



Common Language Infrastructure

Plattformübergreifende Programmierung mit .NET

Tim Roes

Betreuer: Prof. Dr. Holger Vogelsang
Wintersemester 2010/2011

Inhaltsverzeichnis

| | |
|--------------------------------------------------------------|-----------|
| 1 Grundlagen | 1 |
| 1.1 Geschichte von .NET | 1 |
| 1.2 Aufbau von .NET | 1 |
| 1.2.1 Intermediate Language | 2 |
| 1.2.2 Common Language Runtime | 3 |
| 1.3 Common Language Infrastructure | 3 |
| 1.4 Mono | 4 |
| 1.5 Patente | 5 |
| 2 Alternative Technologien | 6 |
| 2.1 C++, C# und Java im Vergleich | 6 |
| 2.2 Geschwindigkeitsvergleich | 7 |
| 2.2.1 Testprogramme | 8 |
| 2.2.2 Referenzrechner | 9 |
| 2.2.3 Versionen der Laufzeitumgebungen/Compiler | 9 |
| 2.2.4 Testergebnisse | 9 |
| 2.2.5 Analyse der Ergebnisse | 10 |
| 3 Implementierungsdetails | 11 |
| 3.1 Dateisystemfunktionen | 11 |
| 3.2 Speicherorte | 12 |
| 3.3 Grafische Oberflächen | 14 |
| 3.3.1 Native Oberflächen | 14 |
| 3.3.2 Plattformübergreifende grafische Oberflächen | 15 |
| 3.3.3 Plattformspezifische grafische Oberflächen | 17 |
| 4 Fazit und Ausblicke | 19 |
| A Special Folders | I |
| B Abbildungen | II |
| C Quellcode zu 2.2 | VI |

Vorwort

“I think there’s a world market for about five computers.”

Diese bekannte Aussage wird dem Mitbegründer der Firma IBM, Thomas J. Watson, zugeschrieben. Dass diese Einschätzung aus den vierziger Jahren sich nicht erfüllt hat, kann man heutzutage in fast jedem Haushalt sehen. Die Computerindustrie floriert und moderne Smartphones verfügen über mehr Rechenleistung als damalige High-End-Rechner. Neben der Leistung haben sich aber auch die Anwendungsgebiete stark verändert. Nicht nur auf normalen PCs, sondern auch auf fast allen Handys und Spielkonsolen laufen in der Zwischenzeit immer ausgereifere Betriebssysteme. Auf Grund dieser Aspekte haben sich auch die Bedürfnisse in der Entwicklung von Anwendungen verlagert und das Thema plattformübergreifende Programmierung ist wichtiger denn je. Dabei stehen Informatiker bei der Planung einer plattformübergreifenden Anwendung vor der Wahl einer geeigneten Programmiersprache beziehungsweise -technologie. In dieser Arbeit wird auf eine dieser Technologien, nämlich das von Microsoft entwickelte .NET-Framework zusammen mit der Sprache C# eingegangen und die Vor- sowie Nachteile erläutert.

Kapitel 1

Grundlagen

1.1 Geschichte von .NET

Das .NET-Framework ist eine Programmier-Technologie entwickelt von Microsoft. Sie entstand aus dem Versuch heraus eine einheitliche Laufzeitumgebung für das Component Object Model (COM)¹ zu entwickeln. Bei COM Version 3 stellte Microsoft fest, dass die Unterschiede zur vorhergehenden Version sehr groß waren, und es wurde daraufhin beschlossen der Technologie einen neuen Namen zu geben. Sie wurde zunächst in Next Generation Windows Service (NGWS) umbenannt. Im Juli 2000 nannte man sie erneut in .NET-Framework um, wovon schließlich am 5.1.2002 Version 1.0 freigegeben wurde. Das .NET-Framework wird bis heute aktiv entwickelt und liegt seit Frühjahr 2010 in der Version 4.0 vor. Eine genauere Übersicht über die einzelnen Versionen und deren Neuerungen ist in [31] zu finden.

1.2 Aufbau von .NET

Anwendungen für das .NET-Framework unterscheiden sich von Anwendungen, die zum Beispiel mit C++ oder C geschrieben wurden. Kompiliert man mit einem C++-Compiler eine Anwendung, so erzeugt der Compiler eine ausführbare Datei, die auf einem Windows System typischerweise die Endung `.exe` hat. Diese Datei besteht aus Maschinsprache für die zur Kompilierzeit angegebene Zielplattform. So ist eine Anwendung, die für die x86 Prozessorarchitektur unter Windows kompiliert wurde, nativ auch nur auf einem x86 Prozessor unter Windows ausführbar. Zum Portieren der Anwendung auf eine andere Zielplattform muss der Quelltext für diese Zielplattform erneut kompiliert werden. Erschwerend kommt in den meisten Fällen hinzu, dass Bibliotheken benutzt werden, die nicht betriebssystemübergreifend vorhanden sind, und

¹COM ist eine von Microsoft entwickelte Technik zur Interprozesskommunikation unter Windows mit sprach- und plattformunabhängigen Objekten.

somit zur Portierung der Anwendung auf eine andere Zielplattform das einfache Neukompilieren mit geänderten Parametern oftmals nicht ausreicht, sondern zunächst noch Änderungen am Quellcode gemacht werden müssen.

Durch das .NET-Framework sollen diese Umstände bei der Portierung einer Anwendung auf eine andere Zielplattform entfallen. Deshalb wandelt der Compiler den Quelltext nicht in Maschinensprache um, sondern in einen Zwischencode, der erst auf dem jeweiligen System durch das dort installierte .NET-Framework in Maschinensprache passend zur Prozessorarchitektur umgewandelt wird. Das Funktionsprinzip ähnelt dem von Java-Anwendungen. In den folgenden Abschnitten werden die einzelnen Bestandteile des .NET-Frameworks erläutert. Für eine detaillierte Beschreibung der Funktionsweise des .NET-Frameworks sei hier auf [28] und [31] verwiesen.

1.2.1 Intermediate Language

Die Intermediate Language (auch IL-Code genannt) ist die Zwischensprache, in die ein .NET kompatibler Compiler Quelltext umwandelt. Microsoft bietet Compiler für die Sprachen Visual Basic, C++/CLI², C#, JScript, J# und einen Assembler für IL-Code. Außerdem gibt es Projekte, die .NET-Compiler unter anderem für die Sprachen Ada, Caml, COBOL, Eiffel, Forth, Fortran, Haskell, LISP, Lua, Mercury, ML, Mondrian, Oberon, Pascal, Perl, PHP, Prolog, Python, RPG, Scheme und Smalltalk entwickeln.[30] Für IL-Code ist es unerheblich, in welcher Programmiersprache der ursprüngliche Quelltext geschrieben wurde. Je nach verwendeter Sprache wird nur ein Teil der Intermediate Language als Ausgabe erzeugt, so wie eventuell nicht jeder C++-Compiler jeden prozessorspezifischen Maschinenbefehl für die zu kompilierende Zielplattform nutzen wird. Im Folgenden wird C# stellvertretend für alle zu IL-Code kompilierbaren Sprachen verwendet werden. Wenn also im Weiteren von .NET-Anwendungen oder C#-Anwendungen die Rede ist, sind damit jegliche Anwendungen gemeint, die in IL-Code kompiliert werden und somit durch .NET oder zum Beispiel Mono (siehe 1.4 *Mono*) ausführbar sind.

Die Ausgabe, die ein .NET-Compiler erzeugt, nennt man ein verwaltetes Modul. Ein verwaltetes Modul beinhaltet neben dem eigentlichen IL-Code Metadaten über die in diesem Modul definierten Typen und über die aus diesem Modul heraus verwendeten Typen. Verschiedene verwaltete Module (auch verschiedener Programmiersprachen) einer Anwendung werden zusammen mit Ressourcendateien zu einer so genannten Assembly zusammengepackt. Eine Assembly ist entweder eine ausführbare Datei (exe-Datei) oder eine Bibliothek (dll-Datei), die von anderen Anwendungen genutzt werden kann.

²C++/CLI bietet die Möglichkeit gewöhnlichen C++ Quelltext in IL-Code zu kompilieren.

1.2.2 Common Language Runtime

Die Common Language Runtime (kurz CLR) ist dafür verantwortlich den IL-Code zur Laufzeit in Maschinensprache umzuwandeln. Man spricht hier von “just in time (kurz JIT) compiling”. Wird eine ausführbare .NET Anwendung gestartet, initiiert das Betriebssystem einen neuen Thread, der die zum Starten benötigten .NET-Bibliotheken lädt und dann zum Einstiegspunkt, also der main-Methode, der eigentlichen Anwendung springt. Wird eine Methode in der Anwendung zum ersten Mal aufgerufen, springt das Programm zum JIT-Compiler. Dieser liest den IL-Code dieser Methode ein und kompiliert ihn. Der kompilierte Code wird anschließend abgespeichert und ausgeführt. Ruft das Programm diese Methode nochmals auf, wird auf den bereits gespeicherten Maschinencode zugegriffen und dieser einfach ausgeführt. Durch das Kompilieren zu Maschinencode während der Laufzeit tritt bei der erstmaligen Ausführung einer Methode eine Zeitverzögerung im Vergleich zum Ausführen von nativen Maschinencode auf. Sollten schnelle Laufzeiten gewünscht sein, bietet .NET die Möglichkeit von Optimierungen des IL-Codes oder Optimierungen durch den JIT-Compiler. Microsoft bietet außerdem ein Tool, um die Assembly nicht erst während der Ausführung in Maschinensprache umzuwandeln, sondern dies bereits bei der Installation erledigen zu lassen.[13]

Die Aufgabe der Common Language Runtime ist es somit von der zu Grunde liegenden Hardware zu abstrahieren und die spezifischen Eigenschaften der benutzten Prozessorarchitektur zu ergen. So sind zum Beispiel die Größen der einzelnen primitiven Datentypen klar definiert und diese können nicht, wie es bei C++ der Fall ist, zwischen verschiedenen Prozessorarchitekturen variieren. Im Hinblick auf plattformübergreifende Programmierung soll die Common Language Runtime nicht nur von der zu Grunde liegenden Hardware abstrahieren, sondern auch von dem verwendeten Betriebssystem und dessen speziellen Eigenschaften.

