anweisung zurückgreifen, das Zwischenpuffern in einer lokalen Variable vom Typ *IMAGE* steht in der Tat nicht zur Verfügung. Das folgende Beispiel demonstriert das Kopieren von beliebig großen BLOb-Daten, dazu lege ich zuerst in der Datenbank *tempdb* die Zieltabelle *BlobTest* an:

```
USE tempdb
GO
CREATE TABLE BlobTest (
 RecID  INTEGER NOT NULL PRIMARY KEY,
 Blob  IMAGE)
GO
INSERT INTO BlobTest
 SELECT 1, q.blob
 FROM Quelldatenbank.dbo.BLOBTEST q
 WHERE q.blob_id = 1
```

Es kann nur einen geben

Frage: Wenn mein Programm gestartet wird und bereits eine andere Instanz läuft, soll die ältere Instanz beendet wird. Wie kann ich so etwas machen?

Antwort: Über die Win32-API-Funktion *FindWindowEx* sucht das Programm beim Start nach der Beschriftung des Hauptfensters. Wenn ein Fenster mit der gleichen Beschriftung gefunden wird, stellt das Programm diesem Fenster die Windows-Botschaft *WM_QUIT* über den Aufruf der Win32-API-Funktion *PostMessage* zu. Immer dann, wenn die sogenannte Message Loop (Schleife zum Auslesen der Botschafts-Warteschlange) der Anwendung die Botschaft *WM_QUIT* in der Warteschlange vorfindet, wird die Schleife verlassen und die Anwendung somit zwangsläufig beendet. Die Abbildung 1 zeigt das Ergebnis, zur Demonstration wird die Formular-Eigenschaft *Caption* erst in der Ereignisbehandlungsmethode für *OnShow* durch die String-Konstante *c_FormTitel* gesetzt. Außerdem zeigt eine *TLabel*-Instanz die aktuelle Thread-Nummer des primären Threads des Hauptfensters an, indem die Win32-API-Funktion *GetCurrentThreadID* um Mithilfe gebeten wird:

```
program Project1;

uses
 Forms, Windows, Messages,
 Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

var
 hTargetWnd : HWND;

begin
 hTargetWnd := FindWindowEx(0, 0, nil, c_FormTitel);
 if hTargetWnd <> 0 then
  PostMessage(hTargetWnd, WM_QUIT, 0, 0);
 Application.Initialize;
 Application.CreateForm(TForm1, Form1);
 Application.Run;
end.
```

Inventur

Frage: Wie kann ich mit einer Schleife alle Objekte einer externen COM-Anwendung nach ihrem Titel überprüfen? Der einfache Visual Basic-Weg über *For Each* steht ja in Delphi nicht zur Verfügung.

Antwort: Das eigene Programm muss nur das Interface *IEnumVARIANT* (Unit ActiveX) bedienen, um genau das Gleiche zu erhalten, was die Visual Basic-Anwender mit der *For Each*-Anweisung bewirken. Das folgende Beispiel ruft auf diesem Weg alle vom Microsoft Agent-Control unterstützten Drehbuch-Anweisungen ab:

```
procedure TForm1.GetAnimationList(
 aSL: TStrings;
 aAgent: IAgentCtlCharacterEx);
var
 pEnum    : IEnumVARIANT;
 vAnimName  : OleVariant;
 dwRetrieved : LongWord;
 hRes    : HResult;
begin
 aSL.Clear;
 pEnum := aAgent.AnimationNames.Enum as IEnumVARIANT;
 while (TRUE) do
 begin
```

Abb. 1: Der Vorgänger wird gekillt

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter www.derentwickler.de

```
  hRes := pEnum.Next(1, vAnimName, dwRetrieved);
  if hRes <> S_OK then
   Break;
  aSL.add(vAnimName);
 end;
end;
```

Fata Morgana

Frage: Ich habe bei einer Stored Procedure ein Datum als Übergabeparameter gewählt. Beim Aufruf im Programm bekomme ich an dieser Stelle die Meldung „Übersetzungsfehler. Der Wert liegt nicht im gültigen Bereich". Wer kann mir einen Tipp geben?

Antwort: Ich kann das Problem nicht nachvollziehen, der folgende Test mit einer InterBase 6 Dialect 3-Datenbank ist erfolgreich. Dort lege ich zuerst über die IBConsole die Tabelle *TESTTBL* an:

```
CREATE TABLE "TESTTBL"
(
 "RECID" INTEGER NOT NULL,
 "DATUM" DATE,
 PRIMARY KEY ("RECID")
);
```

Anschließend bekommt die Datenbank auch die Stored Procedure *DatumTest*, die einen neuen Datensatz anlegt und dabei den als *DATE*-Parameter übergebenen Datumswert nutzt:

```
SET TERM ^;
CREATE PROCEDURE DatumTest (RECID INTEGER, DATUM
DATE) AS
BEGIN
 INSERT INTO TESTTBL (RECID, DATUM) VALUES (:RECID,:D
                                            ATUM);
END ^
SET TERM ;^
```

Das Delphi-Programm greift dann über *TIBDatabase*, *TIBTransaction* und *TIBStoredProc* auf die Datenbank zu, wobei das Ergebnis der visuellen Konfiguration im Objektinspektor im Fall von *TIBStoredProc* wie folgt aussieht. Der *Datums*-Parameter erhält den Wert *ftDate* für die Eigenschaft *DataType*:

```
object IBStoredProc1: TIBStoredProc
 Database = IBDatabase1
 Transaction = IBTransaction1
 StoredProcName = 'DATUMTEST'
 Left = 104
 Top = 40
 ParamData = <
  item
   DataType = ftInteger
   Name = 'RECID'
   ParamType = ptInput
  end
  item
   DataType = ftDate
   Name = 'DATUM'
   ParamType = ptInput
  end>
end
```

Wenn das erledigt ist, kann ich nun im Programm über die folgenden Programmzeilen einen neuen Datensatz erfolgreich hinzufügen. Dieser neue Datensatz wird auch innerhalb der *IBConsole* angezeigt:

```
IBDatabase1.Connected := True;
IBTransaction1.Active := True;
with IBStoredProc1 do
begin
 Params[0].AsInteger := 5;
 Params[1].AsDate := Now;
 ExecProc;
end;
IBTransaction1.Commit;
IBDatabase1.Connected := False;
```

Aufpoliert

Frage: Durch einen Doppelklick auf eine *TDataset*-Nachfolgerkomponente rufe ich den Feldinspektor auf, um dort bei den Feldern zum Beispiel die Eigenschaft *DisplayFormat* zu konfigurieren. Wie kann ich dies erst zur Laufzeit realisieren?

Antwort: Wenn zur Entwicklungszeit im Feldeditor keine persistenten *TField*-Instanzen angelegt werden sollen, muss man die notwendigen Schritte zur Laufzeit nachholen. Um auf die Eigenschaft *DisplayFormat* erfolgreich zugreifen zu können, muss der Delphi-Kompiler erst durch die harte Typumwandlung nach *TFloatField* überlistet werden. Das folgende Beispiel demonstriert diese Technik:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 TFloatField(Table1.Fields[2]).DisplayFormat := '0.00
Prozent';
end;
```

Terminator

Frage: Es geht darum, dass ein Programm von einem Netzlaufwerk von mehreren Rechnern aus gestartet werden kann. Auf Knopfdruck soll man mit Hilfe einer *RPC*-Funktion (oder auch anders, wenn es geht) ein Kommando an alle Rechner bzw. an einen bestimmte Rechner schicken. Dieses Kommando soll bewirken, dass das Programm auf dem betreffenden Rechner geschlossen wird. Aber wie genau implementiere ich so etwas?

Antwort: Wenn es sich nicht um eine veraltete Windows-Version handelt und auch die Rechte für den Remote-Zugriff auf den anderen Rechner vorliegen, kann man über WMI den Prozess beenden. Unter Windows XP reicht es dazu aus, die folgende Anweisung aus dem Fenster der Eingabeaufforderung oder aus dem eigenen Programm heraus abzuschicken:

```
WMIC.EXE /NODE:ZielrechnerName PROCESS WHERE
NAME='ZuKillendeAnwendung.exe' CALL TERMINATE
```

Über diesen Weg kann jeder beliebige Prozess abgeschossen werden, für den ersten Test eignet sich der Taschenrechner (*CALC.EXE*) am besten. Bei älteren Windows-Version muss man WMI nachinstallieren und direkt mit den WMI-Klassen hantieren (nachdem in Delphi über PROJEKT | TYPBIBLIOTHEK importieren die Typbibliothek Microsoft WMI Scripting V 1.x Library in eine *.PAS*-Datei importiert wurde).

Äpfel und Birnen

Frage: Es soll ein Serverprogramm mit Anbindung an eine Datenbank entwickelt werden, auf das über eine Web-Schnittstelle zugegriffen wird. Ich hab mir schon Gedanken gemacht wie ich das quasi von Hand realisiere, und da bin ich über die Technologien wie WebSnap und SOAP gestolpert. Welche der Technologien ist für meine Zwecke sinnvoller?

Antwort: Die Frage lässt sich ganz einfach beantworten, weil es sich um grundsätzlich verschiedene Dinge handelt. Der Zugriff über SOAP (alias Web Service) betrachtet die IT-Anbindung ohne Benutzeroberfläche, d.h. zwei Programme (Server-Programm und Client-Programm) „unterhalten sich", ohne dass ein HTML-Formular im Spiel ist. SOAP ist somit eine reine Programmierschnittstelle.

Hinter dem Markennamen WebSnap verbirgt sich eine Technologie, die HTML für die Darstellung einer Benutzeroberfläche im Browser rendern kann. Allerdings ist das Ende von WebSnap schon abzusehen: Im aktuellen Borland-Papier „Octane and Delphi Q&A – by Simon Thornhill"(community.borland.com/article/0,1410,29952,00.html) ist der folgende Satz zu lesen: „Existing web development technologies will be replaced by Delphi ASP.NET WebForms and .NET WebServices". Somit wird zukünftig auch im Delphi-Lager das ASP.NET mit seinen Web Forms die primäre Technologie werden. Die Entwicklung einer Web Form-Anwendung erfolgt dabei genau so visuell, wie wir das bisher mit den Windows-Formularen in Delphi auch gemacht haben. Und da bei .NET die Programmiersprache (Delphi.NET, VB.NET, C# usw.) keine Rolle spielt, kann man sich bereits jetzt mit dem kostenfreien – aber extrem leistungsfähigen – WebMatrix (www.asp.net/webmatrix/) einarbeiten.

Dienstbote

Frage: Kann mir jemand kurz erklären worin der Vorteil von Stored Procedures liegt und wie diese funktionieren?

Antwort: Der Zugriff auf die direkt in einer Datenbank gespeicherten Prozeduren hat zusammengefasst drei wesentliche Vorteile:

- Bessere Leistung
- Einfache Wartung/Erweiterung
- Bessere Sicherheit

Bevor ein SQL-Server eine Anweisung ausführt, muss der Text geparst, die Zugriffsrechte auf die beteiligten Datenbankobjekte geprüft und im Fall einer *SELECT*-Anweisung der optimale Ausführungspfad analysiert werden. Bei einer normalen *SELECT*-Abfrage arbeitet der Server diese Schritte im ungünstigsten Fall bei jedem Aufruf erneut ab. Bei einer Stored Procedure in der Regel aber nur einmal (d.h. beim Speichern der Prozedur bzw. beim ersten tatsächlichen Aufruf). Ein weiterer Vorteil einer Stored Procedure liegt darin, dass dort ein Teil der Programmlogik implementiert werden kann und somit eine Aufgabe komplett auf dem Server abgearbeitet wird. Der Client ruft nur die Stored Procedure auf und übergibt dabei alle notwendigen Daten als Parameter. Alles weitere erledigt der SQL Server intern, ohne dass zusätzliche Netzwerkzugriffe zum Client notwendig werden. Dies wird insbesondere immer dann sehr wichtig, wenn zwischen dem Client-Rechner und dem SQL Server nur eine Netzwerkverbindung mit niedriger Übertragungsbandbreite zur Verfügung steht. Das folgende Beispiel demonstriert dies:

