

Im Alltag angekommen

Im letzten Teil dieser Artikelserie runden Sie Ihre Testumgebung ab und lassen die Terminal-Anwendungen hinter sich. Ziel ist eine GUI-Anwendung, die künftig als Testrahmen für eigene Versuche dient. Den Zugriff auf eine SQL-Datenbank gibt es dazu inklusive.

In den ersten beiden Teilen dieser Artikelserie [1], [2] haben Sie gelernt, eine Mono-Umgebung einzurichten, und Sie haben erste kleine Anwendungen sowohl unter Windows als auch unter Mono zum Laufen gebracht. Das hat Ihnen hoffentlich Lust auf mehr gemacht. Falls ja, ist es nun an der Zeit, ernsthaftere Aufgaben anzugehen.

Zu Beginn gleich ein kleiner Wermutstropfen: Auf den im vorigen Teil angekündigten Webservice, der mit der 2.0-Run-time läuft, müssen Sie leider noch verzichten.

Die Implementierung in Mono ist schlicht noch nicht so weit gekommen. Zur Zeit wird mit Nachdruck an der fehlenden Implementierung gearbeitet.

Auf einen Blick

Autor



Jan Waiz programmiert seit über 15 Jahren mit Clipper, Visual-Objects, C++, C# und mit verschiedensten Datenbanken von DBF bis hin zu SQL. Außerdem unterstützt er IT-Häuser auf den unterschiedlichsten Ebenen bei Projekten. Sie erreichen ihn über j.waiz@icomedv.de.

dotnetpro.code
A0703MonoApps



Sprachen C#

Technik ASP.NET

Voraussetzungen Visual Studio 2003/2005, VMWare, Mono

Serie

1. Installation und erster Test
2. ASP.NET und Webservices
3. **GUI-Applikationen und Datenbanken**

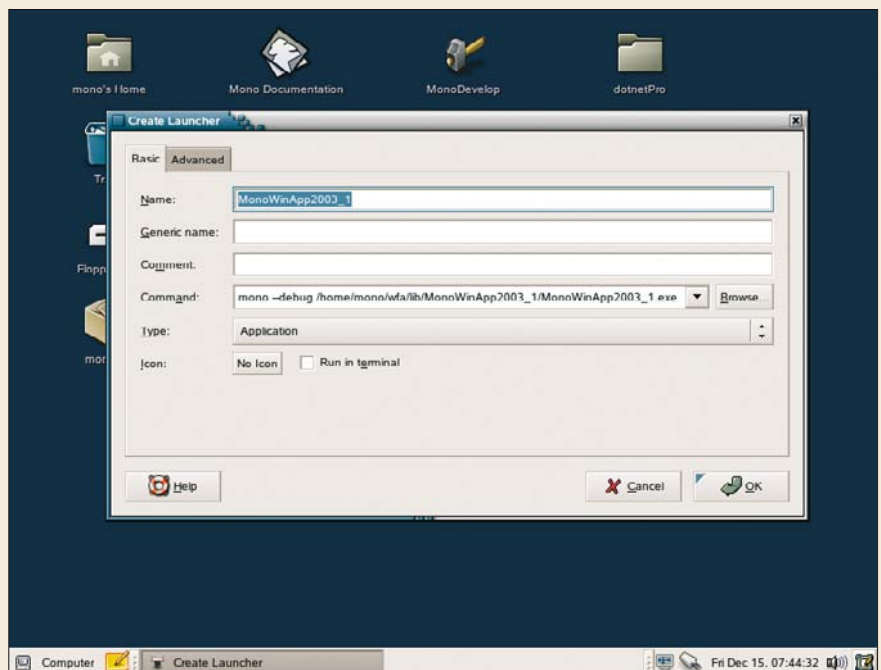


Abbildung 1 Verknüpfungen auf Linuxianisch: Hier heißen sie Launcher.

Mono auf Stand bringen

Nach wie vor schreitet die Entwicklung von Mono in schnellen Schritten voran. Zu dem Zeitpunkt, da dieser Artikel entstanden ist, stand auf der Mono-Homepage ein VMWare-Image mit der Mono-Version 1.2.2.1 zum Download bereit [3]. Da jedes Update dieses Images Fehler beseitigt und neue Merkmale enthält, empfiehlt sich ein Download des gerade aktuellen Images.

Herunterladen und Einrichten des Images sind schnell erledigt; aber das Erstellen der Verzeichnisse für alle hier genannten Programmbeispiele ist auf die Dauer ein lästiges Unterfangen. Auf der Heft-CD finden Sie daher eine aktualisierte Batch-Datei *GoMono.txt* – eine Skriptdatei für WinSCP, das mit deren Unterstützung alle erforderlichen Verzeichnisse auf der VM einrichtet, bevor die jeweiligen Dateien kopiert werden.

Des Weiteren finden Sie auf der Heft-CD ein Verzeichnis *dotnetPro*, das die Verknüpfungen für den Linux-Desktop enthält. Selbstverständlich werden diese von der Skriptdatei ebenfalls kopiert.

Apropos Desktop-Verknüpfung: Sollten Sie nach dem Kopieren derselben und einem Doppelklick darauf im Editor landen, anstatt nach einem *Run in Terminal* gefragt zu werden, öffnen Sie per Rechtsklick die Eigenschaftskarte *Permissions*; aktivieren Sie das Kontrollkästchen *Execute – Allow executing File as Program*. Dabei kann es allerdings zu einem Rechte-Problem kommen.

