Affen und Gnus

Implementierungen von Microsofts .NET-Framework unter GNU/Linux

„.NET kommt von Microsoft - und Microsoft ist böse, also muß auch .NET zwangsläufig böse sein.“ *(Linux Magazin 07/2002)*

Dieser Aufsatz, er befaßt sich nicht mit einem Teilgebiet der Biologie, hat aber trotzdem etwas mit Affen und Gnus zu tun: Es geht um zwei Implementierungen von Microsofts .NET-Framework unter GNU/Linux. Ferner handelt dieser Text von einer Wüste, der Wüste der Programmiersprachen und um Versuche, darin einheitliche Schnittstellen zu schaffen - sozusagen Sandkörner zu sortieren.
In der Windows-Welt heißt der durchaus vielversprechende Ansatz hierfür - er wurde im Januar dieses Jahres endgültig freigegeben - „Microsoft .NET“. Und weil dieser Ansatz so vielversprechend ist und brauchbare Schnittstellen unter den Unmengen verwendeter Programmiersprachen unter dem freien Betriebssystem GNU/Linux bzw. zwischen GNU/Linux und Windows längst überfällig sind, versuchen auch hier Entwickler das von Microsoft konzipierte .NET-Framework zu implementieren. Die bereits am weitesten fortgeschrittenen Projekte hierzu sind DotGNU[1] von der GNU Free Software Foundation und Mono[2] von Ximian.

1. .NET

Bereits gegen Ende 1999, als Microsoft die ersten Publikationen zum Thema .NET herausgab, gab es von verschiedenen Seiten arge Proteste, viele vermutete hinter dem merkwürdigen Kürzel eine neue hinterhältige Geldschneiderei. Etwas später veröffentliche Microsoft dann seine Ideen zu „Passport.NET“, einem Dienst zur Überprüfung der Identität - mit dem Ergebnis, daß die Gegenstimmen noch lauter wurden. Als dann im Januar 2002 das .NET-Framework endgültig freigegeben und die ersten .NET-Produkte verkauft wurden, hieß es von verschiedenen Seiten, .NET wäre nichts weiter als eine typisch microsoftige Methode, alle Produkte erneut verkaufen zu können und dabei lediglich an Titel und Dokumentation ein „.NET“-Suffix anzuhängen.
Andererseits ist natürlich zu bemerken, daß Microsoft die CLR, den Kern von .NET sowie einen C#-Compiler kostenlos zur Nutzung zur Verfügung stellt und die Spezifikationen der CLI (*Common Language Infrastructure*) und der Sprace C# bei der ECMA eingereicht wurden und dort (ECMA334 und ECMA-335)[5] für Jederman verfügbar sind. Ebenfalls im Januar dieses Jahres wurde denn auch innerhalb der GNU/Linux-Gemeinde intensiv über .NET diskutiert. Im Ergebnis dessen begannen zwei Projekte, das .NET-Framework zu implementieren - auch das nicht ohne kritische Fragen und mancherlei verhaltenen Protest.
Was ist nun also .NET? Was macht es so attraktiv? Warum wird darüber so kontrovers diskutiert?
„Die wesentliche Idee hinter Microsoft .NET ist im Grunde, den nächsten Entwicklungsschritt für ds Internet vorzubereiten. Der Schwerpunkt wird nicht mehr auf eine Welt gelegt, in der einzelne Geräte und Websites einfach nur durch das Internet verbunden werden, sondern auf eine Welt, in der die Geräte, Dienste und Computer zusammenarbeiten, um dem Anwender die Arbeit zu erleichtern und ihm Lösungen zu bieten, die sonst nicht oder nur schwer möglich wären.“ [3]
Mittels dieser schwammigen Ausführungen versucht Tom Archer in seinem Buch „Inside C#“ zu erklären, was .NET ist. Leider hatte ich auch nach mehrmaligem durchlesen des Kapitels „Einführung in Microsoft .NET“ dieses vor einem Jahr in der Microsoft Press erschinenen Buches nichteinmal ansatzweise eine Vorstellung darueber, was der Hintergrund zur Entwicklung von Microsoft .NET gewesen sein könnte, was eine speziell für .NET konzipierte Sprache - C#, gesprochen „C-Sharp“ - für Vorzüge gegenüber bereits seit langem eingesetzten Sprachen wie C, C++ oder Java hat und was .NET dem Entwickler letztlich bringt.
Klar wird jedoch, daß Microsoft .NET eine Infrastruktur, ein Framework