1.3 Common Language Infrastructure

Teile des .NET-Frameworks wurden von der ECMA³ 2001 erstmalig unter dem Namen Common Language Infrastructure (CLI) als ECMA-335[2] standardisiert. Ein Jahr später wurde der Standard auch als ISO 23271[5] von der International Organization for Standardization übernommen. Seit Juni 2006 liegt der CLI-Standard bei ECMA in der vierten Version vor und seit dem 27.09.2006 bei der ISO als ISO 23271:2006.

Neben der eigentlichen Framework Technologie wurde außerdem die Sprache C# von ECMA als ECMA-334[1] und durch die ISO als ISO 23270[4] standardisiert. Auch hier

³Eine Normungsorganisation für Informations- und Kommunikationssysteme sowie Unterhaltungstechnik (<http://www.ecma-international.org>).

gab es die letzte Änderung 2006 zur vierten Version des ECMA-334 Standards und dem ISO 23270:2006 Standard.

Spezielle Teile des .NET-Frameworks wurden jedoch nicht standardisiert. Dazu gehören unter anderem die Benutzeroberflächen-Bibliotheken Windows Forms und Windows Presentation Forms (WPF) oder die Datenbankschnittstelle ADO.NET. Für die weitere Betrachtung ist es also wichtig zu erwähnen, dass das .NET-Framework auch Technologien implementiert hat, die über den Standard hinaus gehen.

1.4 Mono

Das .NET-Framework ist sicherlich die bekannteste Implementierung des CLI-Standards, jedoch läuft es nur unter dem Betriebssystem Windows. Neben dem .NET-Framework gibt es allerdings noch eine Reihe weiterer Implementierungen. Die vermutlich am weitesten verbreitete ist das Mono Projekt, welches im Internet unter <http://www.mono-project.com> zu finden ist. Die Entwicklung begann 2001 durch Miguel de Icaza von der Firma Ximio, die 2003 durch Novell aufgekauft wurde.[27] Am 30. Juni 2004 erschien Mono 1.0.[20] Seit dem 19. November 2010 liegt Mono in der Version 2.8.1 vor.

Neben der Implementierung des CLI-Standards umfasst das Mono Projekt einen C#-Compiler, der nach Angaben der Entwickler seit Mono Version 2.8 C# 4.0 komplett unterstützt.[15] Wie in *1.3 Common Language Infrastructure* erwähnt, ist die Implementierung des .NET-Frameworks weitreichender als der CLI-Standard. Auch Mono hat einige dieser nicht standardisierten Bibliotheken von .NET implementiert, wie zum Beispiel WinForms oder ADO.NET.[17] Um die Kompatibilität zu bereits entwickelten .NET-Anwendungen zu prüfen, bietet Mono eine automatisch erstellte Vollständigkeitsliste aller Klasse und deren Methoden[16] sowie den Mono Migration Analyzer[21], ein Tool, das die Kompatibilität einer .NET-Anwendung unter Mono prüft.

Besonders interessant ist Mono in Hinblick auf plattformübergreifende Entwicklung, da das Projekt auf diversen Plattformen lauffähig ist. Unter anderem läuft Mono unter Windows, Linux, Mac OS X oder BSD, aber auch Systeme, wie das iPhone, die Nintendo Wii oder die Playstation 3 werden unterstützt.[23] Da die Betriebssysteme teilweise unterschiedliche Techniken verwenden (zum Beispiel unterschiedliche Dateisysteme), hat Mono neben den oben erwähnten auch eigene Bibliotheken implementiert. Dazu gehören unter anderem POSIX-Bibliotheken, Schnittstellen zu GNOME, eine OpenGL-Schnittstelle und eine erweiterte Datenbankbindung.

Auch wenn in vielen Firmen nach wie vor Microsofts .NET eingesetzt wird und Anwendungen nicht plattformübergreifend gebaut werden[29], findet Mono langsam in

der Praxis Einzug. So wird Mono beispielsweise in der bekannten Anwendung Second Life⁴ verwendet, um Skripte für Objekte innerhalb der virtuellen Welt schreiben zu können.[7] Ebenso nutzt die Spieleentwicklungsumgebung Unity3D (<http://unity3d.com>) Mono zum Skripten der Spiellogik.[36] Unter [22] findet sich eine Liste weiterer Anwendungen, die Mono verwenden.

Aus all diesen Gründen wird hier Mono als plattformübergreifende Implementierung des CLI-Standards betrachtet werden. In Kapitel 3 *Implementierungsdetails* werden die Unterschiede bei der Programmierung von .NET-Anwendungen unter dem original .NET-Framework und Mono etwas genauer betrachtet.

1.5 Patente

Kritikpunkte am .NET-Framework und dessen freien Implementierungen sind oftmals die Patente, die Microsoft an dieser Technologie hält. Kritiker sehen hier die Gefahr, dass Microsoft andere Implementierungen des CLI-Standards mit Patentklagen vom Markt verdrängen will. Im Zusammenhang mit der Standardisierung durch die ECMA räumt Microsoft jedoch jedem das Recht ein, alle Patente zu verwenden, die für die Implementierung des Standards erforderlich sind[8][9] und hat die Standards unter die Microsoft Community Promise gestellt[12]. Dieses Zugeständnis gilt sowohl für den C#-Standard als auch den CLI-Standard. Bibliotheken wie ADO.NET und Winforms, die nicht standardisiert wurden, sind auch nicht durch die Microsoft Community Promise geschützt, jedoch hat Novell eine Patentvereinbarung mit Microsoft getroffen, die Mono gänzlich vor Patentklagen schützen soll.[26] Das detaillierte Lizenzierungsschema von Mono und weitergehende Patentfragen findet man unter [18].

Ob Softwarepatente formal innerhalb der Europäischen Union überhaupt zulässig sind, ist eine andauernde rechtliche Diskussion, die an dieser Stelle nicht behandelt werden soll.

⁴Second Life ist eine Anwendung zur Darstellung einer virtuellen Online Welt (<http://secondlife.com>).

Kapitel 2

Alternative Technologien

Neben dem .NET-Framework beziehungsweise Mono existiert eine große Reihe weiterer Alternativen zur Umsetzung von plattformübergreifenden Anwendungen. Großer Beliebtheit erfreuen sich etwa auch Sprachen wie JAVA und C++, die sich beide in der Praxis bewährt haben.[35] Jede der Sprachen und Techniken hat ihre Vor- und Nachteile. Im folgenden werden JAVA, C++ und C# im Hinblick auf Verwendung für plattformübergreifende Anwendungen betrachtet.

2.1 C++, C# und Java im Vergleich

Die Technologien und Sprachen C++, C# und Java wurden von Beginn an für unterschiedliche Anwendungsgebiete entwickelt und somit wurden auch unterschiedliche Schwerpunkte in der Entwicklung gesetzt. Als Bjarne Stroustrup 1979 die Entwicklung von C++ startete, wollte er hauptsächlich eine einfache Alternative zu C bieten, die einen möglichst hohen Grad an Rückwärtskompatibilität aufwies. An die Entwicklung einer plattformübergreifenden Sprache, dachte er damals nicht. Im Gegenteil: es war angedacht, dass C++ hauptsächlich auf UNIX-Systemen laufen sollte.[33] Betrachtet man die Entwicklungsziele von Java, wird deutlich, dass man hier versuchte, neben der weiteren Vereinfachung der Sprache im Vergleich zu C++, unter anderem einen Schwerpunkt auf Portabilität von Programmen zu legen. [34] Microsoft setzte bei der Entwicklung des .NET-Frameworks vor allem auf Sicherheit, Flexibilität und Geschwindigkeit, um eine moderne Entwicklungsplattform auf hohem Abstraktionsniveau zu schaffen.[32]

Da die Sprachen für unterschiedliche Zwecke entwickelt wurden, bieten sie auch unterschiedliche Voraussetzungen für plattformübergreifendes Programmieren. Wie in *1.2 Aufbau von .NET* erwähnt, wird C++ zu Maschinencode kompiliert, was für plattformübergreifende Anwendungen bedeutet, dass der Entwickler die Anwendung für jedes Betriebssystem und jede Prozessorarchitektur als eigene Version bereit stellen

muss. Da hier die Abhängigkeiten zu Bibliotheken groß sind, die nicht plattformunabhängig existieren, bedeutet dies meist Änderungen im Quelltext, was für kleinere Projekte oftmals nicht rentabel wäre. Java und C# bieten mit ihrem Zwischencode die Möglichkeit an, ein einmalig kompiliertes Programm ohne Änderungen auf beliebigen Betriebssystemen auszuführen, für die eine entsprechende Laufzeitumgebung existiert. Allerdings können in Java und C# auch plattformspezifische Bibliotheken verwendet werden. Die entsprechende Anwendung ist dann aber an eine bestimmte Plattform gebunden.