Die richtigen Rechte

Die installierte VM verfügt über zwei eingerichtete Benutzer: *root* und *mono*, wobei ersterer einem Windows-Administrator und der zweite einem Benutzer mit

eingeschränkten Rechten entspricht. In der VM in Gnome sind Sie beim Start der VM automatisch als Benutzer *mono* angemeldet. In WinSCP verwenden Sie zum Anmelden jedoch den *root*-Account. Daher ist linuxseitig der Benutzer *root* der Eigentümer der per Skript kopierten Dateien und Verzeichnisse und nicht der Benutzer *mono*, als der Sie am Gnome-Desktop angemeldet sind. Das führt dazu, dass Sie das oben genannte Kontrollkästchen nicht ändern können.

Eine Lösung ist, sich bei Gnome ab-, als *root* wieder anzumelden und das Häkchen zu setzen. Aber es geht auch einfacher, denn per WinSCP sind sie ja bereits als *root* angemeldet. Wechseln Sie einfach in WinSCP in der rechten Dateiliste in das Verzeichnis */home/mono/Desktop/dotnet-Pro/*, klicken Sie auf die Datei *MonoKonsol2003* und öffnen Sie ihre Eigenschaften. Unter *Permissions* können Sie nun die Zugriffsstufe *X* setzen; *X* steht für „Execute“ und gestattet dem User, die betreffende Datei auszuführen. Zurück in der VM öffnen Sie den Order *dotnetPro* auf dem Desktop, aktualisieren ihn einmal mit [F5] – nun sollten die Konsolenverknüpfungen korrekt funktionieren.

Eine weitere elegante Möglichkeit ist, dass Sie sich in WinSCP ein weiteres Login für den Benutzer *mono* einrichten. Verwenden Sie dieses für das Kopieren der Dateien und den Aufruf der Skriptdatei, indem Sie der Windows-Verknüpfung als letzten Parameter den Profilenames hinzufügen, unter dem Sie die *mono*-Anmeldung in WinSCP gespeichert haben. Wie das geht, kennen Sie bereits aus dem ersten Teil und dem Abschnitt „WinSCP installieren“ [1].

Sofern Sie bereits die Dateien als *root* kopiert haben, löschen Sie die kopierten Dateien und Verzeichnisse per WinSCP mit dem *root*-Account und führen das Skript künftig unter dem *mono*-Account erneut aus. Ein jungfräuliches Image ist somit in Null-Komma-Nichts eingerichtet und mit allen erforderlichen Verzeichnissen, Verknüpfungen und Dateien versehen.

Verknüpfungen unter Linux

Bevor es mit dem Thema dieses dritten Teils losgeht, noch ein bisschen Gnome-Kosmetik. Bisher wurden die Beispiele über die Shell gestartet. Die ständige Abfrage der Dialogbox sowie das im Hintergrund laufende Terminal sind nicht besonders elegant. Doch es gibt ein Pen-

dant zu einer Windows-Verknüpfung – und funktioniert unter Gnome genauso einfach wie unter Windows.

Klicken Sie mit der rechten Maustaste auf den Linux-Desktop und wählen Sie im Kontextmenü *Create Launcher*. Abbildung 1 zeigt den Dialog, mit dem Sie dann die Verknüpfung einrichten. Es genügt die Angabe eines Namen sowie die entsprechende Kommandozeile. Letztere entspricht derjenigen aus dem Skript. Nachdem Sie die Angaben eingetragen und den Dialog mit *OK* beendet haben, verfügen Sie über ein neues Icon auf dem Desktop, das Sie bei Bedarf einfach per Drag-and-drop in den *dotnetPro*-Ordner ziehen können. Dann können Sie Ihre Anwendungen mit einem Doppelklick und ohne die lästige Terminal-Abfrage starten.

Falls Sie sich übrigens bei den gezeigten Screenshots darüber wundern, dass Ihr Gnome-Desktop anders aussieht, sollten Sie des Spaßes halber einmal das Control-Center öffnen und ein wenig mit den Themes spielen.

Datenbanken sind gefragt

Die meisten Windows-Applikationen arbeiten in irgendeiner Form mit Daten. SQL-Datenbanken haben sich dabei bewährt und gehören zum täglichen Handwerkszeug eines Programmierers. Daher ist es natürlich sehr interessant, wie es sich mit Linux, Mono und den dortigen SQL-Datenbanken verhält. Sie oder Ihr Chef haben sicherlich schon mehrfach die Augenbrauen hochgezogen in Anbetracht der nicht unerheblichen Lizenzkosten der typischen SQL-Datenbanken unter Windows. Dabei sind diese „großen Tanker“ oft gar nicht erforderlich. Die MSDE ist allerdings auch nicht immer eine Alternative, da sie bestimmten Beschränkungen unterliegt.

Dass dennoch auf „Tanker“ zurückgegriffen wird, liegt sicherlich nicht selten an Bequemlichkeit und Gewohnheit, zumal die wirklich interessanten Alternativen wie PostgreSQL oder MySQL auch noch auf einem fremden Betriebssystem laufen. Dabei spielt das Linux/Mono-Gespinn in Sachen Datenbanken erst seine volle Stärke aus, denn es gibt hier mittlerweile sehr ausgereifte und leistungsfähige SQL-Lösungen – kostenlos!