zur Softwareentwicklung darstellt. Dieses Framework besteht aus der CLR (*Common Language Runtime*) und aus einigen Klassenbibliotheken, auch als BCL (*Base Class Library*) bezeichnet. Die CLR fungiert hierbei als eine virtuelle Maschine (VM), in/auf der ein Programm abläuft. Im Unterschied zu der aus Java bekannten VM stellt die CLR jedoch vielmehr eine Abstraktion des Betriebssystems auf hoher Ebene als einen vollständig Abgeschirmten für die Ausführung eines Programms dar. Die BCL, nicht wirklich eine einzige Bibliothek sondern eher eine Sammlung vieler, nach Aufgabenbereichen getrennter Bibliotheken, stellen dem Programmierer die gesammte Funktionalität der CLR in Form einer Programmierschnittstelle zur Verfügung.
Nach intensiver Lektüre einiger Aufsätze von Mitarbeitern der Free Software Foundation wurde mir dann auch klar, was der eigentliche Sinn des .NET Frameworks ist: Das hochgesteckte Ziel besteht darin, Anwendungen nicht mehr mit einer Schnittstelle für jede potentiell verwendete Sprache auszustatten, sondern einen Standad für solche Schnittstellen zu entwerfen und diesen mehr oder weniger sprachenunabhängig mittels einer Lauzeitumgebung umzusetzen. Selbstverständlich müssen hierfür auch Anpassungen in den einzelnen zu verwendenden Programmiersprachen vorgenommen werden. Microsoft verwendet in diesem Zusammenhang gern den Begriff der „.NET-fähigen Sprachen“.

1.1. Die Common Language Runtime

Das Herzstück von .NET stellt die bereits erwähnte Laufzeitumgebung CLR dar. In ihr können viele, in .NET-fähigen Sprachen entwickelte Anwendungen gleichzeitig laufen und dank der sprachübergreifenden Interoperabilität über einheitliche Schnittstellen mit einander kommunizieren.
Eine .NET-fähige Sprache zeichnet sich hierbei dadurch aus, daß es für sie einen .NET-konformen Compiler bzw. Interpreter gibt. Diese müssen bei der Codeerzeugung bestimmte Regeln, die Common Language Specifications (CLS) einhalten. Bereits dadurch, daß das Kompilat unter Berücksichtigung der CLS erzeugt wurde, ist es in der Lage, in der CLR zu laufen und mit anderen .NET-Anwendungen zusammenzuarbeiten.
Als ein bedeutendes Konzept der CLR benennen Microsoft-lastige Quellen ([3], [4]) auch den „verwalteten Code“ (*managed code*). Dahinter verbirgt sich jedoch lediglich der Code einer .NET-Anwendung, der unter der CLR läuft und in diesem Sinne von ihr verwaltet wird. Das Konzept scheint keine wesentlichen Neuerungen im Vergleich mit der von Java bekannten VM aufzuweisen.

1.2. Die Klassenbibliotheken (BSL)

Die Klassenbibliotheken stellen die Programmierschnittstelle für die CLR dar. Da sie die einzige Programmierschnittstelle der CLR sind, repräsentieren sie theoretisch die gesammte Funktionalität dieser - und dementsprechend viele gibt es auch. Selbst der Versuch einen groben überblick über sie zu geben, würde den Rahmen dieser Arbeit sprengen, ganz abgesehen davon, daß es keinen einsichtigen Grund gibt, an dieser Stelle Unmengen von Klassen aufzuzählen.
Ein .NET-konformer Compiler setzt letztlich lediglich die Anweisungen des Quelltextes in eine von der .NET-Laufzeitumgebung wiederum in Bibliotheksaufrufe umsetzbare Form um und bildet somit eine syntaktische Schnittstelle zu den Klassenbibliotheken.

1.3. .NET und C#

C# ist prinzipiell nur eine Programmiersprache von vielen. Was sie zu etwas besonderem macht ist, daß sie erst vor sehr kurzer Zeit entwickelt wurde - und zwar speziell für .NET.
C# wurde in Anlehnung an die Sprachen C, C++ und Java konzipiert, besonders der Enfluß von Java ist noch deutlich zu bemerken.