```
CREATE PROCEDURE TransferMoney
 @iFromKontoNr INT, @iToKontoNr INT, @mValue MONEY,
                                    @iStatus INT OUTPUT
AS
 DECLARE @iError INT
 DECLARE @iRowCount INT
 SET @iStatus = 0
  --Schritt 1: Transaktion starten
 BEGIN TRANSACTION
  --Schritt 2: Konto des Absenders belasten
UPDATE Konto
 SET Betrag = Betrag - @mValue
 WHERE KontoNr = @iFromKontoNr
SELECT @iError = @@ERROR, @iRowCount =
                                    @@ROWCOUNT
  --Schritt 3: Prüfen, ob Schritt 2 erfolgreich war
IF @iRowCount = 0
  --Status 1 = Konto des Absenders wurde nicht aktualisiert
 SELECT @iStatus = 1
  --Zweiten Teilschritt nur im Erfolgsfall starten
IF (@iERROR = 0) AND (@iRowCount = 1)
 BEGIN
   --Schritt 4: Konto des Empfängers gutschreiben
  UPDATE Konto
   SET Betrag = Betrag + @mValue
   WHERE KontoNr = @iToKontoNr
  SELECT @iError = @@ERROR, @iRowCount =
                                    @@ROWCOUNT
  IF @iRowCount = 0
    --Status 2 = Konto des Empfängers wurde nicht
                                    aktualisiert
   SELECT @iStatus = 2
 END
  --Schritt 5: Transaktion im Fehlerfall widerrufen
IF (@iError = 0) AND (@iStatus = 0)
 COMMIT TRANSACTION
ELSE
 ROLLBACK TRANSACTION
  --Fehlernummer zurückliefern
RETURN(@iError)
```

Was bedeutet aber einfachere Wartung/Erweiterung? Wenn der Client nur über Stored Procedures auf die Datenbank zugreift und dabei alle Daten als Parameter (Input- und Output-Parameter) übergeben werden, kann Datenbank-intern die Tabellenstruktur vollständig umgewandelt werden, ohne dass der Client davon betroffen ist. Denn solange sich die Schnittstelle (Name der Stored Procedure und Parameter-Anordnung) nicht ändert, bekommt der Client von Umstrukturierungen in der Datenbank nichts mit.

Der dritte Vorteil betrifft die höhere Sicherheit. Wenn der Client nur über Stored Procedures auf die Datenbank zugreift, aber ansonsten keine Rechte an den Tabellen hat, legt nur die Stored Procedure fest, ob ein *SELECT*-, *INSERT*-, *UPDATE*- oder *DELETE*-Aufruf ausgeführt werden darf oder nicht. Der Client erhält nur das *EXECUTE*-Recht an den Stored Procedures, sodass die gespeicherten Prozeduren exakt festlegen, was der Benutzer zu diesem Zeitpunkt mit den vorgefundenen Daten machen darf.

Kleider machen Leute

Streifzug durch die Windows Forms

Der Borland C#Builder hat es, Microsoft Visual Studio .NET hat aus und auch das brandneue Delphi für das .NET Framework (Codename „Octane“) hat es: Die Rede ist von den Windows Forms aus dem .NET Framework. Doch während es beim C#Builder und auch bei VS.NET gar keine Wahl gibt, steht der Delphianer vor der Qual der Wahl: Nehme ich für die Formulare meiner Benutzeroberfläche die „alte“ VCL.NET oder die „neue“ FCL?

von Andreas Kosch

Aus verständlichen Gründen wird bei den meisten von Ihnen die erste Antwort VCL.NET lauten, denn mit .NET kommt sowieso schon viel Neues auf den Entwickler zu, also macht es auf den ersten Blick doch Sinn, zumindest bei der Gestaltung der Benutzeroberfläche beim Altvertrauten zu bleiben. Allerdings hat jede Medaille ihre zwei Seiten und selbstverständlich hat auch diese Bequemlichkeit ihren Preis. Doch schön der Reihen nach, worin unterscheidet sich die FCL von der VCL.NET?

Hinter dem Begriff FCL steckt die .NET Framework Class Library, wobei dieser Begriff im Microsoft-Umfeld nur sehr selten anzutreffen ist. Dies ist auch kein Wunder, denn in Ermangelung einer Alternative wird dort kein separater Begriff benötigt. Dies ändert sich erst mit „Octane“, denn die neueste Entwicklungsumgebung von Borland kann für die Gestaltung der Benutzeroberfläche von Windows-Anwendungen sowohl mit der .NET-Fassung der vertrauten VCL (Visual Component Library) umgehen als auch mit den visuellen Komponenten aus der FCL. Was die Benutzeroberfläche einer Anwendung angeht, spaltet sich die FCL in die beiden Bereiche Windows Forms (Windows-Anwendungen) und Web Forms (Browser-Anwendungen) auf. Im Fall einer Browser-Anwendung verhält sich „Octane“ genau so wie der C#Builder oder VS.NET – es gibt keine Alternative zur den ASP.NET Web Forms. In dem Borland-Papier „Octane and Delphi Q&A“ liest sich das so: „Existing web development technologies will be replaced by Delphi ASP.NET WebForms and .NET WebServices”. Ein derartiger Satz kann selbstverständlich nicht für die mit früheren Delphi-Versionen entwickelten Windows-Anwendungen gelten. Hier verspricht Borland einen bequemeren Migrationsweg, wobei aber in dem gleichen Borland-Papier deutlich gemacht wird, dass die VCL.NET nur eine Teilmenge der VCL ist und dort nur die am häufigsten genutzten VCL-Komponenten anzutreffen sind.

Was hat es nun aber mit den Nebenwirkungen zu tun, die ich am Anfang als Preis für die Bequemlichkeit angeführt habe? Ein praktisches Beispiel verdeutlicht das Problem am Besten, und um das Ganze so einfach wie nur möglich zu halten, beziehe ich mich auf ein Beispielprojekt, das Borland zusammen mit der dritten Preview-Version von Delphi für .NET ausgeliefert hat. Wenn Sie die Review installiert haben, finden Sie das Beispielprojekt im Verzeichnis C:\Programme\Borland\Delphi for .NET Preview\demos\vcl\shape. Das Beispiel besteht nur aus einem Formular, auf dem die folgenden Komponenten untergebracht sind:

- *TColorBox*
- *TLabel*
- *TShape*
- *TRadioGroup*

Wird die VCL.NET-Anwendung kompiliert, erhalten Sie eine 690 kByte große *EXE*-Datei. Die Dateigröße einer vergleichbaren FCL-Anwendung bewegt sich jedoch im einstelligen kByte-Bereich. Der Grund für diesen Unterschied besteht darin, dass die FCL-Anwendung auf die Klassen in den vorinstallierten Assemblies aus dem .NET Framework zurückgreifen kann, während die VCL.NET-Anwendung alles in das eigene Modul packen muss, da die VCL.NET über P/Invoke direkt auf die Win32-API-Funktionen zugreift. Aber das ist nicht das größte Problem. Versucht man nun, die VCL.NET-Anwendung von einem Netzlaufwerk oder über eine Intranet-URL zu starten, erhält man sofort eine Sicherheits-Exception, die erst dann verschwindet, wenn die Berechtigungen der Anwendung wegen des Rückgriffs auf P/Invoke explizit erhöht wurden. Zwar greifen auch die FCL-Controls auf das Win32-API zu, aber die Microsoft-Assemblies aus dem .NET Framework gelten bereits in der Voreinstellung als sicher. Wenn Sie sich nun fragen „Alles schön und gut – aber warum soll ich meine Anwendung nicht von der lokalen Festplatte aus star-

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter

www.derentwickler.de

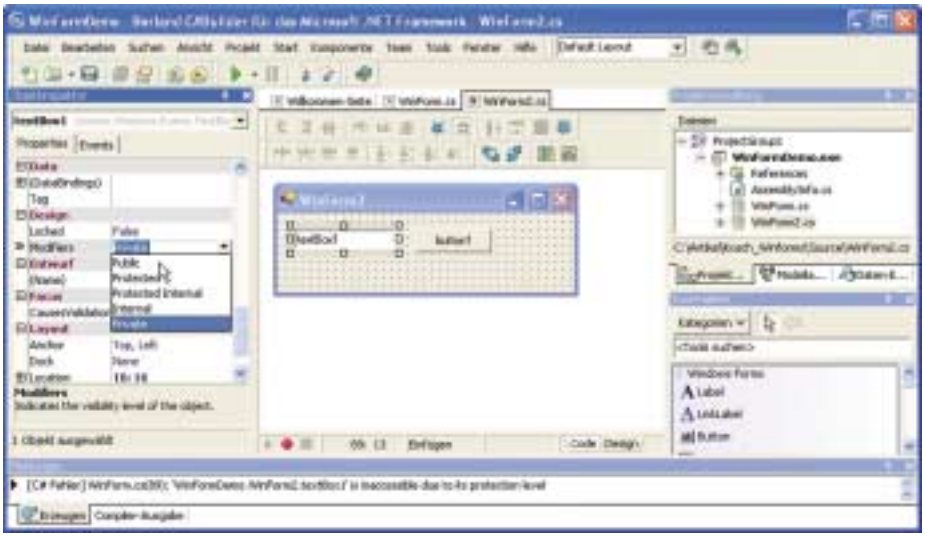


Abb. 1: Die Sichtbarkeits-Regeln sind strenger als bei Delphi

ten?“, schauen Sie sich bitte das folgende Beispiel an. In dieser Anwendung wurden die einzelnen Formulare (Dialoge) in separate Assemblies ausgelagert, die vom Programm über eine URL angefordert werden. Dabei kümmert sich .NET automatisch darum, dass die Assembly-DLL immer dann automatisch auf die eigene Festplatte (Download Cache) kopiert wird, wenn auf dem Server eine neuere Version herumliegt. In diesem Szenario hat ein FCL-Formular gegenüber einem VCL .NET-Formular gleich zwei gravierende Vorteile: Zum einen ist die Dateigröße der FCL-DLL winzig und zum anderen muss auch nicht zwingend an den Rechten gedreht werden:

```
private const string sURL = "http://192.168.10.1/VS/
                                    LoadFormAssembly.dll";
private const string sTypeName = "LoadFormAssembly.
                                    AssemblyForm";

private void buttonLoadFrom_Click(object sender, System.
                                    EventArgs e)
{
 Assembly aFrmAsm;
 aFrmAsm = Assembly.LoadFrom(sURL);
 Type aFrmType = aFrmAsm.GetType(sTypeName);
 Object aFrmObj = Activator.CreateInstance(aFrmType);
 Form aFrm = (Form)aFrmObj;
 aFrm.ShowDialog();
}
```

Der dritte Nachteil der VCL.NET hat mit rein praktischen Erwägungen zu tun – die Controls aus der FCL unterstützen die Fähigkeiten aktueller Windows Versionen besser, sodass die Handhabung im Alltag einfacher ist. Sie sehen, dass die VCL.NET nicht für alle Einsatzfälle die beste Wahl ist. Doch genug der langen Vorrede – schauen wir uns nun die Fähigkeiten der Windows Forms an. Und solange „Octane“ nicht offiziell das Licht der Welt erblickt hat, greife ich in meinen Beispielen auf den Borland C#Builder zurück. Auch dafür gibt es einen plausiblen Grund: Das .NET Framework stellt nicht nur die Klassen und Komponenten aus der FCL zur Verfügung, sondern auch den graphischen Designer für das visuelle Gestalten des Formulars. Immer dann, wenn Sie im C#Builder oder in „Octane“ ein Windows Forms-Formular gestalten, arbeiten Sie mit dem gleichen Microsoft-Designer, der auch in VS.NET genutzt wird. Nur dann, wenn das Formular aus der VCL.NET stammt, bringt Borland in „Octane“ seinen eigenen Designer ins Spiel.

Windows Forms

Im Kern hat sich gegenüber früher nichts geändert: Auch die Windows Forms stellen Eigenschaften, Methoden und Ereignisse zur Verfügung, um das Aussehen und Verhalten des Formulars den eigenen Anforderungen anpassen zu können. Auch in der .NET-Welt ist ein Formular prinzipiell nur ein Control, wobei sich hinter Control sowohl das Steuerelement aus Win32 als auch eine .NET-Klasse verbirgt. In .NET spiegelt sich dies im Vererbungsbaum wieder, die Windows Forms stammen von der Klasse *System.Windows.Forms.Form* ab, welche wiederum von der Klasse *System.Windows.Forms.Control* abgeleitet wurde. Die Klasse *Control* implementiert die Basisfunktionalität, wobei über die Eigenschaft *Handle* der Rückgriff auf das native Window-Handle des Fensters bereitgestellt wird. Da die Klasse *Control* keine Darstellung implementiert, dient sie nur das Basisklasse, von der alle sichtbaren Controls abgeleitet wer- den.

Die Klasse *Forms* implementiert die Darstellung von Fenstern in der Anwendung. Ein sichtbares Fenster ist unter Windows eng mit dem GDI (Graphical Device Interface) verbunden, sodass die Klasse *Forms* auch eine enge Verbindung zum für GDI+ zuständigen Namespace *System.Drawing* hat. Die Klasse *Graphics* stellt Methoden für die Darstellung von Fenstern zur Verfügung. Die eigene Anwendung kann die „richtige“ Instanz der *Graphics*-Klasse auf verschiedenen Wegen erhalten:

- Das *Paint*-Event (reagiert auf *WM_PAINT*) übergibt den Parameter.
- Die Methode *CreateGraphics* der *Control*-Klasse erzeugt eine Instanz.
- Über *FromHdc*, wenn ein Device-Kontext zur Verfügung steht.
- Über *FromHwnd*, wenn das Fenster-Handle zur Verfügung steht.

Schaut man sich dann allerdings das Hantieren mit Formularen an, tauchen sofort die ersten Unterschiede auf. Es beginnt bereits damit, dass wir uns nun selbst um das Erzeugen von zusätzlichen Formularinstanzen kümmern müssen. Um zu verdeutlichen, was ich damit meine, greife ich zuerst auf Delphi zurück, um dort ein weiteres Formular hinzufügen und als Reaktion auf einen Button-Klick anzuzeigen. Nachdem ich im Hauptformular die zweite Unit eingebunden habe, kann ich sofort das zweite Formular über die Methode *ShowModal* anzeigen, weil sich Delphi automatisch in der Projektdatei um das Erzeugen aller Formular-Instanzen gekümmert hat.

Bei einer FCL-Anwendung ist das jedoch nicht der Fall, sowohl der C#Builder als auch VS.NET erzeugen in einer Windows Forms-Anwendung über die Methode *Main* nur das Hauptformular automatisch:

```
[STAThread]
static void Main()
{
 Application.Run(new WinForm());
}
```

Wenn ich das vorhin im Text beschriebene Delphi-Miniprogramm mit dem C#-Builder nachbaue und über die Objektgalerie ein weiteres Formular hinzufüge, muss ich vor dem Aufruf der Methode *ShowDialog* zuerst über *new* eine Formularinstanz erzeugen:

```
private void button1_Click(object sender, System.
                                        EventArgs e)
{
 WinForm2 aFrm2 = new WinForm2();
 aFrm2.ShowDialog();
}
```

Wenn ich nun ein Eingabefeld im zweiten Formular ablege und den dort sichtbaren Text vor dem Anzeigen ändern will, ist das in Delphi überhaupt kein Problem, wie das folgende triviale Beispiel zeigt:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 Form2.Edit1.Text := 'Vorbelegter Text';
 Form2.ShowModal;
end;
```

Baue ich nun das Gleiche mit dem C#Builder nach, so erhalte ich beim Kompilieren die rote Karte (Abb. 1), denn die *TextBox* (das FCL-Gegenstück zu VCL-Komponente *TEdit*) aus dem zweiten Formular ist zur Zeit noch „unsichtbar“:

```
private void button1_Click(object sender, System.
                                        EventArgs e)
{
 WinForm2 aFrm2 = new WinForm2();
 aFrm2.textBox1.Text = "Der neue Text";
 aFrm2.ShowDialog();
}
```

Abb. 2: Die geniale Idee für das Editieren des Hauptmenüs

Mit „unsichtbar“ meine ich dabei die Sichtbarkeits-Regel für den Compiler, in der Voreinstellung wird das Eingabefeld auf dem zweiten Formular als privat gekennzeichnet, sodass der Zugriff darauf nur innerhalb der eigenen Formular-Klasse erlaubt ist. Um diesen Spuk zu verscheuchen, setze ich im Objektinspektor die Eigenschaft *Modifiers* auf den Wert *Public*.