Das Mono-Team hat die VM bereits mit einer MySQL- und einer PostgreSQL-Datenbank ausgestattet. Die Tests und Beispielanwendungen dieser Serie verwenden PostgreSQL. Um auch hierbei der

bisherigen Devise treu zu bleiben und die administrativen Zugriffe auf die Datenbank bequem von Windows aus erledigen zu können, ist ein weiteres Tool und ein wenig Konfigurationsarbeit erforderlich. Aber nur keine Angst, das ist alles recht einfach.

PostgreSQL konfigurieren

Die PostgreSQL-Datenbank der VM ist so konfiguriert, dass sie keine Verbindungen von außen akzeptiert. Dies lässt sich aber schnell ändern. Starten Sie dazu wieder WinSCP und loggen Sie sich bei der VM als *root* ein. Wechseln Sie in das Verzeichnis */var/lib/pgsql/data/* und öffnen Sie mit [F4] die Datei *postgresql.conf*, die die Konfigurationsdaten für das Datenbanksystem enthält. Listing 1 zeigt Ihnen, in welchen Sektionen Sie welche Änderungen vornehmen müssen. Für alle Parameter finden Sie auskommentierte Werte, die Sie nur entsprechend anzupassen brauchen.

In demselben Verzeichnis finden Sie auch die Datei *pg_hba.conf*. Diese benötigt im Abschnitt *Ipv4* einen zusätzlichen Eintrag, den Sie in Listing 2 sehen. Passen Sie die IP-Angabe *192.168.1.0* bei Bedarf an Ihren gültigen IP-Bereich an, wobei die letzte Zahl eine *0* ist.

Die Einstellungen sind in dieser Form nicht für ein Produktivsystem geeignet.

Listing 1

Die Datei *postgresql.conf*.

```
# -----  
# File Locations:  
# -----  
# dotnetpro  
data_directory = '/var/lib/pgsql/data'  
hba_file = '/var/lib/pgsql/data/pg_hba.conf'  
  
# -----  
# Connections and Authentication  
# -----  
# dotnetpro  
listen_addresses = '*'  
port = 5432
```

Listing 2

Die Datei *pg_hba.conf*

```
# IPv4 local connections:  
# dotnetPro  
host all all 192.168.1.0 255.255.255.0 trust
```

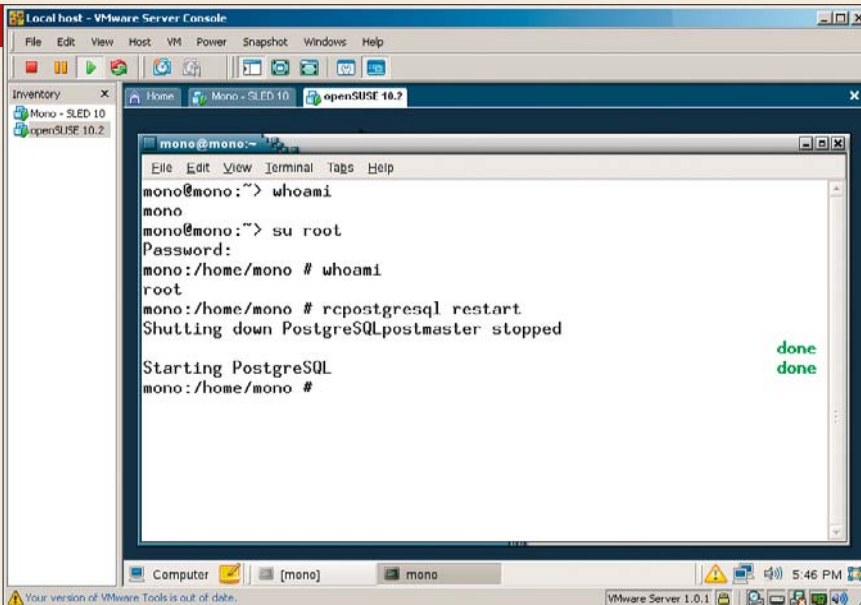


Abbildung 2 So starten Sie PostgreSQL über ein Terminalfenster neu.

Die Angabe `listen_addresses = ,*` akzeptiert zum Beispiel alle eingehenden IP-Adressen. Dies ist kein Problem, denn hier sollen Sie mit einer möglichst unkomplizierten Testumgebung arbeiten, um möglichst einfach zum Ziel zu kommen. Für den produktiven Einsatz empfiehlt es sich dagegen aus Sicherheitsgründen, sich eingehend mit den vielfältigen Parametern zu beschäftigen.

Damit PostgreSQL die Änderungen akzeptiert, müssen Sie die Datenbank neu starten. Sie können dazu die komplette VM neu starten, aber es geht einfacher: Wechseln Sie in die VM und öffnen Sie ein Terminalfenster. Sie können über den Befehl `whoami` feststellen, unter welchem Benutzerkonto Sie in der Konsole ange-

meldet sind, normalerweise `mono`. Den Benutzerkontext können Sie mit dem Befehl `su root` und dem Passwort `mono` einfach auf die erforderlichen `root`-Rechte umschalten. Ein Neustart der PostgreSQL ist dann über `rpostgresql restart` eine einfache Sache. Abbildung 2 zeigt Ihnen ein Terminalfenster mit den Eingaben und den entsprechenden Rückmeldungen.