Sollte eine plattformübergreifende Anwendung neu entwickelt werden empfiehlt es sich, auf Programmiersprachen, die keinen Zwischencode besitzen, zu verzichten, um sich das Kompilieren für verschiedene Zielplattformen zu ersparen. Der oftmals genannte Kritikpunkt der Geschwindigkeit soll im nächsten Abschnitt etwas genauer betrachtet werden. Auf die Frage nach der richtige Wahl der Programmier Technologie kann hier keine pauschale Antwort gegeben werden, da hier noch eine Reihe weiterer Faktoren eine Rolle spielen. Dazu gehören unter anderem Aspekte wie die Verfügbarkeit benötigter Bibliotheken, Erfahrung der beschäftigten Entwickler oder Schnittstellen zu anderen Anwendungen, auf die in dieser Arbeit nicht weiter eingegangen werden soll.

2.2 Geschwindigkeitsvergleich

Ein großer Unterschied zwischen den Sprachen besteht, wie in *1.2.2 Common Language Runtime* bereits erwähnt, in ihrer Geschwindigkeit. C# Programme, wie auch Java Programme, werden erst durch einen JIT-Compiler kompiliert und haben daher einen Geschwindigkeitsnachteil im Vergleich zu direktem Maschinencode, wie ihn ein C++-Compiler produziert. Zum Vergleich der Geschwindigkeiten wurden sechs verschiedene Programme, die Anforderungen an unterschiedliche Aspekte einer Programmiersprache stellen, unter annäherungsweise gleichen Bedingungen ausgeführt. Jeder Test wurde in C++, Java und C# programmiert und auf den Referenzrechnern ausgeführt. Es diente ein Linux Rechner als Referenzrechner für die Tests von C++, Java und C# unter Mono. Für die Tests von C# unter .NET wurde ein Windows-System verwendet. Die C++-Programme wurden mit GCC einmal ohne weitere Parameter kompiliert und einmal mit angeschalteten Optimierungen, das heißt mit dem Parameter `-O2`. Jeder der Tests wurde 30 mal ausgeführt und als Ergebnis das arithmetische Mittel der einzelnen Laufzeiten genommen.

Anzumerken sei, dass dieser Geschwindigkeitsvergleich kein exaktes Maß für die Geschwindigkeit einer Sprache sein soll, da die Geschwindigkeit von verschiedenen Faktoren beeinflusst werden kann. Dazu gehören zum Beispiel die Wahl des Compilers oder auch die Qualität der Implementierung der einzelnen Standardbibliotheken. Letztere kann man im weiteren Sinne jedoch zur Geschwindigkeit einer Programmiersprache

hinzuzählen, da bei der Entwicklung von Anwendungen häufig auf diese Bibliotheken zurückgegriffen wird.

2.2.1 Testprogramme

Um ein möglichst aussagekräftiges Ergebnis zu erhalten, wurden bei den einzelnen Tests Schwerpunkte auf verschiedene Aspekte einer Programmertechnologie gelegt.

Test 1: Iterative Berechnungen Im ersten Test wird die Fähigkeit im Bereich der Arithmetik verglichen. Dazu dient eine iterative Implementierung zur Berechnung von PI. Da die arithmetischen Fähigkeiten und nicht die Speicherverwaltung verglichen werden sollen, wird das errechnete Ergebnis direkt verworfen. Als Abbruchbedingung werden 100.000.000 Schleifendurchläufe gewählt. Der Quellcode ist in den Listings C.1 bis C.3 zu finden.

Test 2: Rekursive Berechnungen Dieser Test besteht aus einer rekursiven Implementierung der Ackermannfunktion.⁵ Der Aufruf erfolgt mit den Werten $n = 4$ und $m = 1$. Der Quellcode ist in den Listings C.4 bis C.6 zu finden.

Test 3: Stringkonkatenation In dritten Test wird 500.000.000 mal der String “a” konkateniert. In C# und Java wurde die `StringBuilder`-Klasse an Stelle der `String`-Klasse verwendet, da die `String`-Klasse für unveränderbare Zeichenketten gedacht ist. Der Quellcode ist in den Listings C.7 bis C.9 zu finden.

Test 4: Verschachtelte Schleifen Im Kern von sechs verschachtelten Schleifen mit jeweils 42 Durchläufen wird eine Variable aus drei der Schleifenzähler berechnet. Der Quellcode ist in den Listings C.10 bis C.12 zu finden.

Test 5: Dateioperationen Zum Test der Dateisystemfunktionen werden 100.000 Dateien angelegt und mit jeweils 1000 Zufallsbytes aus `/dev/urandom` gefüllt. Anschließend werden alle Dateien wieder gelöscht. Da `/dev/urandom` unter Windows nicht vorhanden ist, werden die Dateien zum Test des .NET-Frameworks mit statischen Werten gefüllt. Unter Linux wurde das Dateisystem ext3 verwendet und unter Windows NTFS. Der Quellcode ist in den Listings C.13 bis C.15 zu finden.

⁵Die Ackermannfunktion ist eine extrem schnell wachsende mathematische Funktion. Sie erzeugt bereits bei geringen Eingabeparametern lange Laufzeiten und hohe Rekursionstiefen.

Test 6: Objektorientierung In diesem Test werden 100.000.000 Klassen angelegt. Jede angelegte Klasse ist eine von drei Subklassen einer Basisklasse. Welche Subklasse angelegt wird, entscheidet sich per Zufall. Anschließend wird auf der angelegten Klasse eine vererbte Methode aufgerufen und das Objekt in C++ anschließend wieder gelöscht. Der Quellcode ist in den Listings C.16 bis C.24 zu finden.

2.2.2 Referenzrechner

Die Tests wurden auf folgenden Systemen ausgeführt:

| | Linux (Mono, C++, Java) | Windows (.NET) |
|----------------|------------------------------|-----------------------------|
| Prozessor | Core i7-920 (4 mal 2,66 GHz) | Core i7-860 (4 mal 3,6 GHz) |
| RAM | 8 GB | 8 GB |
| Betriebssystem | 64 Bit Debian 5.0 | 64 Bit Windows 7 |

Tabelle 2.1: Referenzrechner der Geschwindigkeitsvergleiche

2.2.3 Versionen der Laufzeitumgebungen/Compiler

Zum Kompilieren und Ausführen wurden die zum Testzeitpunkt als “stable” gekennzeichneten Versionen der Compiler beziehungsweise Laufzeitumgebungen verwendet:

C++ kompiliert mit GCC 4.4.1

Java Sun Java SE 1.6.0_12

Mono 2.4.2.3

.NET 4.0.30319

2.2.4 Testergebnisse

Folgende Ergebnisse wurden als arithmetisches Mittel aus 30 Durchläufen berechnet. Bei keinem einzelnen Durchlauf gab es Abweichungen von mehr als 10% vom hier errechneten Mittelwert.

Der Test zur rekursiven Berechnung ließ sich weder unter .NET noch unter Java ausführen, da diese mit einer `StackOverflowException` unter .NET beziehungsweise einem `StackOverflowError` unter Java abstürzten. Auch das manuelle Erhöhen des maximalen Speichers für die Laufzeitumgebungen brachte in beiden Fällen kein Erfolg.

| [in Sekunden] | Mono | .NET | C++ | C++ (-02) | Java |
|--------------------------|---------|---------|---------|-----------|---------|
| Iterative Berechnungen | 14,1525 | 16,6332 | 8,0517 | 7,6282 | 21,0115 |
| Rekursive Berechnungen | 21,6973 | - | 21,0254 | 4,1575 | - |
| Stringkonkatenation | 9,2336 | 13,1518 | 10,5504 | 3,8226 | 15,4969 |
| Verschachtelte Schleifen | 4,2859 | 3,3377 | 12,2582 | 0,0014 | 6,1890 |
| Dateioperationen | 19,3679 | 36,6631 | 15,5929 | 15,5226 | 16,0942 |
| Objektorientierung | 4,3174 | 1,1175 | 5,2822 | 3,9270 | 0,7846 |

Tabelle 2.2: Ergebnisse des Geschwindigkeitsvergleichs

2.2.5 Analyse der Ergebnisse

Die Ergebnisse zeigen, dass C++ bei den meisten Anwendungen die besten Laufzeitergebnisse liefert. Jedoch zeigen die Werte auch, dass für Anwendungen, die keine strikten Zeitvorgaben an den Prozess haben, auch Sprachen mit einem Zwischencode, wie C# und Java, eine gute Alternative darstellen. Die Laufzeiten von C# unter Mono können im Allgemeinen mit denen unter .NET konkurrieren und beide liegen in den meisten Fällen unter denen der häufig verwendeten[35] Sprache Java.

Kapitel 3

Implementierungsdetails

Auch wenn sich die meisten Anforderungen problemlos plattformübergreifend realisieren lassen, unterscheiden sich die Betriebssysteme in ihrer Architektur an einigen Stellen fundamental voneinander. Auf einige dieser Unterschiede und die Probleme, die sich daraus ergeben, wird im Folgenden genauer eingegangen und mögliche Lösungsansätze werden erläutert.