In Delphi konnten wir über die Ereignisse *OnCreate* und *OnDestroy* ganz gezielt auf das Erzeugen und Zerstören einer Formular-Instanz reagieren. Bei einer .NET-Anwendung muss sich auch hier aufgrund des automatisch im Hintergrund werkelnden Garbage Collectors einiges ändern. Das Ereignis *Load* informiert das Formular darüber, dass seine Instanz soeben erzeugt wurde. Über das Ereignis *Closing* fragt .NET beim Formular nach, ob es gegen das beabsichtigte Schließen ein Veto einlegen will. Ist das nicht der Fall, informiert das Ereignis *Closed* darüber, dass dieses Formular tatsächlich geschlossen wurde. Somit gibt es also auch in der FCL ein Gegenstück zu den VCL-Ereignissen *OnCloseQuery* und *OnClose*. Was er aber nicht gibt, ist ein Gegenstück zu *OnDestroy*. Dies ist auch kein Wunder, denn den Zeitpunkt für das tatsächliche Zerstören einer Objektinstanz legt nicht mehr unser Programm fest, sondern ausschließlich die CLR (Common Language Runtime) von .NET, wobei der Müllsammler (Garbage Collector) eigenen Regeln folgt. Für uns bedeutet dies, dass alle Aufräumarbeiten in die Methode *Dispose* verlagert werden müssen. Sowohl der C#Builder als auch VS.NET statten daher jedes Formular automatisch mit der folgenden Implementierung aus:

```
/// <summary>
/// Ressourcen nach der Verwendung bereinigen
/// </summary>
protected override void Dispose (bool disposing)
{
 if (disposing)
 {
  if (components != null)
  {
   components.Dispose();
  }
 }
 base.Dispose(disposing);
}
```

Wenn Sie nun irgendwelche Komponenten auf dem Formular ablegen, sorgt die Entwicklungsumgebung dafür, dass diese automatisch immer dann zur *components*-Kollektion hinzugefügt werden, wenn am Ende ein Aufräumen notwendig wird.

Es wäre für Sie ermüdend, wenn ich nun auf die im Objektinspektor verfügbaren Eigenschaften der Windows Forms näher eingehen würde. An dieser Stelle gehen Sie am Besten selbst auf Entdeckungsreise. Sie werden sehen, dass dort Sachen auftauchen, die bisher genutzte VCL-Komponenten überflüssig machen. Das beste Beispiel dafür sind die Eigenschaften *AutoScroll*, *AutoScrollMinSize* und *AutoScrollPosition* als funktionaler Ersatz für die *TScrollBox*-Komponente der VCL. Nicht völlig unerwähnt bleiben darf jedoch die Tatsache, dass die Windows Forms alles in eine einzige Datei kapseln. Dank der Regionen wird dieser Teil in der Voreinstellung vor unseren Augen verborgen. Möchten Sie das Gegenstück zu dem Inhalt der *DFM*-Dateien von Delphi auch im Windows Forms-Programm sehen,

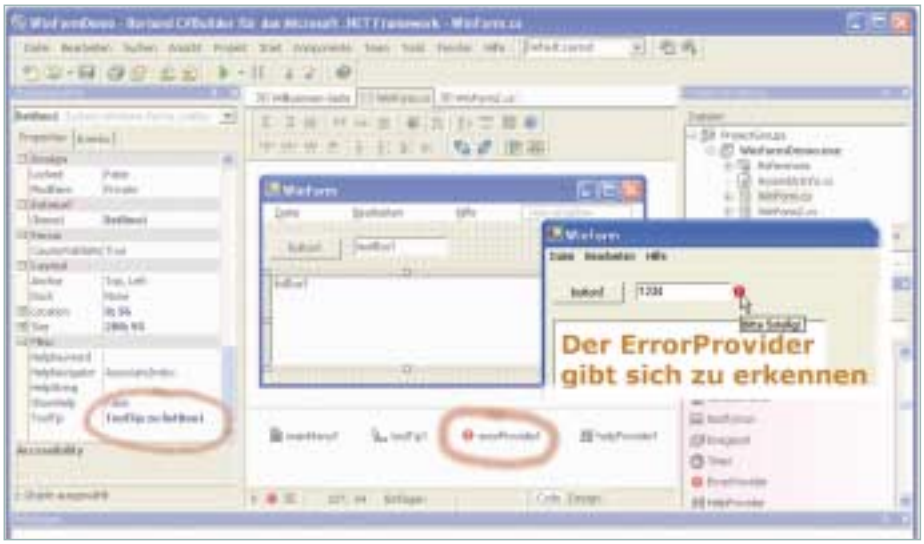


Abb. 3: Der *ErrorProvider* in Aktion

muss die Region „Vom Windows Form-Designer erzeugter Code“ zuerst aufgeklappt werden.

Komponenten

Immer dann, wenn das Formular eine Menüzeile erhalten soll, legen Sie die *MainMenu*-Komponente von der Tool-Palette auf dem Formular ab. Gegenüber Delphi hat sich hier also noch nicht einmal der Name der Komponente geändert. Allerdings geht das Gestalten des Menüs dann bei der FCL deutlich eleganter vonstatten als das bei der VCL der Fall ist. Der Designer zeigt das Menü in Echtzeit im Formular an und kennzeichnet alle die Stellen (Abb. 2), an denen Sie den Aufbau des Menüs gestalten können.

Die VCL von Delphi unterscheidet streng zwischen normalen Komponenten und Datensteuerungs-Komponenten, für die VCL gibt es zum Beispiel einen großen Unterschied zwischen *TListBox* und *TDB- ListBox*. Mit der FCL verschwindet diese Aufteilung, Sie können fast jede Eigenschaft eines fast jeden Controls an eine Datenquelle binden. Dabei muss die Datenquelle nicht unbedingt aus einer Datenbank heraus sprudeln, das folgende Beispiel demonstriert, dass sogar ein Array eine gültige Datenquelle ist. Sobald die Array-Instanz der *ListBox*-Eigenschaft *DataSource* zuordnet wurde, tauchen in der *ListBox* die drei Einträge auf:

```
string[] sIDE = {"Delphi","C#Builder","VS.NET"};
listBox1.DataSource = sIDE;
```

Sie können allerdings auch Controls an öffentliche Eigenschaften einer Klasse binden, sodass die Benutzeroberfläche sofort auf jede Veränderung des Zustands einer Klasseninstanz visuell reagiert. Angenommen, die eigene Klasse *OSUserInfo* stellt unter anderem die Eigenschaften *UserName* und *eMail* zur Verfügung. Wenn die *ListBox* nun nur den Namen aller vorgefundenen Klasseninstanzen anzeigen soll, wird neben der Eigenschaft *DataSource* auch auf die Eigenschaft *DisplayMember* (diese beiden Eigenschaften sind auch dann zuständig, wenn das Control an eine Datenbankquelle gebunden werden soll) zurückgegriffen:

```
OSUserInfo[] aUser = {
 new OSUserInfo("Andreas Kosch", "OssiSoft@aol.com"),
 new OSUserInfo("Manfred Mustermann",
                                        "MM@t-online.de")};
listBox1.DataSource = aUser;
listBox1.DisplayMember = "UserName";
```

Ein gegenüber der VCL neues Konzept steckt auch hinter den Providern der FCL. Ein Provider erweitert die Eigenschaften von Controls des aktuellen Formulars. Zum Beispiel stellt die FCL die Komponente *ToolTip* als Provider zur Verfügung. Wenn Sie diese Komponente von der Tool Palette auf das Formular ziehen, legt der Designer die Komponente im Komponentenbereich unterhalb des Formulars ab. Gleichzeit taucht für alle auf dem Formular untergebrachten Controls im Objektinspektor wie von Geisterhand die neue Eigenschaft *ToolTip* auf (Abb. 3). Beliebig viele Controls können die gleiche *ToolTip*-Komponente nutzen, solange sich nur der anzuzeigende Text unterscheidet. Das gleiche gilt auch für den *ErrorProvider*, dessen Ergebnis ebenfalls in der Abbildung 3 zu sehen ist. Sobald für ein Control ein Fehlertext zugewiesen wird, zeigt die FCL ein blinkendes Ausrufezeichen mit rotem Hintergrund rechts neben dem Control an. Bewegt der Anwender seinen Mauszeiger auf diese Markierung, so bekommt er den Fehlertext als *ToolTip*-Fenster zu Gesicht.

Der Unterschied beim *ErrorProvider* liegt nur darin, dass eine Konfiguration im Objektinspektor wenig Sinn macht. Statt dessen wird die Fehlerbedingung direkt im Source Code festgelegt. Das folgende Beispiel demonstriert dies, immer dann, wenn die Eingabe nicht aus fünf Zeichen besteht, setzt die *TextBox* beim Verlassen des Eingabefeldes den Fehlertext, sodass der Anwender sofort eine visuelle Rückmeldung erhält:

```
private void textBox1_Validating(object sender,
 System.ComponentModel.CancelEventArgs e)
{
 if (textBox1.Text.Length != 5 )
 {
  errorProvider1.SetError(textBox1, "Bitte 5stellig!");
  e.Cancel = true;
 }
 else
 {
  errorProvider1.SetError(textBox1, "");
 }
}
```

Selbstverständlich gibt es auch in der FCL eine Formular-Vererbung, die von Microsoft mit dem Begriff *Visual Inheritance* umschrieben wird. Dieses Thema oder gar die im Vergleich zur VCL neuen Möglichkeiten der visuellen Gestaltung von mehrsprachigen Benutzeroberflächen passen jedoch nicht mehr in diesen Beitrag.

Geschickt gelöst

C#Builder und die Borland Data Providers for ADO.NET

Der Borland C#Builder wurde vor einigen Monaten in den vier Versionen Personal, Professional, Enterprise und Architect veröffentlicht. Bis auf die Personal Edition enthalten diese alle etwas, was meiner Meinung nach das beste an C#Builder ist, nämlich die sogenannten Borland Data Provider for ADO.NET. In diesem Artikel möchte ich zeigen, was genau man mit den BDP for .NET und C#Builder machen kann.

von Bob Swart

ADO.NET und BDP

Das .NET Framework stellt Datenbank-Konnektivität über ADO.NET zur Verfügung und tatsächlich bieten die meisten Datenbankanbieter heute .NET-Zugriff auf ihre Datenbankprodukte an. Hierfür verwenden sie einen Satz von Klassen, die das ADO.NET Data Provider-Modell implementieren. Entwickler können damit gut arbeiten, da sie nicht den Umgang mit einem anderen Satz von Komponenten lernen müssen, wenn sie mit einer anderen Datenbank arbeiten wollen. Mit dem Release von C#Builder hat Borland eine Datenzugriffs-Technologie namens Borland Data Providers for .NET (bzw. for ADO.NET) eingeführt. Dabei handelt es sich zwar um einen neuen Satz von Komponenten, aber dieser setzt ebenfalls das ADO.NET-Entwurfsmuster um.

Die Borland Data Provider implementieren und erweitern die ADO.NET Provider-Klassen – genauer die Klassen *SqlConnection*, *SqlDataAdapter* und *SqlCommand* – und sind für verschiedene DBMS wie InterBase, SQL Server 2000 Developer Edition (aka MSDE), SQL Server 2000, Oracle und DB2 verfügbar.

In diesem Artikel möchte ich demonstrieren, wie die BDP for .NET sich zur Entwurfszeit verhalten und die Schritte, die für das Deployment einer mit ihnen erstellten Applikation nötig sind, beschreiben.

Verbindung mit InterBase

Die Borland Data Provider for .NET sind in jeder C#Builder Version außer der freien Personal Edition verfügbar. C#Builder Professional enthält BDP-Treiber für lokale InterBase- und MSDE-Datenbanken, Enterprise und Architect zusätzliche Treiber für InterBase, SQL Server, Oracle und DB2.

In diesem Artikel wollen wir uns mit InterBase verbinden, sodass C#Builder Professional ausreicht, um „mitzuspielen".

Beginnen Sie ein neues C#Builder-Projekt und geben Sie diesem einen sinnvollen Namen, etwa *BDPIB*. In der rechten oberen Ecke der C#Builder-IDE sehen Sie den Project Manager und direkt darunter drei Register, darunter eines für den Data Explorer.

Der Data Explorer zeigt die verfügbaren BDP-Provider an, die mit der jeweiligen C#Builder-Version installiert sind. Selbstverständlich muss auch die Datenbank, mit der Sie sich verbinden wollen, laufen (also starten Sie entweder InterBase oder lassen Sie diese als NT-Dienst laufen). Sie können den InterBase-Knoten öffnen, um die Verbindungen zu sehen. Die Treiber-Einstellungen sind in *bdpDataSources.xml* gespeichert, die verschiedenen Verbindungen in *bdpConnections.xml*. Beide Dateien befinden sich im Verzeichnis *BDP\1.0\bin directory*. Wenn Sie eine neue Verbindung hinzufügen wollen, können Sie mit der rechten Maustaste auf einen Treiber-Knoten klicken und die Option ADD NEW CONNECTION wählen. Um einen neuen Treiber hinzuzufügen (beispielsweise den BDP-Treiber für MySQL, den Sie auf Borlands CodeCentral-Seiten [1] finden), müssen Sie *bdpDataSources.xml* jedoch von Hand editieren.

Sie können entweder eine neue Verbindung hinzufügen oder aber die Default-Connection bearbeiten, also die Datenbank und andere Properties anpassen, um korrekt auf InterBase zugreifen zu können. Um die bestehende Verbindung zu editieren, wählen Sie über die rechte Maustaste die Option MODIFY CONNECTION, wodurch der Connection Editor (Abb. 1) gestartet wird.

Abb. 1: Connections Editor

Abb. 2: Die Beispieldatenbank im Data Explorer

Neben Username und Passwort ist *Database* die wichtigste Eigenschaft. Beachten Sie, dass Sie, wenn Sie von einem anderen Client-Rechner aus auf die InterBase-Datenbank zugreifen wollen, dem Pfad den Namen oder die IP-Adresse des Servers voranstellen müssen (in diesem Fall *192.168.92.42*). Dies ist besonders wichtig, wenn man ASP.NET-Anwendungen erstellt.