pgAdminIII

Nach dem Neustart des Datenbanksystems sind die Voraussetzungen geschaffen, um es bequem von Windows aus zu verwalten. Dazu benötigen Sie allerdings noch das kostenlose Tool pgAdminIII, das Sie im Internet von der PostgreSQL-

Homepage herunterladen können [4]. Nach der Installation unter Windows und dem Start des Tools klicken Sie rechts im TreeView-Element auf den Ast `Server` oder wählen im Menü `Datei/Server hinzufügen`. Im Dialog `Neuer Server` setzen Sie die Einträge so, wie Sie es in Abbildung 3 sehen, wobei Sie die IP-Adresse an Ihre Gegebenheiten, also Ihre VM anpassen. Ein Passwort für den Benutzer `mono` ist nicht erforderlich, sodass Sie das entsprechende Kontrollkästchen deaktivieren können. Ein Klick auf `OK` sollte dazu führen, dass sich das Tool pgAdminIII bei PostgreSQL auf der VM anmelden kann.

Wenn Sie sich nun anmelden und sich in dem Tool durch den `Server`-Baum bewegen, werden Ihnen sicherlich im Abschnitt `Datenbanken` die bereits vorhandenen Datenbanken `mono` und `postgres` auffallen. Diese sollen Sie erstmal nicht stören, denn sie richten nun eine eigene Datenbank ein.

Über das Kontextmenü des Astes `Datenbanken` oder über das Menü `Editieren/Neues Objekt/Neue Datenbank` gelangen Sie in den Dialog zum Einrichten einer neuen Datenbank. Abbildung 4 zeigt Ihnen, mit welchen Angaben Sie diese erstellen sollten.

Ohne Tabellen macht eine Datenbank wenig Sinn. Viele Wege führen hier zum Ziel. Natürlich lassen sich Tabellen programmgesteuert aus der Applikation heraus erzeugen. Ein anderer Weg führt über pgAdminIII und das Kontextmenü des Astes `Tabellen`. Diesen erreichen Sie in der gerade erstellten `dotnetPro`-Datenbank über `Schemata/public`. Eine weitere Möglichkeit bietet ein SQL-Skript. Um ein solches zu erstellen, öffnen Sie das

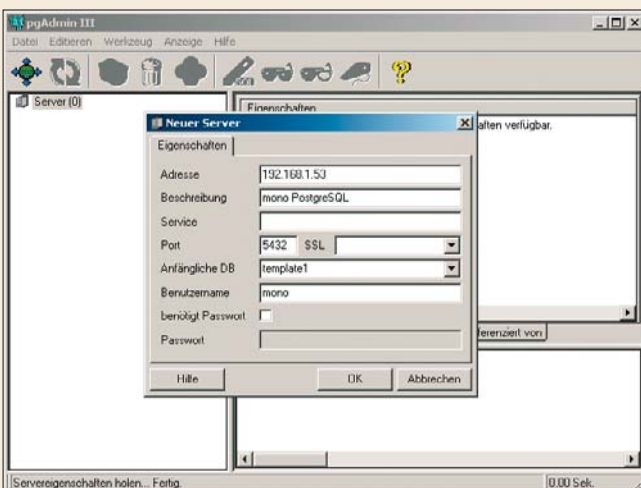


Abbildung 3 Mithilfe von pgAdminIII können Sie die Linux-Datenbank PostgreSQL von Windows aus verwalten.

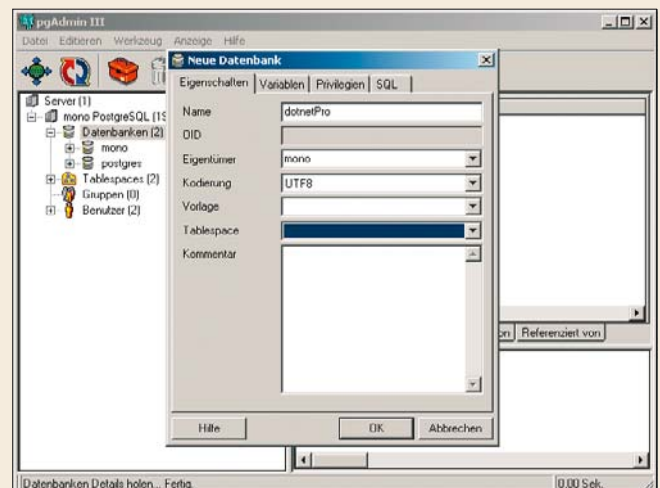


Abbildung 4 Mit pgAdminIII eine neue PostgreSQL-Datenbank erzeugen.