Da C# die Möglichkeit bietet, das Betriebssystem zur Laufzeit zu ermitteln, können die Implementierungen der einzelnen Betriebssysteme getrennt werden, wodurch eine große Kompatibilität zu allen Betriebssystemen gewährleistet wird. Dazu dient die Enumeration `Environment.OSVersion.Platform` vom Typ `System.PlatformID`. Beispiele solcher Betriebssystemweichen sind in den nächsten Kapiteln zu finden.

Zwar ist Mono auch auf Plattformen wie der Nintendo Wii oder dem iPhone lauffähig, jedoch sollen diese in dieser Arbeit nicht näher betrachtet werden. Die folgenden Beispiele beschränken sich deshalb auf die klassischen Desktop-Betriebssysteme Linux, Windows und Mac.

3.1 Dateisystemfunktionen

Jedes Betriebssystem bringt seine eigenen Dateisysteme mit. Während Windows hauptsächlich auf NTFS setzt, verwendet Mac das Dateisystem HFS+. Linux Benutzer sind weitgehend frei in der Wahl des Dateisystems, jedoch wird hier als Standard für viele Distributionen das “extended file system (ext)”⁶ verwendet. Diese Dateisysteme unterscheiden sich in ihrer Schnittstelle vor allem durch Dateiattribute und Dateirechteverwaltung. NTFS bietet eine flexible Rechteverwaltung durch so genannte “Security Descriptors”. Diese ermöglichen es, Rechte an einer Datei für jeden Benutzer und jede

⁶Das “extended file system” liegt aktuell als ext4 vor. In diversen Distributionen (meist Systeme für den Serverbetrieb) wird noch ext3 verwendet.

Gruppe einzeln anzugeben. Ext bietet ohne Erweiterungen eine eher minimalistische Rechteverwaltung an. Es können nur Lese-, Schreib- und Ausführrechte für den Besitzer, die besitzende Gruppe und alle anderen Benutzer gesetzt werden. Auch bei den Dateiattributen unterscheiden sich die einzelnen Betriebssysteme. So kennen NTFS und HFS+ zum Beispiel versteckte Dateien, ext hingegen nicht.

Funktionen, die von den speziellen Aspekten des Dateisystems unabhängig sind, wie zum Beispiel Dateien öffnen, schließen, lesen, schreiben oder löschen, können problemlos auf allen Dateisystemen ausgeführt werden. Dabei kann allerdings die unterschiedliche Adressierungsart ein Problem sein. So existieren unter Windows Laufwerksbuchstaben, als Trennzeichen für Ordner wird der Backslash verwendet und es wird keine Unterscheidung zwischen Groß- und Kleinschreibung gemacht. Unter Linux und Mac wird der Slash als Trennzeichen zwischen Ordnern verwendet, es gibt nur einen Verzeichnisbaum ohne Laufwerksbuchstaben und alle Datei- und Ordnernamen unterscheiden zwischen Groß- und Kleinschreibung. Das Problem der Trennzeichen kann mit Hilfe von `Path.DirectorySeparatorChar` umgangen werden. In dieser Eigenschaft steht jeweils das betriebssystemspezifische Trennzeichen für Pfadangaben. Lösungsvorschläge zur Vermeidung von absoluten Pfadangaben werden im nächsten Kapitel etwas genauer betrachtet. Um Probleme mit der Groß- und Kleinschreibung bei der plattformübergreifenden Programmierung zu vermeiden, sollte auch bei NTFS darauf geachtet werden, eine konsistente Schreibweise zu verwenden.

Sollten Programme Dateirechte oder Dateiattribute setzen müssen, finden sich in den original .NET-Klassen nur die Möglichkeiten für die Bearbeitung von NTFS und FAT Dateiattributen und Rechten, da das .NET-Framework von Microsoft stammt. Das Mono Projekt stellt unter dem Namespace `Mono.Unix` Klassen zur Verfügung, um Dateiberechtigungen in Unix-Dateisystemen wie HFS+ und ext zu setzen. Ein Beispiel für ein Projekt, das dies nutzt, ist der plattformübergreifende Paketmanager Zero Install (<http://0install.de>). Der Schreibschutz einer Datei kann mit einer Betriebssystemweiche wie in Listing 3.1 gesetzt werden.

3.2 Speicherorte

Neben den eigentlichen Funktionen des Dateisystems ist auch die Hierarchie der einzelnen Ordner unterschiedlich. So findet man die “Eigenen Dateien” eines Benutzers unter Windows in `C:\Users\USERNAME\My Documents`, das entsprechende Pendant unter Linux, das “Home-Verzeichnis”, unter `/home/USERNAME` und `/Users/USERNAME` ist das entsprechende Verzeichnis bei Mac. Genauso verhält es sich mit den Benutzereinstellungen, Programmeinstellungen, temporären Dateien und vielem Anderen. Die Benutzung

Listing 3.1: Schreibschutz einer Datei setzen (abgeändertes Beispiel aus Zero Install)

```

1 // ...
2 switch(Environment.OSVersion.Platform) {
3     case PlatformID.Unix:
4     case PlatformID.MacOSX:
5         UnixFileSystemInfo fileSysInfo =
6         UnixFileSystemInfo.GetFileSystemEntry(/* file path */);
7         fileSysInfo.FileAccessPermissions &=
8             ~(FileAccessPermissions.UserWrite
9               | FileAccessPermissions.GroupWrite
10              | FileAccessPermissions.OtherWrite);
11         break;
12     case PlatformID.Win32Windows:
13     case PlatformID.Win32NT:
14         new FileInfo(/* file path */).IsReadOnly = true;
15         break;
16 }
17 // ...

```

von “hardcoded”⁷ Pfaden sollte deshalb vermieden werden. Dies kann bereits bei reinen Windows Programmen zu Probleme führen. So waren etwa die “Eigenen Dateien” vor Windows Vista unter `C:\Documents and Settings\USERNAME\My Documents` zu finden.

C# stellt Entwicklern die Methode `System.Environment.GetFolderPath(System.Environment.SpecialFolder)` zur Verfügung. Der erwartete Parameter vom Typ `System.Environment.SpecialFolder` gibt ein bestimmtes Verzeichnis (zum Beispiel “Eigene Dateien”, “Desktop”, “Anwendungseinstellungen” und Ähnliches) an, zu dem die Methode den Pfad auf dem aktuellen System als `String` zurück gibt.[10][11] In *Anhang A: Special Folders* findet sich eine Tabelle mit Rückgabewerten der Methode auf den unterschiedlichen Betriebssystemen.

Auf Grund der Unterschiede in den Architekturen gibt es nicht auf allen Betriebssystemen ein Pendant für jedes dieser Verzeichnisse. In solchen Fällen muss mit einer Betriebssystemweiche gearbeitet werden, um die vorgesehenen Pfade zu verwenden. Ein Beispiel dafür ist das saubere Anlegen von Anwendungseinstellungen. Unter Windows sollten die Benutzereinstellungen eines Programmes unter `C:\Users\USERNAME\AppData\Roaming\PROGRAM_NAME` liegen, auf einem Linux System sind die Benutzereinstellungen von Programmen unter `/home/USERNAME/.PROGRAM_NAME`⁸ zu finden. Speichert man die Einstellungen nun wie in Listing 3.2, sind die Dateien unter Windows richtig abgelegt, unter Linux jedoch liegen diese unter `/home/USERNAME/.config/MyApp/settings.ini`. Schöner wäre es, wenn sie unter `/home/USERNAME/.MyApp/`

⁷Hardcoded bezeichnet das Einfügen von Einstellungen und Ähnlichem (in diesem Falle Pfaden) direkt in den Quelltext, anstatt sie durch eine Konfiguration oder vorgefertigte Methoden bestimmen zu lassen.

⁸Ein Punkt am Beginn eines Datei- oder Ordernamens in Linux ist vergleichbar mit versteckten Ordnern oder Dateien unter Windows.

settins.ini zu finden wären. Dies lässt sich wieder mit einer Betriebssystemweiche realisieren (siehe Listing 3.3).

Listing 3.2: Speichern von Benutzereinstellungen

```
1 // ...
2 string settingsIni = Environment.GetFolderPath(Environment.
   SpecialFolder.ApplicationData)
3     + Path.DirectorySeparatorChar + "MyApp"
4     + Path.DirectorySeparatorChar + "settings.ini";
5 // ...
```

Listing 3.3: Speichern von Benutzereinstellungen mit Betriebssystemweiche

```
1 // ...
2 string settingsIni;
3 switch(Environment.OSVersion.Platform) {
4     case PlatformID.Win32Windows:
5     case PlatformID.Win32NT:
6         settingsIni = Environment.GetFolderPath(Environment.SpecialFolder.
           ApplicationData)
7             + Path.DirectorySeparatorChar + "MyApp"
8             + Path.DirectorySeparatorChar + "settings.ini";
9         break;
10    case PlatformID.Unix:
11        settingsIni = Environment.GetFolderPath(Environment.SpecialFolder.
           Personal)
12            + Path.DirectorySeparatorChar + ".MyApp"
13            + Path.DirectorySeparatorChar + "settings.ini";
14        break;
15    }
16 // ...
```

3.3 Grafische Oberflächen

Alle bisher betrachteten Aspekte sind eher aus Sicht der Programmierer zu beachten und bleiben für Benutzer der Anwendung weitgehend verborgen. Die Schnittstelle zwischen Benutzer und Anwendung, die grafische Oberfläche eines Programmes, interagiert direkt mit dem Benutzer und wird von ihm unvermeidlich wahrgenommen. Hier bestehen große Unterschiede zwischen den einzelnen Betriebssystemen. Diese reichen von dem unterschiedlichen Design der Oberflächen bis hin zu verschiedenen Anordnungen von Schaltflächen.