Sie können den Button TEST verwenden, um zu prüfen, ob die Einstellungen korrekt sind, und den Dialog schließen, wenn diese ordnungsgemäß funktionieren. Öffnen Sie dann im Data Explorer den Knoten *IBConn1*, um eine Liste der Tabellen, Views und Prozeduren der Beispieldatenbank *employees* zu sehen (Abb. 2).

Um zu sehen, wie einfach die BDP-Komponenten verwendet werden können, wählen Sie nun die Tabelle *EMPLOYEE* aus und ziehen diese auf Ihr Windows Form. Es mag so aussehen, als sei nichts passiert, aber wenn Sie am unteren Rand des Designers schauen, werden Sie sehen, dass zwei nicht sichtbare BDP-Komponenten erzeugt wurden, nämlich *bdpConnection1* und *bdpDataAdapter1*. Erstere ist für die Verbindung mit der InterBase-Datenbank verantwortlich, die zweite nutzt diese Verbindung, um mit der *EMPLOYEE*-Tabelle zu kommunizieren.

Wir können die Verbindungs-Komponente ignorieren (diese haben wir bereits getestet, als wir den Connections Editor verwendeten) und uns der Konfiguration der *bdpDataAdapter*-Komponente zuwenden.

BdpDataAdapter

Wählen Sie die *BdpDataAdapter*-Komponente aus. Im unteren Teil des Objekt-Inspektors sehen Sie zwei spezielle Entwurfszeitunterstützungs-Funktionen (Verbs), nämlich *Configure Data Adapter...* und *Generate Typed Dataset...*. Wenn Sie erstere anklicken, wird der in Abbildung 3 gezeigte Data Adapter Configuration-Wizard angezeigt.

Beim Start zeigt der Wizard den Namen der Tabelle, die Sie aus dem Data Explorer gezogen haben. Jetzt können Sie die Tabelle jedoch wechseln und die Felder angeben, die Sie auswählen, einfügen, löschen oder aktualisieren wollen. Wenn Sie eine neue Tabelle auswählen, bekommen Sie auch eine neue Liste mit Feldern für diese Tabelle angezeigt. Sie können angeben, ob Sie nur Daten auswählen (für Read-only-Applikationen) oder auch Inserts, Deletes und/oder Updates zulassen wollen. Den GENERATE SQL-Button müssen Sie betätigen, um die neuen SQL-Anfragen, die die *BdpDataAdapter*-Komponente verwendet, zu generieren.

Die Checkbox *Optimize* (per Default nicht aktiviert) kann verwendet werden, um Queries zu erzeugen, die nur die primären Index-Felder in der *where*-Klausel verwenden. Dies führt zu effizienteren Anfragen, kann aber zu Problemen führen, wenn zwei oder mehr User Nicht-Schlüsselfelder des selben Datensatzes aktualisieren.

Wie dem auch immer, selbst wenn Sie die Tabelle nicht wechseln und die Felder auf * belassen, so führt ein Klick auf GENERATE SQL zu einer Anfrage, die alle Felder explizit auflistet:

```
SELECT EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT,
    HIRE_DATE, DEPT_NO, JOB_CODE, JOB_GRADE,
    JOB_COUNTRY, SALARY FROM EMPLOYEE
```

Wenn Sie die SQL-Statements erzeugt haben, können Sie über das Register *DataSet* angeben, welches DataSet verwendet werden soll, um das Ergebnis des SQL-Befehls aufzunehmen. Da noch keine DataSet-Komponente existiert, sollten wir die Option, ein neues DataSet zu erstellen, wählen und dieses zum Beispiel *dataSet1* nennen.

Nachdem wir ein DataSet spezifiziert haben, können wir zum Register *Preview* gehen und uns das Ergebnis anschauen (Abb. 4). Die Option *Limit rows* hat dabei übrigens ausschließlich Auswirkungen auf die Wiedergabe im Preview-Dialog, nicht

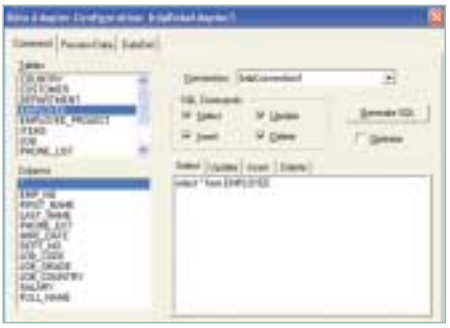


Abb. 3: Der Data Adapter Configuration-Wizard

auf die Darstellung der Daten zur Laufzeit.

Wenn Sie den Data Adapter Configuration-Dialog schließen, werden Sie sich vielleicht fragen, warum Sie die DataSet-Komponente nicht im Bereich für nicht sichtbare Komponenten im Designer sehen. Dies ist aber nur ein kleines Entwurfszeit-Problem: Wählen Sie über die rechte Maustaste LINE UP ICONS und die Komponente wird sich zeigen.

Wir haben bislang zwei der vier Borland Data Provider-Komponenten verwendet: *BdpConnection* und *BdpDataAdapter*. Bleiben also die Komponenten: *BdpCommand* und *BdpCommandBuilder*.

Betrachten von Daten

BdpDataAdapter nutzt *BdpCommand*, um das DataSet mit dem Ergebnis der SQL-Anfrage zu befüllen. Wir können die resultierenden Daten visualisieren, indem wir visuelle Komponenten wie das DataGrid oder andere Klassen, die Listen unterstützen, einsetzen.

Ziehen Sie aus der Kategorie *Data Controls* eine DataGrid-Komponente auf das Formular. Setzten Sie die Eigenschaft *DataSource* auf *dataSet1* und öffnen Sie dann die Eigenschaft *DataMember*, um die richtigen Daten auszuwählen. Anfangs gibt es vielleicht noch nichts für das *DataMember* auszuwählen. Wenn Sie jedoch die Eigenschaft *Active* des *BdpDataAdapters* auf *True* setzen, wird das DataSet mit den in *Table1* enthaltenen Daten gefüllt. Und das beste daran: Sie werden die Daten live zur Entwurfszeit sehen (Abb. 5).

Daten aktualisieren

Natürlich wollen wir unsere Daten nicht nur betrachten, sondern dem Anwender

auch erlauben, diese zu verändern – und diese Änderungen gegebenenfalls wieder rückgängig zu machen. Letzteres ist zum Glück sehr einfach, da ADO.NET (und BDP) auf dem Konzept eines „entkoppelten DataSets" basiert. Das heißt, dass Änderungen, die wir an den Daten im DataGrid durchführen, nicht automatisch an die InterBase-Datenbank selbst zurückgeschickt werden. Die Änderungen werden im DataSet gecacht und wir müssen sie explizit an die Datenbank senden – oder sie aber vorher wieder rückgängig machen.

Fangen wir mit dem Undo an: Die DataSet-Komponente verfügt über eine nette Methode namens *RejectChanges*. Diese nimmt alle Änderungen zurück, die gemacht wurden, seitdem das letzten Update an die InterBase-Datenbank geschickt wurde. Platzieren Sie drei Buttons auf dem Formular, nennen Sie diese *btnUndo*, *btnUpdate* und *btnClose* und setzen Sie deren *Text*-Eigenschaft auf *Undo*, *Update* und *Close*. Schreiben Sie in den *Click*-Event Handler für das *Undo* folgenden Code:

```
private void btnUndo_Click(object sender,
                                    System.EventArgs e)
{
 dataSet1.RejectChanges();
}
```

Jetzt stellt sich natürlich die Frage, wie wir Updates an die Datenbank schicken können. Dies lässt sich ebenfalls mit einer Zeile Code bewerkstelligen, indem wir die Methode *AutoUpdate* aufrufen. Dies kann im *Click*-Event des *Update*-Buttons implementiert werden:

```
private void btnUpdate_Click(object sender,
                                    System.EventArgs e)
{
 if (this.dataSet1.HasChanges())
  this.bdpDataAdapter1.AutoUpdate();
}
```

Beachten Sie, dass die *HasChanges*-Methode dabei hilft zu entscheiden, ob es überhaupt sinnvoll ist, *AutoUpdate* (oder *RejectChanges*) aufzurufen.

Schließlich ist es noch eine gute Idee zu überprüfen, ob irgendwelche Änderungen vorliegen, bevor man die Anwendung beendet. Dies lässt sich im *Click*-Ereignis des *Close*-Buttons erledigen:

```
private void btnClose_Click(object sender,
                                    System.EventArgs e)
{
 btnUpdate_Click(sender, e);
 Close();
}
```

Listing 1

```
private void btnUpdate_Click(object sender,
                                    System.EventArgs e)
{
 if (this.dataSet1.HasChanges())
 {
  BdpCommandBuilder cb = new
                BdpCommandBuilder(bdpDataAdapter1);
  try
  {
   cb.UpdateMode = Borland.Data.Common.BdpUpdate-
                                    Mode.All;  // or Key
   bdpDataAdapter1.Update(dataSet1, "Table1");
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message);
  }
 }
}
```

Abb. 4: Vorschau auf die Daten

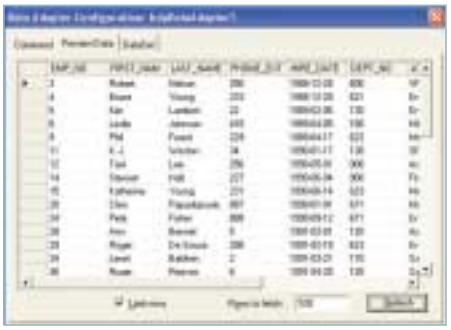


Abb. 5: „Live Data"

BdpCommandBuilder

Es ist toll, wenn man alles in einer einzigen Zeile Code machen kann. Um die Verwendung der *BdpCommandBuilder*-Komponente zu illustrieren, wollen wir jedoch die *AutoUpdate*-Methode mit etwas mehr Details neu schreiben.

Ich erzeuge dazu eine *BdpCommandBuilder*-Instanz und setzte den *UpdateMode* auf *All* (dies platziert alle Felder in die *where*-Klausel des *Update*-SQL-Statements; die Alternative wäre *Key*, womit nur die Primärschlüssel-Felder in die *where*-Klausel gesetzt würden). Dann rufe ich explizit die *Update*-Methode auf und übergebe das DataSet sowie den Namen der zu aktualisierenden Tabelle. Der *BdpCommandBuilder* generiert nun eine SQL-Anfrage, um die Updates an die InterBase-Datenbank zu schicken. Daraus resultiert die in Listing 1 gezeigte Implementierung des *Click*-Events des *Update*-Buttons (inklusive *try-catch*-Block).

Deployment der Applikation

Um die ausführbare Datei unserer C#Builder-Applikation deployen zu können, muss auf dem Client-Rechner zunächst der InterBase-Client (vor allem *gds32.dll*) installiert sein. Danach folgen die C#Builder Executable selbst und die Borland Database Provider-Assemblies. Letzere sind von Borland mit einem Strong Key signiert, sodass sie mit gacutil in den Global Assembly Cache (GAC) deployt werden können.

Folgende BDP-Assemblies müssen deployt werden:

- *Borland.Data.Provider.dll*: die Borland Data Provider for .NET
- *Borland.Data.Common.dll*: gemeinsame BDP-Klassen
- *Borland.Data.Intervase.dll*: InterBase-spezifische Klassen
- *bdpint.dll*: die Win32-DLL, die auf den *gdb32.dll*-InterBase-Client linkt

Bob Swart (aka Dr. Bob) arbeitet als Autor, Trainer, Webmaster, Entwickler und Consultant. Er ist erreichbar unter www.drbob42.com.

Links & Literatur

[1] Treiber für MySQL: codecentral.borland.com/codecentral/ccweb.exe/listing?id=20102

3D-Welten

Spiele-Programmierung mit VB.NET und Direct3D – Teil 1

Dreidimensionale Körper oder gar ganze 3D-Welten darzustellen gehört zur „Königsdisziplin“ in Sachen Grafikausgabe. Doch längst gehört dieses Gebiet zum festen Bestandteil vieler Anwendungen, vor allem im Hinblick auf Spiele, CAD- oder Raytracing-Programme. Somit wachsen natürlich auch ständig die Anforderungen an die Programmierer solcher Anwendungen. Damit diese jedoch nicht im Regen stehen, veröffentlicht Microsoft eine ständig erweiterte Version von Direct3D, einem Bestandteil der Multimediaschnittstelle DirectX. In dieser Artikelserie soll Ihnen anhand mehr oder weniger einfacher Beispiele gezeigt werden, wie Sie Direct3D für sich nutzen können.

von Jens Konerow

Die Programmierung mit Direct3D gehört sicher nicht zu den leichtesten Aufgaben und es bedarf etwas an Theorie, bevor wir mit dem Programmieren loslegen können. Aus diesem Grund befasst sich dieser Teil der Artikel-Serie größtenteils mit der Theorie und weniger mit praktischen Beispielen. Dennoch soll der Spaß an der Programmierung nicht Außen vor bleiben und es folgen kleine Beispiele.

Das Koordinatensystem

Für 3D-Anwendungen kommen in der Regel zwei Koordinatensysteme zum Einsatz: Das linkshändige oder das rechtshändige kartesische Koordinatensystem. In Direct3D gilt ersteres als Standard. Bewegen Sie sich bei diesem Koordinatensystem auf der X-Achse nach Rechts, entspricht das der positiven Richtung. Auf der Y-Achse befinden sich die positiven Werte oberhalb des Koordinatenursprungs. Der positive Bereich der Z-Achse verläuft vom Betrachter her weg (Abb. 1).

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi CD sowie unter
www.derentwickler.de

3D-Primitive

Wird von einer 3D-Primitive gesprochen, ist eine Sammlung von Punkten in einem Raum gemeint, die einen dreidimensionalen Körper bilden. Die einfachste Form einer 3D-Primitive besteht aus einer sogenannte *point list*. Am häufigsten kommen jedoch die Polygone zur Geltung. Die einfachsten Polygone bestehen aus drei Vertexen (engl. vertices), den Eckpunkten eines Polygons, woraus ein Dreieck resultiert. Mit Hilfe dieser Dreiecke lassen sich alle erdenkliche Körper modellieren. Selbst solch komplexe Gebilde, wie das Model eines Menschen, lassen sich mit einzelnen Polygonen konstruieren. Abbildung 2 zeigt einen aus Dreiecken aufgebauten Würfel. In Direct3D gibt es sechs verschiedene Arten von Primitiven, welche spätestens beim Zeichnen von Vertexen eine Rolle spielen werden. Abbildung 3 zeigt Ihnen den Unterschied zwischen allen Varianten.