1.4. Die Platformunabhängigkeit von .NET

Sind .NET bzw. C# Plattformunabhängig? Eine durchaus interessante Frage, wenn man bedenkt, daß Microsoft in Verbindung mit .NET gerne von der „Vorbereitung der nächsten Entwicklungsschritte des Internet“ und von „.NET Gerätesoftware für neue Internetgeräte“ redet - das Internet ist schließlich alles andere als eine in sich geschlossene, homogene Netzwerkumgebung.
Kompiliert man nun also beispielsweise ein C#-Programm unter Windows mit dem von Microsoft herausgegebenen Compiler, so ist das Ergebnis eine Windows-Anwendung. Diese besteht aus einem Stück Wrapper-Code, das die CLR aufruft und dem eigentlichen, dann von der CLR auszuführenden Programmcode. Letzterer ist, ähnlich dem von Java-Compilern generierten Bytecode dahingehend platformunabhängig, als daß er auf jedem Rechner genutzt werden kann, für den es eins CLR gibt. Da der Wrapper am Anfang des Programmes jedoch nur unter Windows lauffähig ist, ist ein .NET-Programm prinzipiell nur wenig plattformabhängiger als ein jedes, für eine spezielle Systemumgebung erstellte Programm.

Zum derzeitigen Stand der Entwicklung muss man sogar sagen, daß C#-Programme noch wesentlich unportierbarer sind als beispielsweise C++ oder ANSI-C Anwendungen: sowohl der zum Übersetzen nötige Compiler als auch die Laufzeitumgebung wurden von Microsoft ausschließlich für Windows herausgegeben. Auch wenn die Standards offen sind und bereits zwei Projekte zur Implementation des Frameworks für GNU/Linux laufen, wird sich daran vorerst wohl nichts ändern. Beide Projekte sind bei weitem noch nicht abgeschlossen und implementieren .NET vorerst auch nur für Rechner mit Intel-kompatiblen CPUs bzw. im Falle von Mono bereits rudimentär für PowerPC. Hierzu jedoch näheres im Abschnitt 2.

2. DotGNU und Mono

So schön der Ansatz von .NET auch ist, das Framework ist bislang nur für Microsoft Betriebssysteme vollständig verfügbar. Da es sich aber um einen offenen Standard, besteht für die freie Entwicklergemeinschaft zumindest die Möglichkeit, ihn auch für zu Windows alternatve Betriebssysteme zu implementieren. Und genau das wird denn auch getan - DotGNU und Mono sind zwei Projekte, in denen seit Anfang dieses Jahres versucht wird, eine CLR, einen C#-Compiler und die BSL zu implementieren. Vorallem DotGNU geht hierbei noch einige Schritt weiter als Microsoft: „DotGNU is being developed jointly under the auspices of FreeDevelopers and the GNU project. The DotGNU system will be a well-integrated part of the GNU system and it will also be made available to the users of all other widely-used operation systems. [...] The DotGNU system will deliver the same and better benefits to end users and business customers, but we will be very careful that no company or other organization can gain any monopoly-like power over the system.“[1]

Soviel zu Allgemeinen Teil - wie lassen sich die beiden Entwicklungsumgebungen nun also installieren und was kann man damit machen?

2.1. Die Installation

2.1.1. Mono - Bereits die Installation wirkt abschreckend

Um genau das auszuprobieren besorgte ich mir vor einigen Tagen die aktuellen Quellen des Projektes - die kann man sich unter *http://www.go-mono.com* herunterladen und, begann ersteinmal die Hilfen zu lesen und wurde im Fall von Mono ersteinmal mit vielen zutiefst merkwürdigen Dingen konfrontiert: Mono, selbst komplett in C# geschrieben, ist zwar inzwischen selfhosting, da es aber keine Binaerpakete gibt, muß man zumindest für die Erstinstallation auf Microsoft-Produkte zurueckgreifen. Na gut, dachte ich mir, vertrieb kurzerhand einen Kollegen von seinem PC an meine Sun und blockierte seinen Rechner einen guten Tag lang - für jemanden, der doch eher selten mit Windows arbeitet gestaltet sich die Installation all dieser Windows-Tools alles andere als einfach:

Zuerst muß man sich eine Cygwin-Umgebung[6], das ist sowas wie UNIX für Windows - eine brauchbare Shell, ein gcc, binutils, etc. herunterladen und installieren. Leider sind all diese Dinge unter Windows nicht nur fast so kompfortabel wie unter UNIX, sie sind auch ebenso groß. Mit einem Downloadvolumen von etwa 150 MBytes ist man dabei.
Als nächstes benötigt man das .NET-Framework SDK[7], das gibt's bei Microsoft und ist ebenfalls ca. 150 MBytes lang - zumindest war die Installation nach etwa drei Mausklicks und 10minütigem Warten abgeschlossen.
Nachdem ich mir im Anschluß daran die *mcs*-Quellen, so heißt der C#-Compiler von Mono, gezogen hatte, konnten diese ziemlich unproblematisch unter der Cygwin-Umgebung entpackt und kompilliert werden - ein einfaches 'make' tat seine Pflicht.
Weiter ging's dann unter GNU/Linux: Hier mußte ich zuerst die *libgc*[10], eine Bibliothek die eine Garbage Collection implementiert, installieren. Danach konnte ich das Mono-Paket entpacken, konfigurieren und kompilieren:

```
gunzip -c mono-0.12.tar.gz | tar -xv
cd mono-0.12
./configure --prefix=/usr/local/mono
make
```

Die Angabe des Prefix, also des Installationsverzeichnis ist nicht zwingend notwendig, erleichtert aber eine spätere Deinstallation.
Bevor man nun das gesammte Paket installieren kann, müssen noch einige Dateien aus dem unter Windows erstelten Kompilat kopiert werden:

```
cp /tmp/mcs-0.12/mcs/mcs.exe /src/mono/mono-0.12/runtime
cp /tmp/mcs-0.12/class/lib/*.dll /src/mono/mono-0.12/runtime
```

Die Installation wird wie üblich mit einem *make install* durchgeführt, nachdem man dann seinen *PATH* und *LD_LIBRARY_PATH* entsprechend angepaßt hat, ist die Installation abgeschlossen:

```
make install
export LD_LIBRARY_PATH=LD_LIBRARY_PATH:/usr/local/mono/lib
export PATH=$PATH:/usr/local/mono/bin
```

Übrigens findet sich im aktuellen *Linux Magazin*[9] auf den Seiten 92 bis 96 ein hervorragender Artiken, der die Installation von Mono sehr genau wiedergibt.

2.1.2. DotGnu Portable.NET

Portable.NET[8] gliedert sich in mehrere Bereiche: Die drei Hauptkomponenten sind *treecc*, ein sog. „buildtool“ (was es tut und warum's das überhaubt gibt, kann man unter *http://www.southern-storm. com.au/treecc.html* nachlesen, *pnet* selbst - hier finden sich die Laufzeitumgebung, der C#-Compiler und einige Tools, sowie die *pnetlib*, die C#-Bibliothek con DotGNU Portable .NET.
Die Installation von Portable.NET gestaltet sich, verglichen mit der von Mono doch sehr einfach - die wichtigsten Komponenten der Alternative zur Alternative sind in C geschrieben und lassen sich dementsprechend mit GNU/Linux-Bordwerkzeugen innerhalb einer Viertelstunde kompilieren. Einzige Ausnahme hierbei ist die C#-Bibliother *pnetlib*, die ist wiederum selbst in C# geschrieben und laesst sich mit dem noch unfertigen *cscc*, so heißt der C#-Compiler von DotGNU, nicht übersetzen. Wenn man hier allerdings nicht zu Microsofts C#-Compiler wechseln möchte, kann man auch einfach das im Paket enthaltene Beispiel-Kompilat verwenden.
Abgesehen davon läuft die Kompilierung wie unter UNIX üblich für jedes Paket nacheinander mittels

```
gunzip -c <package-version.tar.gz | tar -xv
cd package-version
./configure --prefix=/usr/local/dotgnu
make
make install
```

ab. Die Angabe des Prefix, also des Installationsverzeichnis ist nicht zwingend notwendig, erleichtert aber eine spätere Deinstallation.
Ebenso wie bei Mono muß man auch nach der Installation von Portable.NET noch seine Umgebungsvariablen etwas anpassen:

```
export LD_LIBRARY_PATH=LD_LIBRARY_PATH:/usr/local/dotgnu/lib
export PATH=$PATH:/usr/local/dotgnu/bin
```