3.3.1 Native Oberflächen

Windows und Mac bringen jeweils eine native Oberfläche mit, die in das Betriebssystem integriert ist. Unter Mac nennt sich diese Oberfläche Aqua und lässt sich

durch das hauseigene Cocoa-Framework (<http://developer.apple.com/cocoa>) ansprechen. Seit Windows Vista wird Aero als grafische Benutzeroberfläche unter Windows verwendet. Zur Erstellung von GUIs dienen auf Ebene von .NET-Sprachen Windows Forms oder das neuere Windows Presentation Foundation (WPF). Bei Linux hängt die verwendete GUI-Bibliothek von der Wahl der Desktopumgebung ab. So verwenden Gnome und Xfce die Bibliothek GTK, wohingegen KDE auf der qt-Bibliothek aufbaut.

Es sei angemerkt, dass dieser Vergleich von nativen Oberflächen aus technischer Sicht nicht exakt ist, da die Betriebssysteme unterschiedliche Architekturen haben und die erwähnten Technologien teilweise auf unterschiedlichen Systemsschichten angesiedelt sind. Für die Betrachtung aus Sicht eines .NET-Entwicklers sind die Schnittstellen zu den Oberflächen das Wichtige. Für genauere Einblicke in die Systemarchitekturen sei deshalb auf die entsprechende Fachliteratur verwiesen. Im Folgenden werden Programme, die die native Oberfläche des Betriebssystems nutzen, als native Anwendungen bezeichnet.

3.3.2 Plattformübergreifende grafische Oberflächen

Entwickelt man eine plattformübergreifende Anwendung, kann man sich für eine Oberflächenbibliothek entscheiden, die auf allen gewünschten Betriebssystemen ein gutes Ergebnis liefert. Im Folgenden wird eine Auswahl an Oberflächenbibliotheken daraufhin untersucht. Für eine genauere Erläuterung der jeweiligen Bibliothek und ihrer Dokumentation sei auf die Entwicklerseiten oder entsprechende Fachliteratur verwiesen.

Windows Forms

Windows Forms ist momentan die am häufigsten verwendete Oberflächenbibliothek bei .NET-Entwicklern.[29] Obwohl Windows Forms kein Bestandteil des CLI-Standards ist, wurde es in Mono implementiert. Da nicht alle Windows-Steuerelemente auf allen anderen Plattformen verfügbar sind, werden diese in Mono durch das X-Server-System⁹ komplett neu gezeichnet. Dies führt dazu, dass unter Linux und Mac die Steuerelemente befremdlich wirken (siehe *Abbildung B.1*, *Abbildung B.2* und *Abbildung B.3*) und sich nur schlecht in das restliche Design einpassen. Vorteile bietet Windows Forms hauptsächlich unter Windows, da hier native Oberflächen erzeugt und keine zusätzlichen Bibliotheken benötigt werden.[14]

⁹Der X-Server ist eine Anwendung, die in vielen Unix-Derivaten als Grafiksystem fungiert. Sie ist für das Zeichnen von simplen Formen auf dem Bildschirm verantwortlich. Oberflächenbibliotheken wie GTK und qt benutzen unter Linux den X-Server zum Zeichnen ihrer Elemente.

Windows Presentation Foundation

Die Windows Presentation Foundation ist eine neuere Entwicklung von Microsoft. Da die Mono-Entwickler nicht planen, WPF zu implementieren, spielt es als plattformübergreifende Oberflächenbibliothek keine Rolle.[24]

GTK#

Aus dem Mono Projekt heraus entstand auch eine Schnittstelle für die Oberflächenbibliothek GTK+ (<http://www.gtk.org>), die für Windows, Linux und Mac entwickelt wird. Die C#-Schnittstelle trägt den Namen GTK# und läuft auf allen Betriebssystemen, auf denen auch GTK+ verfügbar ist. In Desktopumgebungen wie zum Beispiel Gnome, die auf GTK aufbauen, erzeugt man mit GTK# native Anwendungen. Unter Linux-Desktopumgebungen, die auf KDE aufbauen, Windows oder Mac erzeugt man dank anpassbarer GTK-Themen akzeptable Ergebnisse. Der Unterschied zu nativen Anwendungen ist jedoch noch wahrnehmbar (siehe *Abbildung B.4*, *Abbildung B.5* und *Abbildung B.6*).[19]

Qyoto (qt)

Für die Oberflächenbibliothek qt existiert eine C# Schnittstelle namens Qyoto. Wie auch GTK# benutzt sie die normale qt-Implementierung auf dem entsprechenden Betriebssystem. Für Linux-Desktopumgebungen wie KDE, die auf auf qt basieren, kann man mit Qyoto native Anwendungen erzeugen. Ähnlich wie bei GTK# kann qt mit Themen angepasst werden, um somit dem nativen Aussehen auf einer bestimmten Plattform näher zu kommen. Ein großer Nachteil von Qyoto sind die fehlenden Binaries für Windows und Mac. Ein Installationspaket wird nur für Linux zur Verfügung gestellt und das Kompilieren für andere Plattformen erweist sich als sehr schwierig. Die Benutzung als plattformübergreifende grafische Oberfläche ist daher nur eingeschränkt möglich.[6]

MonoMac

MonoMac ist eine Schnittstelle zum Cocoa-Framework, um Anwendungen im nativen Mac Design zu entwickeln und den Entwicklern eine Schnittstelle zu anderen Cocoa-Bibliotheken zu bieten. Diese Oberflächenbibliothek läuft nur unter Mac und ist von daher nicht als plattformübergreifende GUI-Bibliothek geeignet.[25]

wx.NET

wx.NET ist ein .NET-Wrapper für die wxWidgets Bibliothek (<http://wxWidgets.org>). Es handelt sich dabei um eine plattformübergreifende GUI-Bibliothek, die zum Zeichnen der Steuerelemente die nativen Steuerelemente des Betriebssystems nutzt. So hat eine wxWidgets Oberfläche unter Linux, Windows und Mac jeweils das native Aussehen. Der Nachteil ist, dass dadurch nur die gemeinsame Teilmenge aller Steuerelemente zur Verfügung gestellt werden kann.[37]

3.3.3 Plattformspezifische grafische Oberflächen

Wie oben ersichtlich wurde, hat jede Bibliothek gewisse Schwächen. Neben dem Aussehen der Steuerelemente existieren außerdem Unterschiede zwischen den Betriebssystemen, was zum Beispiel die Reihenfolge von Schaltflächen angeht (siehe auch *3.3 Grafische Oberflächen*). Die Reihenfolge der Schaltflächen *OK* und *Abbrechen* lässt sich noch recht leicht je nach Betriebssystem anpassen. Allerdings gibt es auch verschiedene Styleguides für die einzelnen Plattformen. So werden zum Beispiel verschiedene Steuerelemente je nach Betriebssystem anders verwendet. Um Oberflächen für die einzelnen Betriebssysteme zu erstellen, die sowohl das richtige Design haben als auch die Styleguides berücksichtigen, müssen diese getrennt voneinander erstellt werden.

An dieser Stelle sollte man mit einer Betriebssystemweiche arbeiten und die am nativsten aussehende GUI-Bibliothek für jedes System verwenden. Die Benutzeroberfläche für Windows könnte mit Windows Forms (*3.3.2 Windows Forms*) programmiert werden, die für Gnome unter Linux mit GTK# (*3.3.2 GTK#*) und die für Mac mit MonoMac (*3.3.2 MonoMac*). Eine beispielhafte Umsetzung mit dem Factory-Pattern¹⁰ könnte wie folgt aussehen. Dabei sind `WindowsGui`, `LinuxGui` und `MacGui` jeweils die speziellen Implementierungen des Interfaces `IGui` für die entsprechenden Betriebssysteme.

Um solch eine Anwendung zu kompilieren, müssen alle Oberflächenbibliotheken auf dem Entwicklersystem zur Verfügung stehen. Zur Laufzeit werden die entsprechenden Bibliotheken erst beim Aufruf der GUI benötigt, so dass die Anwendung ohne Probleme unter Windows laufen und dort die Windows Forms Oberfläche starten kann, ohne dass dafür MonoMac oder GTK# installiert sein muss.

Im obigen Beispiel wird keine Unterscheidung zwischen Desktopumgebungen, die auf qt basieren, und solchen, die auf GTK basieren, gemacht. Unter Linux ist es möglich, diverse Teile der einzelnen Desktopumgebungen zu vermischen, und so lässt sich technisch keine klare Trennung zwischen den einzelnen Desktopumgebungen vornehmen.

¹⁰Das Factory-Pattern ist eine Anlehnung an das Abstract-Factory-Pattern der *Gang of Four*[3]. Es existiert jedoch nur eine Factory-Klasse, die selbst entscheidet, welche Implementierung des Produktes, in diesem Fall der GUI, erstellt werden soll, und diese Auswahl nicht dem nutzenden Klienten überlässt.