Matrizen

Programmiertechnisch gesehen ist eine Matrix in Direct3D ein zweidimensionaler Array. Typisch ist hierbei die Verwendung einer 4x4 Matrix. Jene besitzt vier Zeilen sowie vier Spalten. In der 3D-Programmierung dienen die Matrizen vor allem der Bewegung im Raum. Was es genau mit den Matrizen auf sich hat und wie Sie damit Rechnen können soll zum gegebenen Zeitpunkt besprochen werden.

World-, View- und Projection-Matrix

Um eine dreidimensionale Welt auf dem Monitor darstellen zu können, gibt es in Direct3D drei verschiedene Matrizen, mit dessen Hilfe die gewünschte Perspektive eingenommen werden kann. Dabei durchlaufen sämtliche Körper, die es darzustellen gilt, einige Transformationen. Wenn Sie in einer Anwendung beispielsweise einen Würfel kreieren, so gelten alle Vektoren der Vertexen in einem lokalen Koordinatensystem, relativ zum Ursprung. Benötigen Sie nun drei Würfel in Ihrer 3D-Welt, so könnten Sie entweder drei Würfel mit den gewünschten Koordinaten manuell erstellen oder Sie nutzen das eine Model und platzieren Kopien Ihres Würfels in der Welt, indem Sie die Objekte verschieben (Translation). Außerdem ließe sich der Würfel zusätzlich drehen (Rotation) oder in der Größe verändern (Scaling). Zu beachten ist hierbei jedoch, dass die Reihenfolge der Transformationen eine entscheidende Rolle spielt. So erhalten Sie nicht dasselbe Ergebnis wenn ein Objekt zunächst durch Translation und anschließend durch Rotation verändert wird, anstatt das Objekt erst zu Rotieren und dann zu verschieben. Doch dazu später mehr, wenn es explizit um die Hintergründe der Matrizen geht. Nach den Transformationen besitzt jeder Vertex nun einen neuen Vektor, welcher die Position in der World-Matrix repräsentiert.

Natürlich muss es ebenfalls möglich sein, sich in der dreidimensionalen Welt zu bewegen. Dazu dient die View-Matrix, welche als Kamera angesehen werden kann. Drei Vektoren geben die Position, die Orientierung, sprich die Blickrichtung

Anzeige

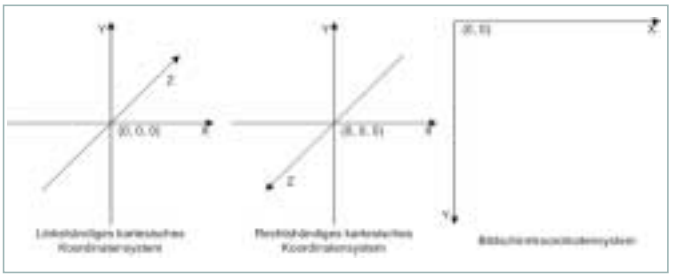


Abb. 1: Unterschiede der Koordinatensysteme

der Kamera, und die Richtung nach oben an.

Als letztes sei die Projection-Matrix zu erwähnen. Das Blickfeld unseres Auges lässt sich als ein Kegel darstellen (Abb. 4). Dieser Sachverhalt wird auch als Sichtkegel (Viewing Frustrum) bezeichnet. Nimmt die Entfernung zu, so vergrößert sich ebenfalls unsere Sicht. Hingegen können wir bei geringer Entfernung eher weniger überschauen. Genauso verhält es sich mit der Projection-Matrix. Im Gegensatz zum Auge verwendet Direct3D ein pyramidenartiges Blickfeld anstatt eines Kegels, was auf die Beschaffenheit des Monitors zurück zu führen ist. Sie bestimmt außerdem ab welcher Entfernung Objekte nicht mehr zu sehen sind bzw. wie nah ein Objekt sein darf, bis es als hinter der Kamera gilt.

Surfaces

Zwischen der Modellierung einer virtuellen Welt und der Darstellung dieser auf dem Monitor besteht ein Konflikt, denn alle im Speicher gelagerten Koordinaten liegen in der dritten Dimension vor. Hingegen ist ein Monitor auf zwei Dimensionen beschränkt. Wie Sie im vorherigen Abschnitt erfahren haben, wird dieses Problem mit Hilfe der unterschiedlichen Matrizen gelöst, denn diese erzeugen das benötigte 2D-Bild einer gewünschten Szene. Jene Daten werden auf einer Surface gespeichert. Eine Surface ist nichts anders als ein Speicherbereich, in dem die Daten sequentiell vorliegen. Soll ein Bild auf den Monitor ausgegeben werden, so baut dieser das Bild zeilenweise von oben links nach unten rechts auf. Unten angekommen fährt die Elektronenkanone des Monitors in einer Diagonalen wieder nach oben links (Das Zurücksetzen der Elektronenkanone in der Diagonalen wird im Fachjargon auch Vertical Retrace genannt). Dieser Prozess nimmt natürlich einige Zeit in Anspruch, wobei Sie davon ausgehen können, dass die CPU schneller sein wird als die Elektronenkanone des Monitors. Daraus resultiert das Problem, dass Sie ein völlig falsches Bild zu Gesicht bekommen, wenn Sie zu schnell versuchen ein neues Bild auszugeben. Eine Mischung aus zwei oder sogar noch mehreren Bildern wäre wahrscheinlich die Folge. Stattdessen bemüht man einen Backbuffer, auf dem die Daten für den nächsten Frame gespeichert sind. Ein Backbuffer ist ebenfalls eine Surface, die für den Anwender nicht sichtbar ist (Off-Screen Surface). Hingegen befinden sich die sichtbaren Bilddaten auf der Primary Surface oder auch Framebuffer genannt. Während also der Monitor mit dem Aufbau des aktuellen Frames beschäftigt ist, kann im Hintergrund bereits das nächste bzw. die nächsten Bilder berechnet werden, denn es gibt keine Beschränkung auf einen Backbuffer.

Ein weiteres Problem tritt auf, wenn der Inhalt einer Surface gerade auf den Monitor gebracht wird und die Anwendung zum selben Zeitpunkt bereits neue Daten in die Surface schreibt. Aus diesem Grund besitzt eine Surface die Möglichkeit des Locking, d.h. wird die Surface von einer Komponente „gelockt“, besitzt diese exklusive Rechte auf jene Ressource und alle anderen Komponenten müssen warten, bis die Surface wieder freigegeben wird. Sehr hilfreich gestaltet sich dieses Feature ebenfalls bei Anwendungen mit mehreren Threads.

Rasterization

2D-Grafiken liegen im Speicher meist als Bitmap vor, d.h. für jeden Pixel existiert ein Farbwert. Viele diese Pixel mit unterschiedlichen Farbwerten ergeben dann ein Bild. Der Prozess bei dem diese Pixel aus den 3D-Objekten errechnet werden nennt man Rasterisierung oder eben Rasterization. Moderne Grafikkarten verfügen dafür über entsprechende Technik. DirectX 9 bietet keinen Software-Rasterizer an.

Device-Objekt erstellen

Nach einiger Theorie soll es nun in die Praxis gehen. In diesem ersten Beispiel wird eine Instanz der *Device*-Klasse erzeugt. Ein *Device*-Objekt lässt sich als Fundament jeder Direct3D-Anwendung ansehen, denn es fungiert als „Vermittler“ zwischen der Hardware und Ihrer Anwendung. Fügen Sie am Anfang jedes Projekts Verweise auf die Bibliotheken *Microsoft.DirectX* und *Microsoft.Direct X. Direct3D* hinzu, um auf alle nötigen Klassen, Enumerationen etc. zugreifen zu können.

Vor dem Initialisieren des *Device*-Objekts werden dessen Eigenschaften in einer Instanz der Klasse *PresentParameters* festgelegt. Mitunter legen Sie hier fest, ob Ihr Programm später im Fenster- oder im Vollbildmodus ausgeführt wird:

```
PresentParams = New PresentParameters
PresentParams.Windowed = True
PresentParams.SwapEffect = SwapEffect.Discard
```

Durch die erste Zuweisung wird Direct3D mitgeteilt, dass die Anwendung im Fenstermodus laufen soll. Welches Verhalten in Bezug auf die Backbuffer an den Tag gelegt werden soll, definieren Sie mit der zweiten Zuweisung. Der Wert *SwapEffect.Discard* besagt, dass Direct3D nach dem Anzeigen des Backbuffers dessen Inhalt nicht zwingend erhalten muss, d.h. nach der Ausgabe kann in der Off-Screen Surface möglicherweise Datenmüll enthalten sein. In Tabelle 1 finden Sie alle Mitglieder der *SwapEffect*-Enumeration noch einmal erklärt.

Nachdem die Eigenschaften festgelegt sind, wird das *Device*-Objekt initialisiert:

```
objDevice = New Device(0, DeviceType.Hardware, Me,
                       CreateFlags.SoftwareVertexProcessing, _
PresentParams)
```

Zunächst ist ein Grafikadapter anzugeben. Übergeben Sie den Wert *0* um den Default-Adapter zu verwenden. In den meisten Fällen entspricht das der gewünschten Grafikkarte. Ausnahmen ergeben sich nur dann, wenn mehr als ein Mo-

nitor an der Grafikkarte angeschlossen ist oder wenn der Benutzer eine alte 3D-Beschleunigerkarte besitzt, die neben einer 2D-Karte arbeitet. Doch diese Fälle sollen zunächst außer Acht gelassen werden. Das zweite Argument besagt, dass ein Hardware-Device erstellt werden soll, welcher alle Funktionen der Grafikkarte, die im Hardware Abstraction Layer (HAL) definiert sind, benutzen soll. Der HAL kommt vom Hersteller der Grafikkarte in Begleitung der Grafiktreiber. Ihre Anwendung wird niemals direkt in Kontakt mit dem HAL stehen. Als drittes Argument können Sie wahlweise ein Objekt vom Typ *Control* oder ein Handle übergeben, durch den das Render-Ziel definiert wird. Ob die Vertex-Daten der Polygone durch die Hardware und/ oder durch die Software berechnet wird, bestimmt das vorletzte Argument (Tab. 2). Als letztes ist schließlich die Instanz der *PresentParameters*-Klasse mit den gewünschten Eigenschaften zu übergeben.

Zeichnen von Vertexen

Nachdem der Device nun erstellt worden ist, soll es zunächst an die Verarbeitung von Vertexen gehen, wobei Sie noch nicht in Berührung mit Matrizen kommen. Sämtliche Vektoren der Vertexen werden im sogenannten Vertexbuffer gespeichert, weshalb dieser im ersten Schritt initialisiert wird:

```
objVBuffer = New
VertexBuffer(GetType(CustomVertex.TransformedColored),
                                        3, objDevice, _
Usage.SoftwareProcessing, CustomVertex.
                TransformedColored.Format, Pool.Default)
```

Ein Vertex kann mehr als nur die bloßen Koordinaten beinhalten, sondern wie im obigen Fall auch Farbangaben. Direct3D stellt in der Klasse *CustomVertex* allerlei Strukturen für Vertexen unterschiedlichster Art zur Verfügung. Übergeben Sie als erstes Argument den gewünschten Vertex-Typ, anschließend folgt die Anzahl der Vertexen, die Sie in dem Buffer speichern wollen. Der dritte Parameter verlangt nach dem Direct3D-Device. Welche Verwendung Sie für den Vertexbuffer haben, ist durch das vierte Argument zu bestimmen. Im obigen Fall soll Direct3D das Vertex Processing der CPU überlassen. Ist das Flag nicht gesetzt, versucht Direct3D die Berechnungen durch die Hardware auszuführen. Anschließend folgen Angaben zum Format sowie zum Speicherort. Direct3D wählt bei Übergabe des Werts *Default*, abhängig von den gesetzten Flags aus der *Usage*-Enumeration, einen Speicherort aus.

Ein Vertex vom Typ *TransformedColored* besitzt neben den Koordinaten auch Farbangaben. Transformationen müssen nicht vorgenommen werden, denn die Koordinaten beziehen sich bereits auf die Bildschirmkoordinaten, sprich in der linken oberen Ecke befindet sich der Punkt (0, 0). Als erstes gilt es sich exklusive Rechte auf den Vertexbuffer zu sichern, d.h. dieser muss „gelockt“ werden:

```
Dim Verts() As CustomVertex.TransformedColored =
                    CType(objVBuffer.Lock(0, LockFlags.None), _
CustomVertex.TransformedColored())
```

In der obigen Variante verlangt die *Lock()*-Methode lediglich nach dem Offset sowie nach einem Flag aus der *LockFlags*-Enumeration. Als Ergebnis erhalten Sie eine Referenz auf den selektierten Bereich des Vertexbuffers. Da es sich hierbei bereits um einen transformierten Vertex-Typ handelt, ist lediglich die Angabe der X- und Y-Koordinaten sowie ein Farbwert erforderlich:

```
Verts(0).X = 320 : Verts(0).Y = 250
Verts(0).Color = Drawing.Color.Red.ToArgb
```

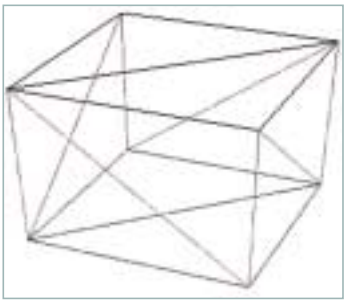


Abb. 2: Aufbau eines Würfels aus Dreiecken

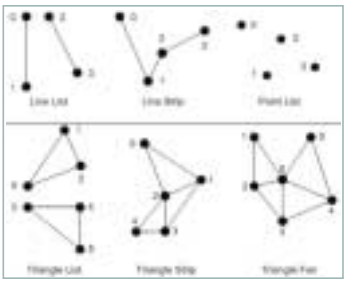


Abb. 3: Arten von 3D-Primitiven in Direct3D

Zum Abschluss müssen Sie die Ressource natürlich wieder freigeben:

```
objVBuffer.Unlock()
```

Nun fehlt nur noch die Methode zum Rendern der Vertex-Daten. Vor jedem Zeichenvorgang soll in diesem Fall der Backbuffer gelöscht und blau eingefärbt werden, wodurch sich ein blauer Hintergrund ergibt. Dazu ist die *Clear()*-Methode des Objekts *Device* zu bemühen, welche nach vier Argumenten verlangt:

```
bjDevice.Clear(ClearFlags.Target, Color.Blue, 1.0F, 1.0F)
```

Wichtig sind im Moment eigentlich nur die ersten beiden Parameter, denn bei

Flag	Beschreibung
Discard	Inhalt des Backbuffers wird nach der Anzeige nicht zwingend beibehalten. In der Debug-Version von DX wird der Backbuffer mit Zufallsdaten überschrieben. Außerdem garantiert *Discard* die effizienteste Anzeige-Methode in Bezug auf Schnelligkeit.
Copy	Der Inhalt des Backbuffers bleibt erhalten. Die Backbufferanzahl ist auf einen beschränkt. Die Methode wartet nicht während des Vertical Retrace.
Flip	Eine Backbufferkette ist erlaubt. Die Indizes reichen von 0 bis n – 1. Wird die Methode zum Anzeigen des neuen Frames ausgeführt, wird der Backbuffer mit Index 0 zum Frontbuffer und der Frontbuffer wird zum Backbuffer mit dem Index n – 1.