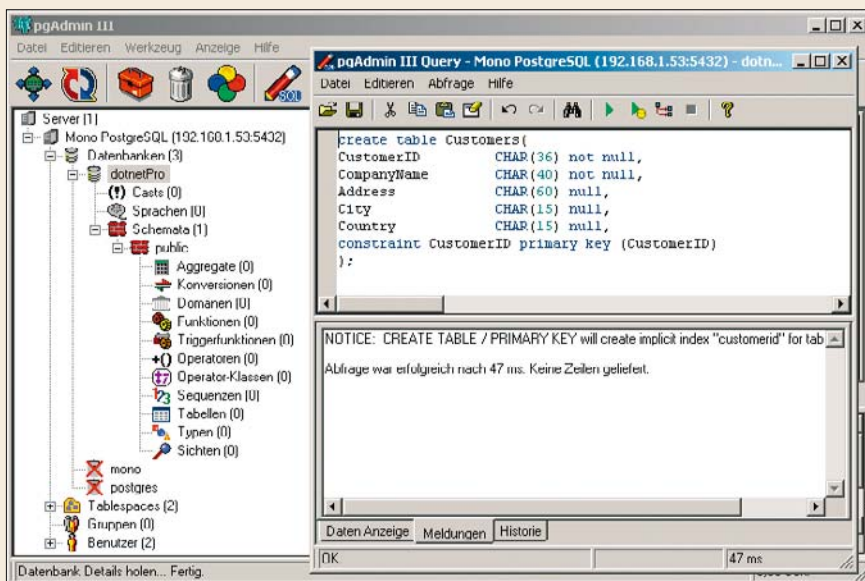


Abbildung 5 Mit dem Abfragetool von pgAdminIII können Sie per SQL Tabellen erzeugen.

Abfragetool von pgAdminIII über das Menü *Werkzeug/Abfragetool* oder über das entsprechende Symbol auf der Toolbar des Programms. Im Abfragetool öffnen Sie das Skript *pgsqlCreate.sql*, das Sie auf der Heft-CD finden. Es erzeugt eine kleine Tabelle namens *Customers* mit ein paar Spalten. Es steht Ihnen natürlich frei, komplexere Beispiele auszuprobieren. In Abbildung 5 sehen Sie das Abfragetool nach erfolgreicher Ausführung des *Create*-Statements. Wenn Sie es schließen, steht Ihnen in pgAdminIII nun die Tabelle *Customers* zur Verfügung.

Was noch fehlt, sind einige Testdaten. Öffnen Sie im Abfragetool die Datei *PgSqlInsert.sql*, die Sie ebenfalls auf der Heft-CD finden. Nach Ausführung des Skripts verfügt PostgreSQL nun über eine Datenbank namens *dotnetPro* mit der Tabelle *Customers* und ein paar Testdaten.

So weit, so gut. Das war eine Menge Stoff bis hierhin. Bevor Sie die Früchte der Arbeit ernten und mit einer Testanwendung darauf zugreifen können, ist es Zeit für einen kurzen Zwischenbericht:

- In WinSCP haben Sie ein zweites Login für den *mono*-Account zum Kopieren der Dateien angelegt.
- Sie haben die Konfigurationsdateien *postgresql.conf* und *pg_hba.conf* auf der VM angepasst.
- Sie haben PostgreSQL neu gestartet, damit die Änderungen an der Konfiguration wirksam wurden.
- Mit dem Tool pgAdminIII haben Sie unter Windows eine Verbindung zu PostgreSQL auf der VM hergestellt.

- Außerdem haben sie mithilfe des Tools auch gleich die Datenbank *dotnetPro* erzeugt, die Tabelle *Customers* hinzugefügt und ein paar Testdaten eingespielt.

Die Testanwendung

Und schon geht es weiter. Auf der Heft-CD finden Sie die fertige Testapplikation *MonoWinApp2003_1* für Visual Studio 2003, die Sie auf Ihrem PC in das Verzeichnis *C:\dotnetPro\GoMono\MonoWinApp2003_1* kopieren und in der IDE öffnen. Wenn Sie nun gleich versuchen, die Applikation zu übersetzen, werden Sie diverse Fehlermeldungen erhalten, die sich auf fehlende *Npgsql*-Namensbereiche beziehen. *Npgsql*?

Für die Zugriffe auf PostgreSQL könnten Sie auf Standardklassen zurückgreifen. Für die Verwendung in Verbindung mit einer PostgreSQL empfiehlt sich aber der Einsatz des *Npgsql*-Datentreibers, da dieser auf PostgreSQL optimiert ist. Sie können diesen ebenfalls von der PostgreSQL-Homepage herunterladen [4] und installieren. Fügen Sie anschließend in Ihrem Projekt einfach einen Verweis auf die Datei *Npgsql.dll* aus dem Installationsverzeichnis hinzu und die Solution lässt sich fehlerfrei erstellen.

Sie könnten aber auch die ODBC-Klassen verwenden, müssten dazu aber das entsprechende Paket unter Linux nachinstallieren, da dies in der VM nicht eingerichtet ist. Die Einbindung des *Npgsql*-Providers dürfte da wesentlich einfacher

sein. Außerdem wird dessen Verwendung ohnehin empfohlen.

Auf der Heft-CD finden Sie die aktuelle Skriptdatei *GoMono.txt*, mit der Sie die VS-Applikation bequem auf die VM kopieren können. Mehr zu diesem Skript finden Sie in den ersten beiden Teilen dieser Serie. Das Skript erzeugt in der VM in dem Verzeichnis */home/mono/Desktop/dotnetPro/* außerdem eine Verknüpfung; Sie sollten danach in der VM im Ordner *dotnetPro* ein entsprechendes Icon sehen, über das Sie die Applikation in der VM starten können. Abbildung 6 zeigt Ihnen die Anwendung unter Windows und in der VM.

Vielleicht sind Sie nun ein wenig erstaunt: Eine GUI unter Mono? In der Tat laufen auch Applikationen mit einer grafischen Oberfläche problemlos unter Mono. Wohlgermerkt: Die Anwendung wurde in Visual Studio erstellt und somit auch die Formulare. Auch die Events funktionieren einwandfrei, wie Sie bei einem Blick in die Quellcodes feststellen können. Sie finden nicht eine Zeile Extracode, der die Runtime betreffen würde.