Listing 3.4: GuiFactory zur Erzeugung plattformspezifischer GUIs

```
1 class GuiFactory {
2     public static IGui CreateGui() {
3
4         switch(Environment.OSVersion.Platform) {
5             case PlatformID.Win32Windows:
6             case PlatformID.Win32NT:
7                 return new WindowsGui();
8             case PlatformID.Unix:
9                 return new LinuxGui();
10            case PlatformID.MacOSX:
11                return new MacGui();
12            default:
13                throw new UnsupportedOperationException
14                    ("No Gui for this operating system present.");
15        }
16    }
17 }
18 }
```

Eine Methode, die vermeintlich bevorzugte Oberflächenbibliothek zu erkennen, wäre die Überprüfung, ob mehr Prozesse eine `gtk.so` oder eine `qt.so` Bibliothek geladen haben.

Kapitel 4

Fazit und Ausblicke

Im Verlauf dieser Arbeit wurden Grundlagen von .NET besprochen und mit alternativen Technologien zur plattformübergreifenden Programmierung verglichen. Einige mögliche Probleme bei der Implementierung wurden aufgezeigt sowie beispielhafte Lösungen skizziert.

Auch wenn plattformübergreifende Programmierung in vielen Firmen noch keinen Einzug erhalten hat, kann man doch sehen, dass C# beginnt sich in kommerziellen plattformübergreifenden Produkten zu etablieren. Je mehr Plattformen existieren und je mehr diese genutzt werden, umso größer wird auch der Bedarf nach plattformübergreifenden Techniken. So ist anzunehmen, dass solche Technologien in Zukunft weiter in den Vordergrund rücken werden. Bei der Programmierung für mehrere Plattformen müssen deren spezielle Eigenarten genau beachtet und gegebenenfalls verschiedene Implementierungen bereitgestellt werden. Wird von Anfang an große Sorgfalt darauf gelegt, eine Anwendung portierbar zu machen, lässt sich der Aufwand zur Portierung auf eine neue Plattform minimieren.

Mit .NET beziehungsweise dem Mono Projekt existiert eine mächtige Technologie zur Entwicklung plattformübergreifender Anwendungen, die eine gute Alternative zu Frameworks wie Java oder Programmiersprachen wie C++ bietet. Es liegt an den Entwicklern diesen Aufwand zu betreiben und auf solchen Technologien aufzubauen.

“There is no programming language—no matter how structured—that will prevent programmers from making bad programs.”

- Larry Flon

Literaturverzeichnis

- [1] ECMA INTERNATIONAL: *Standard ECMA-334 C# Language Specification*. – URL <http://www.ecma-international.org/publications/standards/Ecma-334.htm>. – Zugriffsdatum: 21.11.2010
- [2] ECMA INTERNATIONAL: *Standard ECMA-335 Common Language Infrastructure (CLI)*. – URL <http://www.ecma-international.org/publications/standards/Ecma-335.htm>. – Zugriffsdatum: 28.11.2010
- [3] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns*. Kap. Abstract Factory, S. 87ff, Addison-Wesley Longman, 1994
- [4] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 23270:2006*. – URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=36768. – Zugriffsdatum: 28.11.2010
- [5] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 23271:2006*. – URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=42927. – Zugriffsdatum: 28.11.2010
- [6] KDE: *Development/Languages/Pyto*. – URL <http://techbase.kde.org/Development/Languages/Pyto>. – Zugriffsdatum: 1.12.2010
- [7] LINDEN LAB: *Second Life Mono*. – URL <http://wiki.secondlife.com/wiki/Mono>. – Zugriffsdatum: 27.11.2010
- [8] MICROSOFT: *ECMA-334 4th Edition patent statements*. – URL <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma%20PATENT/ECMA-334%20&%20335/ECMA-335%204th%20Edition%20patent%20statements.pdf>. – Zugriffsdatum: 28.11.2010
- [9] MICROSOFT: *ECMA-334 4th Edition patent statements*. – URL <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma%20PATENT/ECMA-334%20&%20335/ECMA-334%204th%20Edition%20patent%20statements.pdf>. – Zugriffsdatum: 28.11.2010

- [10] MICROSOFT: *Environment.GetFolderPath-Methode*. – URL <http://msdn.microsoft.com/de-de/library/system.environment.getfolderpath>. – Zugriffsdatum: 8.12.2010
- [11] MICROSOFT: *Environment.SpecialFolder-Enumeration*. – URL [http://msdn.microsoft.com/de-de/library/system.environment.specialfolder\(v=VS.90\)](http://msdn.microsoft.com/de-de/library/system.environment.specialfolder(v=VS.90)). – Zugriffsdatum: 8.12.2010
- [12] MICROSOFT: *Microsoft Community Promise*. – URL <http://www.microsoft.com/interop/cp/default.aspx>. – Zugriffsdatum: 28.11.2010
- [13] MICROSOFT: *NGen.exe (Native Image Generator)*. – URL [http://msdn.microsoft.com/de-de/library/6t9t5wcf\(v=VS.100\).aspx](http://msdn.microsoft.com/de-de/library/6t9t5wcf(v=VS.100).aspx). – Zugriffsdatum: 19.11.2010
- [14] MICROSOFT: *Windows Forms*. – URL <http://msdn.microsoft.com/en-us/library/dd30h2yb.aspx>. – Zugriffsdatum: 1.12.2010
- [15] MONO PROJECT: *Mono C# Compiler*. – URL http://www.mono-project.com/CSharp_Compiler. – Zugriffsdatum: 21.11.2010
- [16] MONO PROJECT: *Mono Class Status Pages*. – URL <http://go-mono.com/status>. – Zugriffsdatum: 24.11.2010
- [17] MONO PROJECT: *Mono Compatibility*. – URL <http://mono-project.com/Compatibility>. – Zugriffsdatum: 24.11.2010
- [18] MONO PROJECT: *Mono FAQ:Licensing*. – URL <http://www.mono-project.com/Licensing>. – Zugriffsdatum: 28.11.2010
- [19] MONO PROJECT: *Mono GTK#*. – URL <http://www.mono-project.com/GtkSharp>. – Zugriffsdatum: 1.12.2010
- [20] MONO PROJECT: *Mono History*. – URL <http://www.mono-project.com/History>. – Zugriffsdatum: 21.11.2010
- [21] MONO PROJECT: *Mono Migration Analyzer*. – URL <http://mono-project.com/MoMA>. – Zugriffsdatum: 24.11.2010
- [22] MONO PROJECT: *Mono, Software*. – URL <http://mono-project.com/Software>. – Zugriffsdatum: 27.11.2010
- [23] MONO PROJECT: *Mono Supported Platforms*. – URL http://mono-project.com/Supported_Platforms. – Zugriffsdatum: 27.11.2010
- [24] MONO PROJECT: *Mono WPF*. – URL <http://www.mono-project.com/WPF>. – Zugriffsdatum: 30.11.2010

- [25] MONO PROJECT: *MonoMac*. – URL <http://www.mono-project.com/MonoMac>. – Zugriffsdatum: 1.12.2010
- [26] NOVELL INC.: *Joint letter to the Open Source Community*. – URL <http://www.novell.com/linux/microsoft/openletter.html>. – Zugriffsdatum: 28.11.2010
- [27] NOVELL INC.: *Ximian*. – URL <http://www.novell.com/linux/ximian.html>. – Zugriffsdatum: 21.11.2010
- [28] RICHTER, Jeffrey: *Microsoft .NET Framework-Programmierung in C#*. Microsoft Press, Mai 2006
- [29] ROES, Tim ; WEISSHAAR, Anna: *Survey on GUI Toolkits in .NET*. 2010. – URL http://www.timroes.de/survey_2010.pdf. – Zugriffsdatum: 13.12.2010
- [30] SCHWICHTENBERG, Holger: Marktübersicht: .Net-Programmiersprachen, Babylonische Vielfalt. In: *iX* (2007), Nr. 10/2007, S. 102–108
- [31] SCHWICHTENBERG, Holger: *Microsoft .NET 3.5 Crashkurs*. Microsoft Press, 2008
- [32] SCHWICHTENBERG, Holger: *Microsoft .NET 3.5 Crashkurs*. Kap. Ziele von .NET, S. 2f, Microsoft Press, 2008
- [33] STROUSTRUP, Bjarne: *The C++ Programming Language*. S. 7f, Addison-Wesley, Juni 1997
- [34] SUN MICROSYSTEMS: *The Java(tm) Language: An Overview*. – URL <http://java.sun.com/docs/overviews/java/java-overview-1.html>. – Zugriffsdatum: 11.11.2010
- [35] TIOBE SOFTWARE BV: *TIOBE Programming Community Index for November 2010*. – URL <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. – Zugriffsdatum: 11.11.2010
- [36] UNITY TECHNOLOGIES: *Mono Upgrade Details*. – URL <http://unity3d.com/support/documentation/Manual/MonoUpgradeDetails.html>. – Zugriffsdatum: 27.11.2010
- [37] WX.NET PROJEKT: *wx.NET*. – URL <http://wxnet.sourceforge.net>. – Zugriffsdatum: 1.12.2010