Tab. 1: Mitglieder der *SwapEffect*-Enumeration

Flag	Beschreibung
HardwareVertexProcessing	Berechnet Vertex-Daten in der Hardware.
SoftwareVertexProcessing	Berechnungen werden durch die CPU ausgeführt.
MixedVertexProcessing	GPU und CPU berechnen die Vertex-Daten.

Tab. 2: Flags der *CreateFlags*-Enumeration

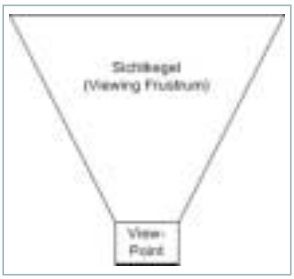


Abb. 4: Projektionskegel von Direct3D

den letzten beiden Parametern handelt es sich um die Werte für den Z-Buffer sowie für den Stencil-Buffer, welche beide erst noch besprochen werden müssen. Übergeben Sie als erstes einen Wert aus der *ClearFlags*-Enumeration (Tab. 3). Anschließend folgt ein Objekt vom Typ *Color* für die Farbbestimmung.

Sämtliche Zeichenoperationen finden zwischen zwei Methodenaufrufen statt. Zum einen wäre das die *BeginScene()*-Methode und zum anderen die *EndScene()*-Methode. Zum Schluss erfolgt zusätzlich ein Aufruf der Methode *Present()*, um die Grafiken auf den Monitor zu bringen:

```
objDevice.BeginScene()

'Zeichenoperationen

objDevice.EndScene()
objDevice.Present()
```

Zunächst muss immer eine Quelle angegeben werden, von der das *Device*-Objekt die Vertex-Daten beziehen kann. Hierzu dient die *SetStreamSource()*-Methode:

```
objDevice.SetStreamSource(0, objVBuffer, 0)
```

Das erste Argument bestimmt die Nummer des Streams, gefolgt von dem Vertexbuffer, der die Daten enthält. Als letztes ist ein Offset anzugeben. Im nächsten Schritt soll das Format der Vertex-Daten dem Device-Objekt mitgeteilt werden:

```
objDevice.VertexFormat = CustomVertex.
                    TransformedColored.Format
```

Zum Abschluss wird nun das Dreieck gezeichnet, wofür Ihnen Direct3D die Methode *DrawPrimitives()* zur Verfügung stellt. Angeben müssen Sie dieser Methode um welchen Typ von Primitive es sich handelt, den Start-Vertex sowie die Anzahl der zu zeichnenden Primitive.

```
objDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, 1)
```

2D-Text ausgeben

Selbst in einem 3D-Spiel sind zweidimensionale Texte von Bedeutung, vor allem in Bezug auf Menüs beispielsweise. Folgende Variante stellt die einfachste Möglichkeit dar, 2D-Texte in Direct3D ausgeben zu können. Zunächst muss ein *Font*-Objekt initialisiert werden, jedoch handelt es sich hierbei nicht um die *Font*-Klasse aus dem *System.Drawing*-Namensraum. Stattdessen muss zusätzlich auf die *Microsoft.DirectX.Direct3DX*-Bibliothek verwiesen werden. Anschließend lässt sich eine Instanz der genannten Klasse aus dem Direct3D-Namensraum erstellen. Dem Konstruktor dieser Klasse sind dabei zwei Argumente zu übergeben: Zum einen wieder das *Device*-Objekt und zum anderen eine Instanz der *System.Drawing.Font*-Klasse:

```
objFont = New Font(objDevice,
                   New System.Drawing.Font("verdana", 12))
```

Mittels der *DrawText()*-Methode des Objekts Font lässt sich der gewünschte Text an passender Stelle auf den Backbuffer zeichnen:

```
objFont.DrawText("Direct3D Text-Beispiel",
                   New Rectangle(0, 0, Me.ClientSize.Width, _
Me.ClientSize.Height), DrawTextFormat.
      Center Or DrawTextFormat.VerticalCenter, Color.White)
```

Ein beliebiger Text ist als erstes Argument zu übergeben, gefolgt von einem Rechteck, in dem der Text platziert wird. Durch den dritten Parameter der Methode ist es Ihnen zusätzlich gestattet festzulegen, wo der String in dem Rechteck ausgerichtet werden soll. Eine Farbe ist durch den letzten Parameter wählbar.

Die dritte Dimension

Als dreidimensionale Version soll nun das vorherige Beispiel umgesetzt werden. Damit die zusätzliche Dimension auch erkennbar wird, lässt die Anwendung mit Hilfe der World-Matrix das Polygon um die Y-Achse rotieren. Erste Änderungen ergeben sich bereits nach dem Initialisieren des *Device*-Objekts:

```
objDevice.RenderState.CullMode = Cull.None
objDevice.RenderState.Lighting = False
```

Jedes Polygon besitzt eine Vorder- und eine Hinterseite. In der Regel lässt Direct3D die Hinterseite unberücksichtigt, es sei denn Sie setzten die *CullMode*-Eigenschaft, wie im obigen Quelltext zu sehen, auf den Wert *None* der *Cull*-Enumeration (Tab. 4). In welche Richtung die Vorderseite eines Polygons zeigt, ist durch den sogenannte Normal Vektor (engl. normal vector) definiert. Gehen Sie beim Festlegen der Vektoren eines Vertex immer im Uhrzeigersinn vor, um die Vorderseite zu erhalten.

Da das Dreieck wieder seine eigenen Farben besitzen soll, muss die Beleuchtung ausgeschaltet werden, andernfalls erschiene das Polygon in schwarzem Gewandt. Eine Zuweisung des Werts *False* an die *Lightning*-Eigenschaft ist ausreichend. Wenn Sie auf das vorherige Beispiel zurückgreifen, bestehen die nächsten Schritte aus dem Abändern der Vertex-Typen, denn bereits transformierte Typen sind hier unbrauchbar. Beachten Sie, dass ein Vertexbuffer immer nur einen Typ von Vertex zur selben Zeit enthalten kann. Benötigen Sie unterschiedliche Typen, so müssen Sie mehrere Buffer in Anspruch nehmen:

Flag	Beschreibung
Stencil	Überschreibt den Stencil-Buffer
Target	Überschreibt den Backbuffer
ZBuffer	Überschreibt den Tiefenspeicher (Z-Buffer)

Tab. 3: Mitglieder der *ClearFlags*-Enumeration

Flag	Beschreibung
None	Hinterseite des Polygons wird angezeigt.
Clockwise	Hinterseiten mit im Uhrzeigersinn geordneten Vertexen werden nicht angezeigt.
CounterClockwise	Hinterseiten der Vertexen die gegen den Uhrzeigersinn angeordnet sind werden nicht angezeigt.

Tab. 4: Mitglieder der *Cull*-Enumeration

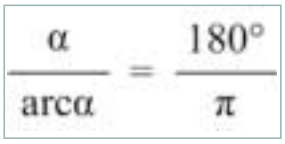


Abb. 5: Gleichung zur Umrechnung von Grad und Radiant

```
objVBuffer = New VertexBuffer(GetType(CustomVertex.
                    PositionColored), 3, objDevice, _
Usage.SoftwareProcessing,
        CustomVertex.PositionColored.Format, Pool.Default)
```

Eine Variable vom Typ *PositionColored* gibt pro Vertex den 3D-Vektor sowie einen Farbwert an. Der Methodenaufruf zum Zugriff auf den Vertexbuffer muss natürlich ebenfalls abgeändert werden, sowie die Koordinatenangaben:

```
Dim Verts() As CustomVertex.PositionColored =
            CType(objVBuffer.Lock(0, LockFlags.None), _
CustomVertex.PositionColored())

Verts(0).X = 0 : Verts(0).Y = 50 : Verts(0).Z = 0
Verts(0).Color = Drawing.Color.Red.ToArgb
```

Als letztes abzuändern gilt es die folgende Codezeile in der Methode *Render()*:

```
objDevice.VertexFormat =
                CustomVertex.PositionColored.Format
```

Zum Abschluss fehlen lediglich noch die unterschiedlichen Einstellungen der Matrizen, welche in der Subroutine *SetupMatrices()* ausgelagert und aus der *Render()*-Methode heraus aufgerufen werden soll. In der *Transforms*-Klasse, zu erreichen über die *Transform*-Eigenschaft des *Device*-Objekts, sind die nötigen Eigenschaften enthalten, um die World-, View- und Projection-Matrix zu setzen. Beginnen Sie mit der World-Matrix.

```
objDevice.Transform.World = Matrix.RotationY(fAngle)
```

Die Matrix-Struktur enthält einige Methoden zum manipulieren von Matrizen. Durch den obigen Quelltext wird das Polygon um seine Y-Achse gedreht bzw. die Welt dreht sich unter diesem. Beachten Sie, dass die Maßeinheit des zu übergebenden Winkels Radiant entspricht und nicht Grad. Mit folgender Gleichung lässt sich das Gradmaß ins Bogenmaß umrechnen. Anschließend folgt die View-Matrix:

```
objDevice.Transform.View = Matrix.LookAtLH
            (New Vector3(0, 3, -100), New Vector3(0, 0, 0), _
New Vector3(0, 1, 0))
```

Für linkshändige kartesische Koordinatensysteme ist die *LookAtLH()*-Methode zum Festlegen der View-Matrix zu verwenden, andernfalls käme die Methode *LookAtRH()* in Frage. In diesem Artikel soll jedoch lediglich das linkshändige Koordinatensystem Verwendung finden. Von Ihnen wird die Übergabe dreier Argumente erwarten, welche die Position der Kamera, den Punkt auf den die Kamera ausgerichtet werden soll und die Richtungsangabe nach oben angeben. Da Direct3D für die Angabe eines Punktes im Raum Vektoren vorsieht, ist die *Vector3*-Struktur zu verwenden. Diese beinhaltet die Felder *X*, *Y* und *Z*. Als letztes kommt die Projection-Matrix an die Reihe:

```
objDevice.Transform.Projection =
            Matrix.PerspectiveFovLH(CSng(Math.PI) / 2, _
1.0F, 1.0F, 200.0F)
```

Durch den ersten Parameter der *PerspectiveFovLH()*-Methode der *Matrix*-Struktur wird der Winkel des Sichtkegels, wie am Anfang des Artikels beschrieben, definiert. Auch hier ist der Winkel wieder in Radiant anzugeben. Das zweite zu übergebende Argument gibt das Größenverhältnis des Sichtfelds an (Aspect Ratio). Dabei wird die Breite durch die Höhe geteilt. Die letzten beiden Parameter bestimmen wie dicht ein Objekt der Kamera kommen darf bis es als hinter dieser gilt bzw. ab welcher Entfernung das entsprechende Objekt nicht mehr auf dem Monitor erscheinen soll. Abschließend können Sie die Primitive wie gewohnt zeichnen lassen.

Der Tiefenspeicher (Z-Buffer)

Sobald eine dritte Dimension ins Spiel kommt, tritt ein neues Problem auf, wenn die Szene gerendert werden soll. Werden die Polygone in einer wahllosen Ordnung gerendert, überdecken möglicherweise Elemente aus dem Hintergrund jene aus dem Vordergrund, woraus natürlich ein verfälschtes Erscheinungsbild resultiert. Eine Lösungsvariante für dieses Problem bietet der Tiefenspeicher, welcher auch als Z-Buffer bezeichnet wird. Dieser existiert neben dem Frontbuffer und enthält Tiefenwerte der einzelnen Pixel. Am Anfang jeder Szene ist es üblich jedem Pixel den größten Tiefenwert zu geben, sprich jeder Pixel befindet sich somit in maximaler Entfernung. Das entspricht dem Wert 1, denn hierbei gelten sogenannte Einheiten-Vektoren (engl. Unit Vector) bei denen 1 den größten Wert darstellt. Vor dem Zeichnen eines neuen Pixels wird nun geprüft, ob der Pixel näher ist als jener im Tiefenspeicher. Ist dies der Fall darf der neue Pixel gerendert werden, andernfalls nicht.

Den Z-Buffer aktivieren

In diesem Beispiel soll der Einsatz des Tiefenspeichers demonstriert werden. Dabei existieren zwei Dreiecke nebeneinander, die sich an einer Ecke überlappen, wobei das linke Polygon an einer ständigen Vor- und Rückwärtsbewegung beteiligt ist. Dadurch ist bei deaktiviertem Z-Buffer erkennbar, dass sich das linke Dreieck niemals hinter dem rechten befindet, obwohl dies der Fall sein müsste. Doch nun zum nötigen Quelltext. Zunächst muss die *Device*-Beschreibung an zwei Punkten verändert werden:

```
PresentParams.EnableAutoDepthStencil = True
PresentParams.AutoDepthStencilFormat = DepthFormat.D16
```

Die erste Zuweisung veranlasst Direct3D dazu automatisch einen Tiefenspeicher anzulegen. Hingegen beschreibt die zweite Zuweisung, dass es sich dabei um einen 16-Bit Z-Buffer handeln soll. Nach dem das *Device*-Objekt initialisiert ist, können Sie den Tiefenspeicher aktivieren und der Anwendung gestatten, in diesem zu schreiben:

```
objDevice.RenderState.ZBufferEnable = False
objDevice.RenderState.ZBufferWriteEnable = False
```

Ausblick

Nachdem in diesem Teil der Artikelserie einige Grundbegriffe geklärt sowie erste Beispiele umgesetzt worden sind, soll es im nächsten Teil um die Hintergründe der Matrizen gehen. Sie werden u.a. sehen, wie Sie mit diesen rechnen können. Außerdem geht es um den Einsatz von Texturen.