Auf der ersten Karteikarte *Environment* gibt die Testanwendung ein paar Informationen zu dem System aus, unter dem sie läuft. Eine kleine Informationsklasse liefert die entsprechenden Ergebnisse – nichts Aufregendes, sondern reine Information.

Die zweite Karteikarte beherbergt den aus Teil 2 bekannten Webservice. Dies zeigt, dass Sie auch aus einer GUI-Applikation ohne Hindernisse auf Webdienste zugreifen können und die Anwendung nicht in irgendeiner Weise speziell an die Mono-Runtime anpassen müssen.

Einfacher Datenbankzugriff

Interessanter wird es auf dem dritten Register *SQL-Access* (Abbildung 7). Im Quellcode finden Sie den relevanten Teil in der Prozedur *btnConnect_Click()*. Wie Sie sehen, unterscheidet sich der Zugriff auf den SQL Server und der auf PostgreSQL via *Npgsql* lediglich in den verwendeten Klassen, ist aber sonst gleich.

Da die *Npgsql*-Klassen über die gleichen Interfaces verfügen wie die *Sql*-Klassen des .NET Frameworks, ist die Kapselung in eigenen Klassen kein Problem. Im Internet gibt es genug Beispiele, die zeigen, wie sich zur Laufzeit der verwendete Daten-Provider dynamisch ändern lässt. Die *Npgsql*-Klassen fügen sich dabei nahtlos ein.

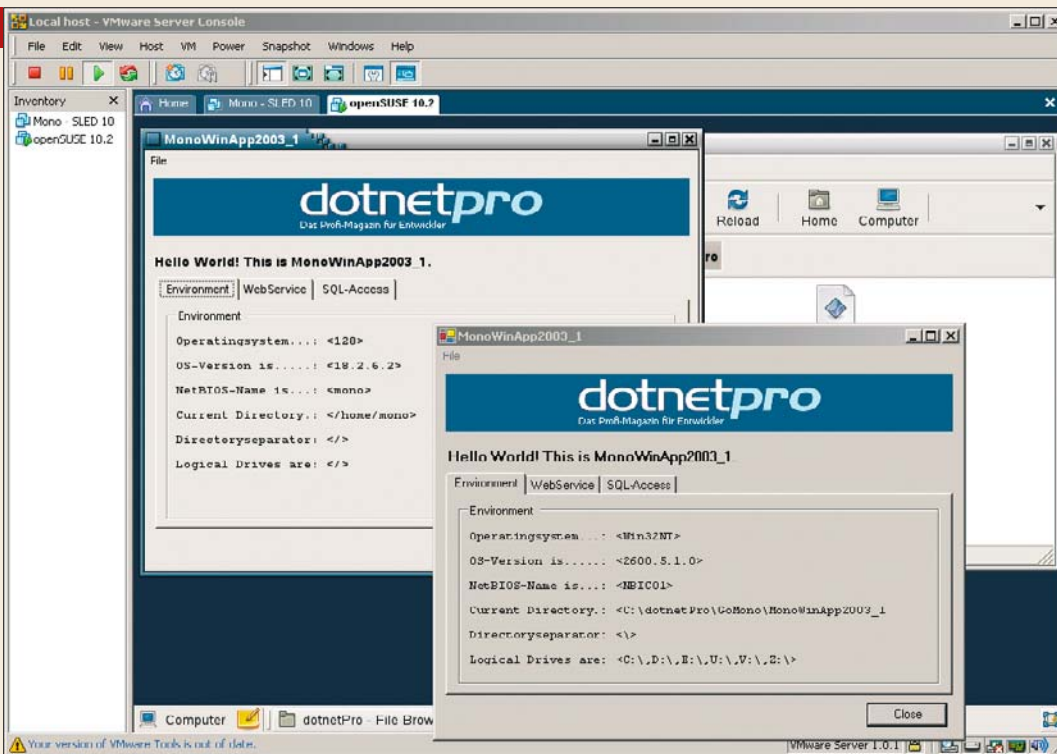


Abbildung 6 Die Testanwendung läuft gleichzeitig unter Windows und unter Mono.

Starten Sie nun die Anwendung unter Windows bei laufender VM und wechseln Sie im Kombinationslistenfeld auf der Karteikarte *SQL-Access* den zu verwendenden Daten-Provider. Die Textbox zeigt daraufhin die jeweilige Verbindungszeichenfolge an, die Sie hier manuell ändern können. Dauerhafte Änderungen, etwa bei der IP-Adresse, können Sie natürlich auch im Quellcode vornehmen. Sie finden den entsprechenden Abschnitt im Konstruktor des Formulars. Die Schaltfläche *Connect* stellt eine Verbindung zu der jeweiligen SQL-Datenbank her und gibt die Daten der Tabelle *Customer* aus.

Versuchen Sie doch einmal bei laufender VM aus Windows heraus per *Npgsql* auf PostgreSQL zuzugreifen. Sie werden feststellen, dass es keinerlei Probleme bereitet. Das Lesen des Quellcodes dürfte keine Probleme bereiten. Die Lösung ist deshalb bewusst einfach gehalten.