Anhang A

Special Folders

Die folgenden Ergebnisse werden von der Methode `Environment.GetFolderPath(Environment.SpecialFolder)` zurück gegeben. Es wurden nur die Eingabeparameter aufgelistet, die unter Linux oder Mac einen Rückgabewert haben. Alle anderen sind auf Grund der unterschiedlichen Architekturen nicht unter Linux und Mac verfügbar.

| Environment.SpecialFolder | Pfad |
|----------------------------------|---------------------------------------------|
| ApplicationData | C:\Users\USERNAME\AppData\Roaming (Windows) |
| | /home/USERNAME/.config (Linux) |
| | /Users/USERNAME/.config (Mac) |
| CommonApplicationData | C:\ProgramData (Windows) |
| | /usr/share (Linux) |
| | /usr/share (Mac) |
| Desktop, DesktopDirectory | C:\Users\USERNAME\Desktop (Windows) |
| | /home/USERNAME/Desktop (Linux) |
| | /Users/USERNAME/Desktop (Mac) |
| LocalApplicationData | C:\Users\USERNAME\AppData\Local (Windows) |
| | /home/USERNAME/.local/share (Linux) |
| | /Users/USERNAME/.local/share (Mac) |
| MyMusic | C:\Users\USERNAME\Music (Windows) |
| | /home/USERNAME/Music (Linux) |
| | /Users/USERNAME/Music (Mac) |
| MyPictures | C:\Users\USERNAME\Pictures (Windows) |
| | /home/USERNAME/Pictures (Linux) |
| | /Users/USERNAME/Picture (Mac) |
| Personal, MyDocuments | C:\Users\USERNAME\Documents (Windows) |
| | /home/USERNAME (Linux) |
| | /Users/USERNAME (Mac) |

Anhang B

Abbildungen

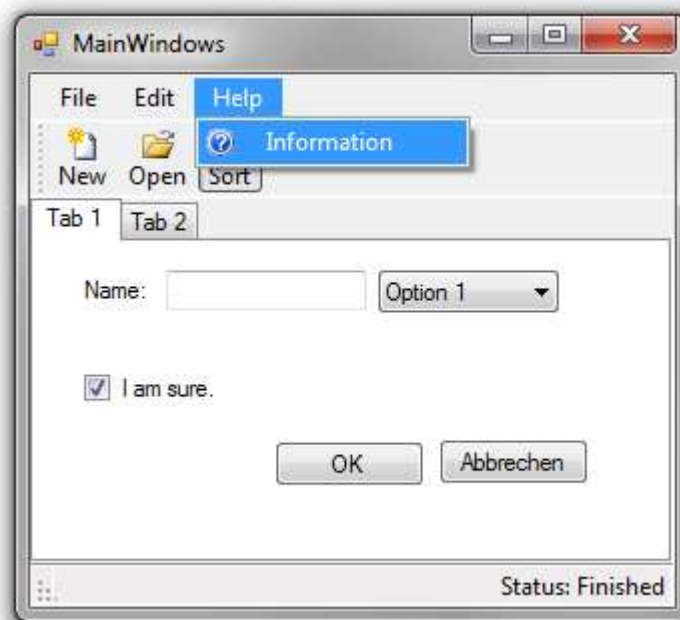


Abbildung B.1: Windows Forms unter Windows



Abbildung B.2: Windows Forms unter Linux



Abbildung B.3: Windows Forms unter Mac



Abbildung B.4: GTK# unter Windows



Abbildung B.5: GTK# unter Linux

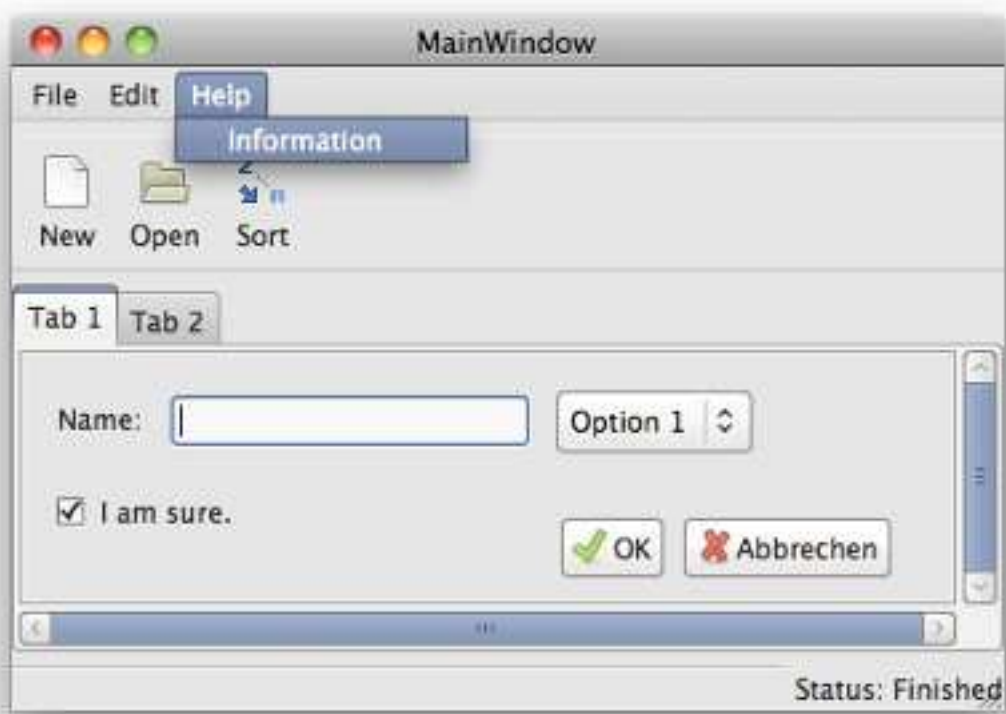


Abbildung B.6: GTK# unter Mac

Anhang C

Quellcode zu 2.2

Listing C.1: Iterative Pi-Berechnung in C++

```
1 #include <cmath>
2
3 int main() {
4
5     long n = 100000000;
6     double a = 1;
7     double b = 1 / sqrt(2);
8     double s = 0.5;
9     double t;
10    double tmp;
11
12    for (int i = 0; i < n; i++) {
13        tmp = a;
14        a = (a + b) * 0.5;
15        b = sqrt(tmp * b);
16        t = pow((a - tmp), 2);
17        t = t * pow(2, i);
18        s = s - t;
19    }
20
21 }
```

Listing C.2: Iterative Pi-Berechnung in C#

```

1 using System;
2
3 public class IterativeBerechnung {
4
5     public static void Main(string[] args) {
6
7         long n = 100000000;
8         double a = 1;
9         double b = 1 / Math.Sqrt(2);
10        double s = 0.5;
11        double t;
12        double tmp;
13
14        for (int i = 0; i < n; i++) {
15            tmp = a;
16            a = (a + b) * 0.5;
17            b = Math.Sqrt(tmp * b);
18            t = Math.Pow((a - tmp), 2);
19            t = t * Math.Pow(2, i);
20            s = s - t;
21        }
22    }
23 }
24 }

```

Listing C.3: Iterative Pi-Berechnung in Java

```

1 public class IterativeBerechnung {
2
3     public static void main(String[] args) {
4
5         long n = 100000000;
6         double a = 1;
7         double b = 1 / Math.sqrt(2);
8         double s = 0.5;
9         double t;
10        double tmp;
11
12        for (int i = 1; i < n; i++) {
13            tmp = a;
14            a = (a + b) * 0.5;
15            b = Math.sqrt(tmp * b);
16            t = Math.pow((a - tmp), 2);
17            t = t * Math.pow(2, i);
18            s = s - t;
19        }
20    }
21 }
22 }

```

Listing C.4: Rekursive Berechnungen in C++

```

1 #include <iostream>
2
3 long ack(long n, long m) {
4
5     if (n==0)
6         return m + 1;
7     else if(m == 0)
8         return ack(n-1,1);
9     else
10        return ack(n-1,ack(n, m-1));
11
12 }
13
14 int main() {
15     ack(4,1);
16 }

```

Listing C.5: Rekursive Berechnungen in C#

```

1 using System;
2
3 public class RekursiveBearbeitung {
4
5     public static void Main(string[] args) {
6         ack(4,1);
7     }
8
9     public static long ack(long n, long m) {
10        if (n == 0)
11            return m + 1;
12        else if (m == 0)
13            return ack(n-1,1);
14        else
15            return ack(n-1,ack(n,m-1));
16    }
17 }

```

Listing C.6: Rekursive Berechnungen in Java

```

1 public class RekursiveBerechnung {
2
3     public static void main(String[] args) {
4         ack(4,1);
5     }
6
7     public static long ack(long n, long m) {
8         if (n == 0)
9             return m + 1;
10        else if (m == 0)
11            return ack(n-1,1);
12        else
13            return ack(n-1,ack(n,m-1));
14    }
15 }

```

Listing C.7: Stringkonkatenation in C++

```

1 #include <string>
2
3 using namespace std;
4
5 int main() {
6
7     long n = 500000000;
8     string str = "a";
9     for(int i = 0; i < n; i++)
10         str += "a";
11
12     return str[2999999];
13
14 }

```

Listing C.8: Stringkonkatenation in C#

```

1 using System;
2 using System.Text;
3
4 public class Stringkonkatenation {
5
6     public static void Main(string[] args) {
7
8         long n = 500000000;
9         StringBuilder str = new StringBuilder("a");
10        for(int i = 0; i < n; i++)
11            str.Append("a");
12
13    }
14
15 }