Designfragen

Webdesign: Methoden und Technologien

Auch während einer Rezession werden Firmen gegründet und Geschäfte gemacht. Beides bedeutet, dass neue Websites hinzukommen und alte Präsenzen aufgefrischt oder neu gestaltet werden. Doch mit welchen Tools? Welche Möglichkeiten existieren und wie werden sie am besten eingesetzt?

von Malte Kollakowski

Die Frage, wie Websites am besten gestaltet werden sollten ist – immer noch – ungeklärt. Und sie wird es wohl für immer bleiben, denn die Arbeit mit dem Medium Internet und den vorhandenen Tools ist so vielfältig, wie die Erstellung eines neuen Hauses. Es gibt viele Aspekte, die besonderer Kontrolle und Berücksichtigung bedürfen, ansonsten wird statt eines wird aus dem behaglichen Neubau eine kalte und unattraktive Bauruine. Im Gegensatz zum Hausbau, wo man fast zwangsläufig auf die Arbeit von Spezialisten (Elektriker, Klempner, Dachdecker, etc.) angewiesen ist, kann man im Bereich Webdesign alles selbst in die Hand nehmen. Hier treten allerdings die verwendeten Tools weit deutlicher in den Vordergrund, als es beim Hausbau der Fall ist. Natürlich geht es auch dort nicht ohne Spezialwerkzeug, aber beim Webdesign können gute Tools die Arbeit weit schneller und einfacher voranbringen als es beim Bau durch ein paar kräftige Männer, die mal eben anpacken, der Fall sein kann.

Die Tools ...

Im Folgenden werden einige Tools zum Webdesign vorgestellt. Die Liste stellt weder einen Anspruch auf Vollständigkeit, noch kann sie es erreichen. Alleine die namentliche Aufzählung aller möglichen Tools würde den Rahmen dieses Artikels bei Weitem sprengen. Für die Interessenten der weiteren Aspekte des Webdesigns sei an dieser Stelle auf [1], [2], [3], [4] und [5] verwiesen, wo auf diverse spezielle Bereiche eingegangen wird.

Aus den genannten Gründen werden hier nur einige Tools repräsentativ vorgestellt, eine umfangreichere (aber keinesfalls vollständige) Liste von Tools und Editoren findet sich weiter unten unter Links & Literatur. Die vorhandenen Tools bieten inzwischen eine große Fülle von Funktionalitäten, so dass in den Grundfunktionalitäten eigentlich kein Unterschied in der Leistungsfähigkeit festzustellen ist. Wer also nur ein Tool zum Webdesign braucht, sollte sich von seiner Intuition leiten lassen und das Tool nehmen, das ihm am meisten Zusagt.

Adobe Acrobat

Das PDF-Format ist wohl das bedeutendste und am weitesten verbreitete Format für die Weitergabe von Daten. Auch wenn PDF keine eigentliche Technik zur Gestaltung von Webseiten darstellt, ist es anhand seiner Bedeutung durchaus zu den Mitteln zu zählen, die eine Website um Inhalt zu bereichern – vor allem, wenn es sich um Produkte handelt, über die man sich nur informieren möchte, den Kaufakt jedoch an anderer stelle Durchführen will oder sogar muss. Durch die Vorteile von PDF, insbesondere die gute Möglichkeit die Dokumente auszudrucken und die Datei zu speichern, eignet sich das Format für den Benutzer, der etwas von seinem Arbeitsplatz „mitnehmen“ möchte. Vor allem im Service-Bereich, wo z.B. aktuelle Handbücher oder andere Hilfstexte in ansprechender Form dem Kunden angeboten werden sollen, eignen sich PDF-Dokumente in höherem Maße als HTML oder alle anderen Möglichkeiten des Webdesigns. Schließlich kann aus jeder Vorlage mit den entsprechenden Tools – vornehmlich mit dem Acrobat Creator, der inzwischen in der Version 6.0 erhältlich ist – eine PDF-Datei erstellt werden.

Macromedia Flash MX 2004

Die einen lieben es, die anderen hassen es: Flash. Flash bietet dem Anwender die Möglichkeit, komplett interaktive Websites zu erstellen, die auf vektororientierten (Bitmaps, Videos und Sound können natürlich eingebunden werden) Objekten basieren. Dadurch können Flash-Sites sehr aufwändiges Design mit viel Animation und Interaktivität zeigen und gleichzeitig noch sehr Ressourcen schonend mit wenig Bandbreite auskommen.

Web-Puristen werden natürlich bemerken, dass die Browser nicht direkt mit Flash umgehen können und immer erst das Flash-Plugin installiert werden muss, womit immer diejenigen User ausgegrenzt werden, die einen zu den Plugins inkompatiblen Browser besitzen oder aus irgendwelchen Gründen – etwa weil die Installation von Programmen und Tools durch den Arbeitgeber verhindert und verboten ist – keinen Zugriff auf Flash haben können. Es ist also denkbar sinnfrei, die Seiten mit Anleitungen zu Druckern komplett in Flash zu gestalten. Auf der anderen Seite: Flash kann – von kundiger Hand und mit Blick auf die zu erzielende Wirkung erstellt – eine Bereicherung von Websites darstellen. Gewisse Produkte und Services lassen sich erst dann richtig

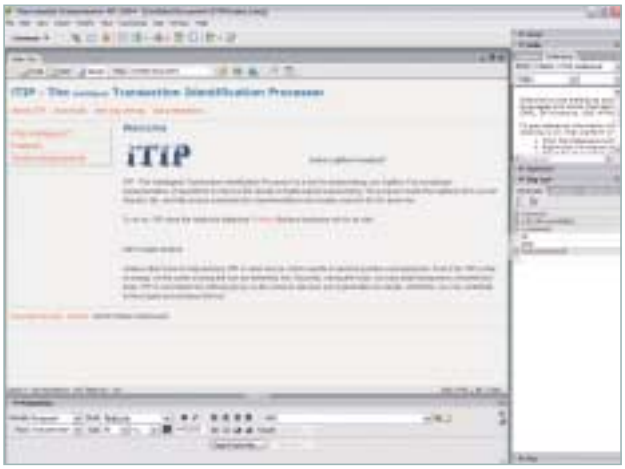


Abb. 1: Die Oberfläche von Dreamweaver MX 2004

anbieten, wenn eine über das in von HTML (oder HTML+JavaScript, etc.) machbare Funktionalität verwendet wird, die eben nur Flash anbietet.

Macromedia Dreamweaver MX 2004

Beim ersten Start des Programms kann der User wählen, ob er die Oberfläche als Designansicht oder als Codeansicht eingestellt haben möchte. Entsprechend der Wahl bekommt der User zur Arbeit eine WYSIWYG-Oberfläche oder eine Editoransicht geboten. Diese Art der Vorkonfiguration erschließt sich einem nicht unbedingt, da während der Arbeit mit dem Programm jederzeit zwischen Design- und Quelltextmodus (wahlweise auch beides gleichzeitig – durch Splitdarstellung) gewechselt werden kann.

Die ehemals heillos in Fenster durcheinander gewürfelte Oberfläche ist ja bereits schon in den letzten Versionen erheblich aufgeräumt worden, in der MX 2004-Version ist es noch einen kleinen Tick besser geworden. Etwas ärgerlich, wenn wohl auch unvermeidlich, sind die überfrachteten Menüs, was der guten Handhabbarkeit von Dreamweaver jedoch kaum einen Abbruch tut.

Zu den erweiterten Funktionen von Dreamweaver gehören eine umfangreiche Unterstützung für die diversen vorhandenen Scriptsprachen und Möglichkeiten zur Gestaltung von Dynamischen Websites. Ob ASP, ASP.NET, PHP, Java Server Pages (JSP) oder ColdFusion, Dreamweaver bringt die passende Unterstützung mit, die allerdings bei den anderen unterstützten Möglichkeiten im Vergleich zu ColdFusion weniger umfangreich ausfällt. Die Arbeit mit Dreamweaver geht weitestgehend intuitiv von statten. Jedes angeklickte Objekt wird in seinen Attributen im unteren Bereich des Arbeitsfensters dargestellt und kann in allen Feinheiten geändert werden.

Ärgerlich – wenn auch verständlich – ist die lästige Produktaktivierung, die man sich bei Macromedia wohl von Microsoft abguckt hat. Derartige Mechanismen verschrecken höchstens User, die ihre Daten nicht jedem zur Verfügung stellen wollen. Andere Tools für Web- und Sitedesign existieren von anderen Herstellern in gleicher oder besserer Qualität (je nach Standpunkt und Vorlieben) und wer die Macromedia-Tools partout nicht käuflich erwerben will, wird entsprechende Mittel und Wege finden.

Macromedia HomeSite 5.5

Über HomeSite gibt es eigentlich nicht viel zu sagen. Zum einen wird das Programm, das ehemals von Allaire entwickelt wurde und dann beim Kauf durch Macromedia in dessen Produktpalette wanderte, nicht oder nur im Sinne von Bugfixes weiterentwickelt. Zum anderen liegt es beim Macromedia Dreamweaver MX mit bei. Zudem ist ein großer Teil des Funktionsumfangs, den HomeSite mitbringt auch in der einen oder anderen Weise (z. T. auch 1:1) im Editorteil von Dreamweaver zu finden. Insofern hat sich zumindest für den Kunden der Kauf von Allaire durch Macromedia gelohnt. HomeSite selbst gilt als eines der besten Tools, um Websites zu gestalten – wenn man das Handwerk im Sinne von „Handwerk“ versteht und kein WYSIWYG zur Webseitengestaltung benötigt.

Macromedia Studio MX 2004

Das Macromedia Studio MX enthält die oben vorgestellten Produkte der MX-Reihe – zusätzlich ist auch ColdFusion MX – selbst die Erstellung von dynamischen Websites ist so direkt möglich, nachdem das umfangreiche Paket auf der Festplatte installiert ist. Zusätzlich finden sich FreeHand MX und FireWorks MX im Bundle. Ersteres eignet sich in seiner Eigenschaft als vektororientiertes Zeichentool hervorragend zum Gestalten des grafischen Designs einer Website (oder wahlweise als Grundlage für die Arbeit in Flash), während man mit FireWorks-Grafiken beliebigen Formats bearbeiten und für andere Anwendungen, wie z.B. Flash, Dreamweaver, etc. vorbereiten kann.

Je nach Sichtweise ist es ein „Rundum-Glücklich-Paket“ oder ein „Verbilligtes Bundle“ – wenn man bei gut 1000,- € von billig sprechen kann. Im Vergleich zum Erwerb der Einzelkomponenten ist es jedoch ein echtes Schnäppchen. Nicht von der Hand zu weisen ist, dass der Besitzer des Macromedia Studio MX 2004 im Wesentlichen für keinen Teilbereich des Website Designs andere Tools benötigt. Was natürlich nicht unbedingt bedeutet, dass er für einige spezielle Anwendungsgebiete oder je nach persönlichen Vorlieben andere Tools zur Rate ziehen wird.

Eine erweiterte Version des Macromedia Studio, genannt Macromedia Studio with Flash Pro enthält – wie der Name bereits andeutet – statt des normalen Macromedia Flash MX 2004 die Professional Version von Flash. Für den Einsatz beim Design von Websites dürfte jedoch die normale Version mehr als ausreichend sein, wenn auch der annähernd gleiche Preis für die etwas leistungsfähigere „Flash Pro“-Variante spricht.

Adobe Golive 6

Golive präsentiert sich in der gewohnten Oberfläche aller Adobeprodukte, sodass sich der bereits in Photoshop oder Elements eingearbeitete User keine Probleme mit der Handhabung des Tools haben dürfte.

Das Design-Fenster beinhaltet jede für die Site nur mögliche Betrachtungsweise,

Abb. 2: Das Arbeitsfenster von Namo 5.5

die über Karteikartenreiter gesteuert wird; von der Layoutansicht im Layout-Editor, über eine Frame-Ansicht, die obligatorische Vorschau und natürlich den „puren" Quellcode ist alles vorhanden, was Designerherzen höher schlagen lässt: Kontrolle Pur – allerdings auch etwas viel Vielfalt und Unübersichtlichkeit. Gerade für Neulinge in dem Produkt ist die Masse der Funktionen und Möglichkeiten einfach nur erdrückend. Was fehlt – dasselbe gilt z.B. auch für Macromedia – ist eine Funktion, mit der man am Anfang alle wahrscheinlich nicht benötigten Funktionen ausblenden kann. Eine Art Anfängermodus wäre direkt eine Innovation. Andere Produkte bieten solche Funktionen.

Namo Interactive Namo 5.5

Namo von Namo Interactive ist dafür bekannt, dass positive und innovative Features aus den anderen bereits vorgestellten Produkten übernommen und um sinnvolle Erweiterungen ergänzt werden. Für vergleichsweise wenig Geld bekommt man so das Beste, was sich bei anderen Designtools, wie z.B. Macromedia HomeSite, Macromedia Dreamweaver oder Adobe GoLive durchgesetzt hat. Namo ist wahrscheinlich mehr das Produkt, das wirklich mit Blick auf die Kunden entwickelt wurde, als es alle anderen bisher vorgestellten Editoren/Designtools von sich behaupten können – auch wenn sich bei denen das inzwischen auch geändert hat. Namo bietet alle Funktionen, die man für die schnelle und komfortable Seitengestaltung benötigt – und ist zudem preisgünstiger als viele der anderen Produkte. Wer wenig Geld ausgeben will und dennoch gut mit seiner Arbeit vorankommen möchte, der sollte sich Namo einmal aus der Nähe betrachten.