Die Anwendung unter .NET 2.0

Auf der Heft-CD finden Sie auch die Applikation *MonoWinApp2005_1*, die Sie in Ihr lokales Verzeichnis *C:\dotnetPro\GoMono\MonoWinApp2005_1* kopieren; öffnen Sie die Projektmappe in Visual Studio 2005. Sie werden feststellen, dass sie in Aussehen und Funktion im Wesentlichen identisch zu der 2003er Version ist. Wenn Sie das Projekt mithilfe von

WinSCP nach Mono kopieren und laufen lassen, stellt sich Folgendes heraus:

GUI-Applikationen aus dem 2.0-Framework lassen sich bereits unter Mono ausführen (Abbildung 8). Aber es gibt noch kleinere Unsauberkeiten bei der Darstellung und Funktion. Dies betrifft zum Beispiel *Resize-Events*: eine annähernd korrekte Darstellung der GUI-Elemente stellt sich erst nach einem manuellen *Resize* der GUI ein.

Und das, obwohl eine Überprüfung mit dem Tool *MoMA* – sie kennen es bereits aus dem letzten Teil der Serie – keine Unverträglichkeiten ausgewiesen hat. Das beweist lediglich, dass auch ein derartiges Tool, so hilfreich es auch ist, nur erste Anhaltspunkte liefern kann und einen Test der eigenen Applikationen unter Mono nicht ersetzen kann.

Die Umsetzung des 2.0-Frameworks ist aber noch nicht abgeschlossen und es könnte sein, dass zu dem Zeitpunkt, da Sie diesen Artikel in der *dotnetpro* lesen, bereits eine neue Mono-Version auf dem Server des Projekts verfügbar ist, die dieses Probleme möglicherweise behebt und neue Merkmale enthält. Das ist gerade das Tolle an dem *VMWare-Image*: Mit ihm und mit den Tools, die Sie im Lauf dieser Serie kennengelernt haben, können Sie problemlos und schnell testen, wie sich Ihre Anwendungen unter Mono verhält.

Der Stand der Dinge

Fassen wir zusammen, was wir im Laufe der Artikelserie gesammelt haben:

- Das Virtualisierungssystem *VMWare*,
- ein *VMWare-Image* mit Linux und Mono,
- das Tool *MoMA* für die schnelle Überprüfung auf Mono-Tauglichkeit,
- das FTP-Programm *WinSCP* zum Datenaustausch zwischen Windows und Linux,
- das Tool *pgAdminIII* für die Administration der *PostgreSQL-Datenbank* auf der VM von Windows aus,
- den *Npgsql-Datentreiber* für den Zugriff auf *PostgreSQL*
- und Testprogramme.

Sie sind somit in der Lage, schnell und einfach Testszenarien zu schaffen um festzustellen, ob eine .NET-Anwendung unter Mono läuft. Ist erst einmal alles installiert und eingerichtet, beschränkt sich der Aufwand auf wenige Minuten. Das Ergebnis:

- Sie entwickeln Programme von vornherein multiplattformfähig.
- Sie erfahren zu einem sehr frühen Zeitpunkt von eventuellen Inkompatibilitäten zu Mono.
- Das versetzt Sie in die Lage, auf Anforderungen flexibel reagieren und in Zeiten zunehmenden Kostendrucks dennoch für Kunden praktikable und

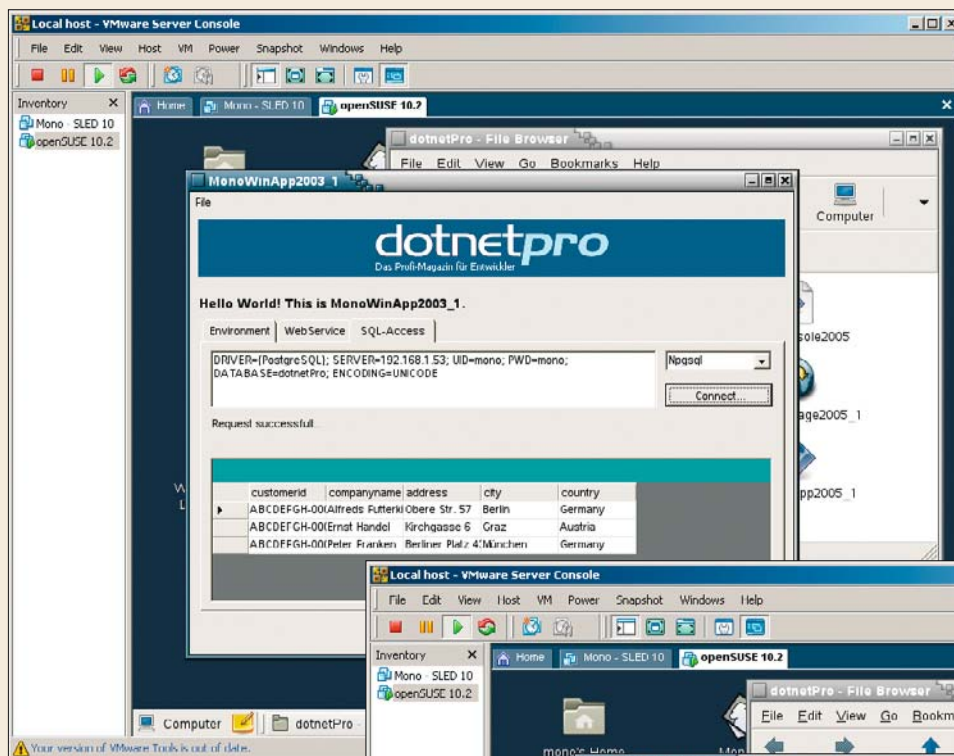


Abbildung 7 Auf der Karteikarte SQL-Access finden Sie die eigentlichen Datenbankfunktionen der Anwendung.