2.2. Stand der Entwicklung

Nachdem man dann die Installation der beiden Pakete erst einmal hinter sich gebracht hat, kann mit beiden gearbeitet werden. Von ihrem derzeitigen Funktionsumfang unterscheiden sie sich nur minimal: Beide bieten dem Nutzer eine Laufzeitumgebung und einen C#-Compiler an. Der C#-Compiler erzeugt in beiden Fällen *.exe*-Dateien, die unter WIndows als auch unter GNU/Linux, hier aber nur, indem man sie von der CLR starten läßt, lauffähig sind.
Einen kleinen Unterschied gibt es im Fortschritt der Implementierung der Klassenbibliotheken. Ximian gibt hier derzeit eine Fertigstellung von erwa 80% an, die Free Software Foundation ist bereits bei etwa 90%.
Als kleinen Ausgleich stellt Ximian dafür bereits einen Just-In-Time-Compiler für C# bereit.
Die gewünschte Sprachenunabhängigkeit ist allerdings noch nicht erreicht, gibt es doch derzeit noch keine .NET-fähigen Compiler für all die anderen Programmiersprachen, die es da so gibt. Allerdings trifft dies auch auf .NET unter Windows zu.
Was es unter GNU/Linux schon gibt, ist die *GTK#*-Bibliothek[11], eine Schnittstelle von .NET zu den Graphikbibliotheken von *Gnome*[11]. Überhaupt wird vorallem innerhalb der Entwicklergemeinde von Gnome derzeit intensiv über die Einsatzmöglichkeiten der zwei freien .NET-Implementierungen diskutiert.

2.3. Einsatzmöglichkeiten/Ausblick

Grundsätzlich gehe ich derzeit davon aus, daß eine der Beiden, vielleicht auch das Produkt einer Fusion aus DotGNU Portable.NET und Mono, zu einem festen Bestandteil von GNU/Linux werden wird. Da große Teile des GNU-Systems auch unter anderen unixoiden Betriebssystemen eingesetzt werden - beispielsweise wird Sun Microsystems[12] die nächsten Solaris-Versionen nicht mehr mit CDE als Desktopumgebung, sondern mit Gnome ausliefern - ist eine Ausbreitung des .NET-Standards in der UNIX-Welt durchaus wahrscheinlich.
Die DotGNU-Website sagt hierzu: „The DotGNU system will be a well-integrated part of the GNU system and it will also be made available to the users of all other widely-used operation systems.“
Der Funktionsumfang, den das .NET-Framework derzeit bietet, ist gewaltig. Und er wird ständig durch noch mächtigere Bibliotheken ergänzt, beispiels-

weise gab Microsoft vor einem Monat eine Erweiterung der .NET-Entwicklungstools mit dem Namen *.NET Speech* heraus, wie der Name schon vermuten läßt, erweitern diese die Funktionalität um die Unterstützung von Sprachbefehlen. Sollte es in naher Zukunft sowohl für Windows als auch unter UNIX einheitliche CLRs und .NET-Bibliotheken geben, so wäre dies durchaus ein gewaltiger Schritt in der Entwicklung der EDV: Entwicklerteams könnten Anwendungen in beliebigen Sprachen und auf beliebigen Systemen entwickeln, das Kompilat wäre problemlos in anderen Umgebungen lauffähig.
Allerdings ist dies ein Ansatz, den auch Java verfolgt - auch hier wird ein platformunabhängiger Bytecode erzeugt und in einer Virtual Machine ausgeführt. Warum wird nun aber Java nicht für alles, was immer es auch sein mag, genutzt? Auch Java hat gewisse Grenzen, einige davon sind Sicherheits-, andere Architekturbedingt. Und was hat .NET damit zu tun? Ganz einfach: .NET verfolgt den gleichen Ansatz wie Java, die Erzeugung eines plattformunabhängigen Codes. Damit wird .NET auch mit den gleichen Problemen wie Java zu kämpfen haben, zumindest, wenn es sich, wie von Microsoft so häufig propagiert, zu einem Standard für das Internet oder ganz allgemein für die Nutzung in heterogenen Netzwerkumgebungen entwickeln soll.

Quellen

[1] http://www.dotgnu.org
[2] http://www.go-mono.com
[3] „Inside C#“, Tom Archer, Microsoft Press, ISBN 3-86063-635-9
[4] http://www.microsoft.com/net
[5] http://www.ecma.ch
[6] http://www.cygwin.com
[7] http://www.microsoft.com/germany/ms/entwicklerprodukte/downloads
[8] http://www.southern-storm.com.au/portable_net.html
[9] LINUX Magazin 07/2002, http://www.linux-magazin.de
[10] http://www.hpl.hp.com/personal/Hans_Bohem/gc/
[11] http://www.gnome.org
[12] http://www.sun.com