Evrsoft 1stPage 2000

Die Freeware 1st Page von Evrsoft ist ein reiner HTML-Editor, der alle wesentlichen Funktionen zum Design komplexer Sites mitbringt. Die Oberfläche kommt den älteren Hasen des Webdesigns bekannt vor, da es sich bei 1st Page um eine Mischung aus dem bekannten HomeSite von Allaire/Macromedia und Paint Shop Pro (rechte Seite mit der Farbpalette). Zusätzlich zu der übersichtlichen Oberfläche bietet das Produkt verschiedene „Schwierigkeitsstufen" an, in denen diverse Funktionalitäten abgeschaltet – oder umgekehrt gesehen – hinzugeschaltet werden können: Easy, Expert und Hardcore. Diese Modi sind während der Arbeit jederzeit frei wählbar und auch beim Programmstart über Shortcuts im Startmenü direkt zu erreichen.

Neben den üblichen Funktionen, wie diversen Assistenten für komplexere HTML-Gebilde bietet das Programm zusätzlich eine große Datenbank von über 450 JavaScripts an – ein Grund mehr, das Programm in seine Bibliothek aufzunehmen.

... und die Techniken

Sich nur auf die Tools zu konzentrieren und dann zu sagen, dass man mit denen alles machen kann, was man will, wäre sicherlich etwas voreilig. Die vorgestellten Tools helfen einem bei der Umsetzung von statischen und zum Teil auch von dynamischen Websites, ohne eine genaue Vorstellung, welche Technologien für die Erstellung von dynamischen Sites vorhanden sind, wird man nicht richtig weiterkommen.

Zunächst ist einmal kann die Wahl der Tools (auch) von der zu verwendenden Plattform für die dynamischen Seiten abhängen. Natürlich kann man, wenn ColdFusion verwendet werden soll, auch mit Adobe GoLive arbeiten. Allerdings werden die Toolseitigen Unterstützungen für diese von Macromedia hergestellte Technologie in den Macromedia-Produkten umfangreicher und auch „passender" ausfallen. Allerdings ist anzumerken, das gerade die großen Tools inzwischen eine fast umfassende Unterstützung von fast allen Technologien anbieten.

Im Folgenden werden einige dieser Möglichkeiten zur dynamischen Websitegestaltung kurz vorgestellt. Je nach Anwendungsgebiet eignet sich die eine oder andere Methode besser oder schlechter als andere. Eine pauschal gute oder schlechte oder sogar einfache oder schwere Möglichkeit existiert hier nicht, die Argumente für oder gegen eine Verwendung der einen oder anderen Technik ist vor Projektbeginn abzuwägen.

ASP

Die Active Server Page Technologie (ASP) wird mit dem Microsoft Internet Information Server mitgeliefert. Für UNIX-Maschinen gibt es auch passende Adaptionen, wie z.B. Sun One Active Server Pages (wwws.sun.com/software/chilisoft/). Mit ASP können dynamische Websites mit Datenbankzugriff erstellt werden.

Die Anwendung von ASP erfolgt über spezielle Tags. Diese werden vor dem Abschicken der angeforderten Seiten an den User vom Server geparst. Anhand der Befehle, die der Server erkennt, generiert dieser die entsprechende Seite und sendet diese an den User. Auf diese Weise lassen sich auf einfache Weise dynamische Seiten, Formulare und Datenbankabfragen verwirklichen.

ColdFusion

Macromedia gibt mit der Cold Fusion Markup Language (CFML) dem Site-Manager dieselben Möglichkeiten der Datenbankanbindung, wie sie auch ASP von Microsoft bietet. Mittels diverser Tags <*cf*...> und </*cf*...> kann der Anwender komplexe Datenbankabfragen, dynamische Seiten und Userverwaltungen im Web realisieren. Ebenso wie bei ASP werden die

Seiten für den User auf dem Server gestaltet und abgeschickt. Wie bereits erwähnt unterstützt Dreamweaver MX 2004 auch direkt die Entwicklung mit ColdFusion.

PHP

PHP ist inzwischen als Version PHP4 am Markt vertreten – als Open Source-Projekt ist es natürlich kostenlos; für die Verwendung spricht jedoch weniger der nicht vorhandene Preis, als eher die Tatsache, dass der Quellcode frei verfügbar ist.

PHP eignet sich besonders für die Erstellung von dynamisch generierten Webseiten, eine Datenbankanbindung – die der von ASP und/oder ColdFusion in nichts nachsteht – ist jedoch vorhanden, sodass sich mit PHP alles realisieren lässt, was auch mit den zuvor genannten Produkten erstellt werden kann. Auch PHP arbeitet wie ASP oder ColdFusion inzwischen mit fast jedem erdenklichen Internetserver (Apache, Internet Information Server, etc.) zusammen und kann auf alle üblichen Datenbanken via ODBC zugreifen. Wie nicht anders zu erwarten ist PHP auch Tagbasiert (*<?php ... ?>*) und lehnt sich in seiner Syntax an C an. Die Funktionen und auch der Funktionsumfang entsprechen grob gesagt in etwa dem von ASP und ColdFusion.

Mit dem Maguma Studio 1.1.2 (www.maguma.com/de/index.html) von Maguma existiert sogar ein speziell für PHP entworfener Editor, der auf alle Fälle einen Blick wert ist.

Java (Java Server Pages / Enterprise Java Beans)

Auch mit Java lassen sich dynamische Webseiten erstellen. Der Aufwand ist hier teilweise erheblich höher als bei den oben erwähnten Lösungen, da richtige Java Programme, Servlets und EJBs programmiert werden müssen. Java ist dafür aber auch in vielen Punkten mächtiger als die doch eher als Scriptsprachen zu bezeichnenden ASP, PHP und ColdFusion.

Die in Java geschriebenen Anwendungen laufen dem Server (ein entsprechender Applicationserver vorausgesetzt) und generieren den sichtbaren Output als HTML-Seite.

Generell besteht die Wahl der Architektur – also zwischen „einfachem" Servlet und einem auf der EJB-Architektur aufsetzenden System. Generell bietet die EJB-Architektur für große Projekte den größeren Nutzen, da nach einer gewissen Einarbeitungszeit die Entwicklung schneller von der Hand geht. Für einfachere Projekte oder solche, die mit großer Wahrscheinlichkeit nicht erweitert werden sollen, eignet sich die EJB-Technologie weniger, da der Overhead bei der Entwicklung zu groß ist. An dieser Stelle sollte mit Servlets gearbeitet werden.

Fazit

Gute Tools zur Erstellung von Webseiten und Websites fallen immer noch nicht vom Himmel. Vor allem preislich hat man zum Teil für die bei größeren Projekten notwendige Qualität und den Funktionsumfang zum Teil recht tief in die Tasche zu greifen. Vielleicht rührt hier auch der Grund her, dass gerade aus dem Open Source-Bereich immer wieder die Meinung zu hören ist, dass ein guter Webdesigner nichts weiter braucht als einen Browser und einen Editor. Für die private Homepage oder kleinere Projekte mag das stimmen, aber für Projekte, die viele tausend einzelne Seiten haben, wäre diese Einstellung schon extrem hinderlich. So viel Zeit und/oder so viel Nerven, solche Projekte ohne automatische Hilfe und weitere Tools durchzuziehen, hat niemand.

Links & Literatur

- Adobe GoLive 5: www.adobe.de/products/golive/
- Evrsoft 1st Page 2000 v2.0: www.evrsoft.com/1stpage/
- Macromedia Dreamweaver MX 2004: www.macromedia.com/software/dreamweaver/
- Macromedia Homesite 5.5: www.macromedia.com/software/homesite/
- Namo WebEditor 5.5: www.namo.com/products/web-editor/
- Globalscape Cute FTP: www.cuteftp.com/
- Ipswitch WS_FTP: www.ipswitch.com/products/file-transfer.html
- SmartFTP: www.smartftp.com
- [1] M. Kollakowski, F. Bensberg: Wege nach Rom; *Der Entwickler* 1/2001
- [2] M. Kollakowski: Beautiful KlickiBunti -Tipps für ein gelungenes Web Design; *Der Entwickler* 1/2002
- [3] F. Bensberg, V. Manthey: Web Intelligence - Analyse des Online-Kundenverhaltens mit Web Intelligence-Tools; *Der Entwickler* 6/2002
- [4] F. Bensberg., V. Manthey, M. Kollakowski: Auf Spurensuche - Marktübersicht: Web Intelligence-Tools; *Der Entwickler* 6/2003
- [5] M. Kollakowski: Qualtitativ Hochwertig? Datenqualitätsmanagement für Web Log Mining; *Der Entwickler* 6/2003

IMPRESSUM

Herausgeber:
Software & Support Verlag GmbH

Anschrift der Redaktion:
Der Entwickler
Software & Support Verlag GmbH
Kennedyallee 87
D-60596 Frankfurt am Main
Tel. +49 (0)69 63 00 89 - 0
Fax. +49 (0)69 63 00 89 - 89
redaktion@derentwickler.de
www.derentwickler.de

Chefredaktion: Masoud Kamali
Redaktion: Maxi Waitzendorfer
Redaktionsassistenz: Mark Hohe-Dorst
Herstellungsleitung: Jens Mainz
Layout, Titel: Dominique Bergmann, Jessica Demirkaya, Tobias Friedberg, Melanie Hahn, Jens Mainz, Sissy Mertens, Maria Rudi

Autoren dieser Ausgabe:
Joachim Dürr, Andre Gildemeister, Timo Göbel, Frank Gunzer, Andrea Held, Max Kleiner, Malte Kollakowski, Jens Konerow, Andreas Kosch, Erdal Kücük, Thomas Pfister, Steffen Rendle, Ralph Steyer, Bob Swart, Bernd Ua

Anzeigenverkauf:
Software & Support Verlag GmbH
Patrik Baumann
Tel. +49 (0)69 63 00 89 - 0
pbaumann@derentwickler.de
Es gilt die Anzeigenpreisliste Nummer 10

Pressevertrieb:
IPV Inland Presse Vertrieb GmbH
Tel.:+49(0)4023711-0,
www.ipv-online.de

Druck: Druckhaus Berlin Mitte

Abo-Service:
Software & Support Verlag GmbH
Tel.: (069) 63 00 89 - 0
Fax.: (069) 63 00 89 - 89
www.derentwickler.de/service

Abonnementpreise der Zeitschrift:

Inland:	6 Ausgaben	€ 29,50
Studentenpreis:	6 Ausgaben	€ 23,60
europ. Ausland:	6 Ausgaben	€ 39,50
Stud. europ. Ausland:	6 Ausgaben	€ 33,60

Abonnementpreise der Profi-CD:

Inland:	6 CD-ROM	€ 72,-
Studentenpreis:	6 CD-ROM	€ 62,-
europ. Ausland:	6 CD-ROM	€ 82,-
Stud. europ. Ausland:	6 CD-ROM	€ 72,-

Zeitschrift + CD

Inland:	6 CD-ROM	€ 99,-
Studentenpreis:	6 CD-ROM	€ 80,-
europ. Ausland:	6 CD-ROM	€ 109,-
Stud. europ. Ausland:	6 CD-ROM	€ 90,-

ISSN: 1619-7941

Erscheinungsweise: zweimonatlich

Inserenten

Interessante Themen in anderen Magazinen des Software & Support Verlags:

dot.net Ausgabe 6.2003 ➠ www.dotnet-magazin.de

- Patterns – Visitor Pattern mit C#
- Patterns – Model/View/Controller mit ASP.NET
- .NET Sicherheit – Code Access Security
- Enterprise – Integration von J2EE, CORBA und .NET mit Borland Janeva

XML magazin & WEB SERVICES Ausgabe 1.2004 ➠ www.xmlmagazin.de

- Special BPM – Prozessmodellierung mit XML
- Sichere Web Services – WS-Security und SAML

LinuX Enterprise Ausgabe 1+2.2004 ➠ www.linuxenterprise.de

- Wireless – Die besten Open Source-Tools und Marktübersicht Router
- Security – Entfernte Betriebssystemerkennung

Javamagazin Ausgabe 1.2004 ➠ www.javamagazin.de

- HTTP-Caching mit Servlets und Java Server Pages
- Deployment – Konfiguration und Zusammenbau von Java Enterprise-Anwendungen

Vorschau

Am 5. Februar 2004 erscheint die nächste Ausgabe:

Lesen Sie darin unter anderem:

Datenbanken im Überblick

Daten müssen in heutigen Unternehmen an vielen Stellen verwaltet, gespeichert und ein schneller Zugriff ermöglicht werden. Deshalb ist eine entsprechende Datenbank das A und O im Unternehmen. Damit Sie nicht den Überblick verlieren und schnell auf die für Sie, wichtigsten Informationen zugreifen können, möchten wir Ihnen einen Überblick über die aktuell verfügbaren Datenbanken mit ihren Features und Neuerungen, im kommerziellen und Open Source-Bereich, geben.

Spiele-Programmierung mit Direct3D

Im zweiten Teil dieser Artikelreihe möchten wir Ihnen, mit einem Blick hinter die Kulissen der Matrizen, den mathematischen Hintergrund von Direct3D näher beleuchten und mit etwas Code festigen. Außerdem bekommt die 3D-Welt ein neues Gesicht, indem Objekte mit Texturen versehen werden. Dabei spielt sowohl Multi-Texturing als auch Transparenz eine Rolle. Für ein schöneres Ambiente sorgen Materialien und dynamische Lichter.

Thread-Programmierung

Wir befinden uns in einem Zeitalter, in dem die Leistung von Prozessoren nicht mehr in Vielfachen von Megahertz, sondern Gigahertz gemessen wird. Trotz dieser rasanten Leistungsentwicklung kommt es einem teilweise so vor, als ob die Software so langsam wie eh und je läuft; der aufmerksame Benutzer schaltet ein Performance-Meter ein und bemerkt dabei, dass der Prozessor in Zeiten höchster Aktivität kaum ausgelastet ist. Warum ist das so? Und warum geht die Arbeit im Zeitalter von Gigahertz-Prozessoren und Hyperthreading immer noch so langsam vor sich? Wie kann man bei seinen eigenen Anwendungen die Leistung erhöhen? Wir gehen der Sache auf den Grund.

Und wie immer dabei:

Unsere Rubriken Delphi, C# & C++, .NET, Datenbanken und Tipps & Tricks!

Aus aktuellem Anlass kann es zur Verschiebung von Artikeln kommen.