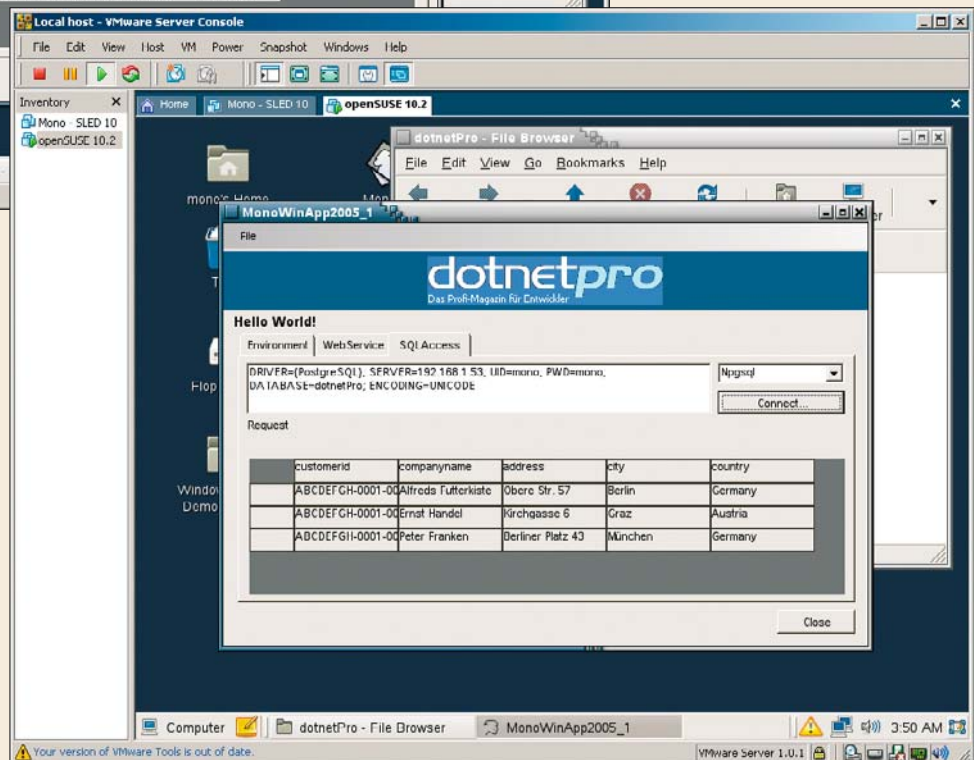


Abbildung 8 Die Testanwendung in der 2005er Variante unter Mono.

wirtschaftliche Lösungen erstellen und Alternativen anbieten zu können.

Zugegeben: Installation und Administration eines produktiven Linuxsystems in Verbindung mit einer SQL-Datenbank und Mono stehen nochmal auf einer anderen Stufe und sind ebenso anspruchsvoll wie bei Windows. Dies mag zunächst abschrecken, sollte Sie aber nicht davon abhalten, Linux und Mono im Auge zu behalten. Für Tests ist die VM eine ideale Plattform und bei Bedarf wenden Sie sich zum Beispiel an eine der vielen Linux-Usergroups, die es in jeder größeren Stadt gibt. Sie werden dort garantiert Unterstützung finden.

Fazit

Mono ist eine Alternative. Sicher nicht in jedem Fall, aber sinnvoll eingesetzt ist das freie .NET Framework eine fantastische Möglichkeit, Aufgaben kostengünstig und stabil zu lösen. Es ist eine Frage der friedlichen Koexistenz zweier Welten die hervorragend zusammenpassen und harmonieren.

In jedem Fall lohnt es sich aber, die Mono-Entwicklung zu verfolgen und ei-

gene .NET-Projekte regelmäßig mit MoMA zumindest auf Mono-Tauglichkeit abzuklopfen.

Aus dem Umfeld der 1.1-Runtime lässt sich ruhigen Gewissens sagen, das alles funktioniert. ASP.NET, Webservices und GUI-Anwendungen sind kein Problem. Das gilt in zunehmenden Maße auch für die 2.0-Runtime. Bis auf Webservices lassen sich bereits ASP.NET- und GUI-Anwendungen unter Mono ausführen. Eine intensive Nutzung des MoMA empfiehlt sich jedoch. Bestimmte Techniken wie zum Beispiel die Verwendung von Generics wurden nicht berücksichtigt, da es

den Rahmen dieser Artikelserie einfach sprengen würde. Es sei aber darauf hingewiesen, das Generics implementiert sind.

Die dotnetpro wird Sie auch weiterhin auf dem Laufenden halten, was es Neues zu diesem Thema gibt. |||||

- [1] Jan Waiz, .NET in Stereo, .NET für Linux, dotnetpro 1/2007, Seite 62 ff.
- [2] Jedes .NET mit jedem Client, .NET für Linux, Teil 2, dotnetpro 2/2007, Seite 60 ff
- [3] Mono-Downloads, www.mono-project.com/Downloads
- [4] PostgreSQL Downloads, www.postgresql.org/download/