```

Listing C.9: Stringkonkatenation in Java

```

1 public class Stringkonkatenation {
2
3     public static void main(String[] args) {
4
5         long n = 500000000;
6         StringBuffer str = new StringBuffer("a");
7
8         for(int i = 0; i < n; i++) {
9             str.append("a");
10        }
11
12    }
13
14 }

```

Listing C.10: Verschachtelte Schleifen in C++

```

1 using namespace std;
2
3 int main() {
4
5     long ctr = 42;
6     long x = 0;
7
8     for(int i = ctr; i > 0; i--)
9         for(int j = ctr; j > 0; j--)
10            for(int k = ctr; k > 0; k--)
11                for(int l = ctr; l > 0; l--)
12                    for(int m = ctr; m > 0; m--)
13                        for(int n = ctr; n > 0; n--)
14                            x = n + i + k;
15
16 }

```

Listing C.11: Verschachtelte Schleifen in C#

```

1 public class VerschachtelteSchleifen {
2
3     public static void Main(string[] args) {
4
5         long ctr = 42;
6         long x = 0;
7
8         for(long i = ctr; i > 0; i--)
9             for(long j = ctr; j > 0; j--)
10                for(long k = ctr; k > 0; k--)
11                    for(long l = ctr; l > 0; l--)
12                        for(long m = ctr; m > 0; m--)
13                            for(long n = ctr; n > 0; n--)
14                                x = n + i + k;
15
16     }
17
18 }

```

Listing C.12: Verschachtelte Schleifen in Java

```

1 public class VerschachtelteSchleifen {
2
3     public static void main(String[] args) {
4
5         long ctr = 42;
6         long x = 0;
7
8         for(long i = ctr; i > 0; i--)
9             for(long j = ctr; j > 0; j--)
10                for(long k = ctr; k > 0; k--)
11                    for(long l = ctr; l > 0; l--)
12                        for(long m = ctr; m > 0; m--)
13                            for(long n = ctr; n > 0; n--)
14                                x = n + i + k;
15
16     }
17
18 }

```

Listing C.13: Dateioperationen in C++

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main() {
5
6     long n = 100000;
7     int bs = 1000;
8     char* b = (char*)malloc(sizeof(char)*bs);
9
10    FILE* fs = fopen("/dev/urandom","r");
11
12    for(int i = 0; i < n; i++) {
13
14        char* c = new char[i % 10 + 10];
15        sprintf(c,"/tmp/test%i",i);
16        fread(b,1,bs,fs);
17        FILE* tmp = fopen(c,"w");
18        fwrite(b,1,bs,tmp);
19        fclose(tmp);
20
21    }
22
23    fclose(fs);
24
25    for(int i = 0; i < n; i++) {
26        char* c = new char[i % 10 + 10];
27        sprintf(c,"/tmp/test%i",i);
28        remove(c);
29    }
30
31 }

```

Listing C.14: Dateioperationen in C#

```
1 using System;
2 using System.IO;
3
4 public class Dateioperationen {
5
6     public static void Main(string[] args) {
7
8         long n = 100000;
9         int bs = 1000;
10        byte[] b = new byte[bs];
11        FileStream fs = new FileStream(@"dev/urandom", FileMode.Open,
12            FileAccess.Read);
13
14        for(int i = 0; i < n; i++) {
15
16            fs.Read(b,0,bs);
17            FileStream tmp = new FileStream(@"tmp/test"+i,FileMode.Create);
18            tmp.Write(b,0,bs);
19            tmp.Close();
20        }
21        fs.Close();
22
23        for(int i = 0; i < n; i++) {
24            File.Delete("/tmp/test" + i);
25        }
26    }
27
28 }
```

Listing C.15: Dateioperationen in Java

```
1 import java.io.File;
2 import java.io.FileInputStream;
3 import java.io.FileOutputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6
7 public class Dateioperationen {
8
9     public static void main(String[] args) throws FileNotFoundException,
10         IOException {
11
12         long n = 100000;
13         int bs = 1000;
14         byte[] b = new byte[bs];
15         FileInputStream fs = new FileInputStream("/dev/urandom");
16
17         for(int i = 0; i < n; i++) {
18
19             fs.read(b,0,bs);
20             FileOutputStream tmp = new FileOutputStream("/tmp/test" + i);
21             tmp.write(b,0,bs);
22             tmp.close();
23         }
24
25         fs.close();
26
27         for(int i = 0; i < n; i++) {
28             File f = new File("/tmp/test" + i);
29             f.delete();
30         }
31     }
32 }
```

Listing C.16: Objektorientierung in C++

```
1 #include <iostream>
2
3 class parent {
4
5     public:
6         virtual int getValue() { return 0; }
7 };
8
9 class sub1 : public parent {
10     int ret;
11     public:
12         sub1() : ret(1) { };
13         virtual int getValue() { return ret; }
14 };
15
16 class sub2 : public parent {
17     int ret;
18     public:
19         sub2() : ret(2) { };
20         virtual int getValue() { return ret; }
21 };
22
23 class sub3 : public parent {
24     int ret;
25     public:
26         sub3() : ret(3) { };
27         virtual int getValue() { return ret; }
28 };
29
30 int main() {
31
32     int x = 0;
33     parent* p;
34
35     for(int i = 0; i < 100000000; i++) {
36
37         if((i ^ 42) % 3 == 0) {
38             p = new sub1();
39             x += p->getValue();
40             delete p;
41         } else if((i ^ 42) % 3 == 1) {
42             p = new sub2();
43             x += p->getValue();
44             delete p;
45         } else {
46             p = new sub3();
47             x += p->getValue();
48             delete p;
49         }
50
51     }
52
53     std::cout << x << std::endl;
54
55 }
```

Listing C.17: Objektorientierung in C# (parent)

```

1 public class parent {
2
3     public virtual int getValue() {
4         return 0;
5     }
6
7     public static void Main(String[] args) {
8
9         int x = 0;
10
11        parent p;
12
13        for(int i = 0; i < 1000000000; i++) {
14            if((i^42) % 3 == 0) {
15                p = new sub1();
16                x += p.getValue();
17            } else if((i^42) % 3 == 1) {
18                p = new sub2();
19                x += p.getValue();
20            } else {
21                p = new sub3();
22                x += p.getValue();
23            }
24        }
25
26        Console.WriteLine(x);
27
28    }
29
30 }

```

Listing C.18: Objektorientierung in C# (sub1)

```

1 public class sub1 : parent {
2
3     public override int getValue() {
4         return 1;
5     }
6
7 }

```

Listing C.19: Objektorientierung in C# (sub2)

```

1 public class sub2 : parent {
2
3     public override int getValue() {
4         return 2;
5     }
6
7 }

```

Listing C.20: Objektorientierung in C# (sub3)

```

1 public class sub3 : parent {
2
3     public override int getValue() {
4         return 3;
5     }
6
7 }

```

Listing C.21: Objektorientierung in Java (parent)

```

1 public class parent {
2
3     public int getValue() {
4         return 0;
5     }
6
7     public static void main(String[] args) {
8
9         int x = 0;
10
11        parent p;
12
13        for(int i = 0; i < 1000000000; i++) {
14            if((i^42) % 3 == 0) {
15                p = new sub1();
16                x += p.getValue();
17            } else if((i^42) % 3 == 1) {
18                p = new sub2();
19                x += p.getValue();
20            } else {
21                p = new sub3();
22                x += p.getValue();
23            }
24        }
25
26        System.out.println(x);
27
28    }
29
30 }

```

Listing C.22: Objektorientierung in Java (sub1)

```

1 public class sub1 extends parent {
2
3     private int ret = 1;
4
5     public int getValue() {
6         return ret;
7     }
8
9 }

```

Listing C.23: Objektorientierung in Java (sub2)

```
1 public class sub2 extends parent {  
2  
3     private int ret = 2;  
4  
5     public int getValue() {  
6         return ret;  
7     }  
8  
9 }
```

Listing C.24: Objektorientierung in Java (sub3)

```
1 public class sub3 extends parent {  
2  
3     private int ret = 3;  
4  
5     public int getValue() {  
6         return ret;  
7     }  
8  
9 }
```

Listings

| | | |
|------|---------------------------------------------------------------------------|------|
| 3.1 | Schreibschutz einer Datei setzen (abgeändertes Beispiel aus Zero Install) | 13 |
| 3.2 | Speichern von Benutzereinstellungen | 14 |
| 3.3 | Speichern von Benutzereinstellungen mit Betriebssystemweiche | 14 |
| 3.4 | GuiFactory zur Erzeugung plattformspezifischer GUIs | 18 |
| C.1 | Iterative Pi-Berechnung in C++ | VI |
| C.2 | Iterative Pi-Berechnung in C# | VII |
| C.3 | Iterative Pi-Berechnung in Java | VII |
| C.4 | Rekursive Berechnungen in C++ | VIII |
| C.5 | Rekursive Berechnungen in C# | VIII |
| C.6 | Rekursive Berechnungen in Java | VIII |
| C.7 | Stringkonkatenation in C++ | IX |
| C.8 | Stringkonkatenation in C# | IX |
| C.9 | Stringkonkatenation in Java | IX |
| C.10 | Verschachtelte Schleifen in C++ | X |
| C.11 | Verschachtelte Schleifen in C# | X |
| C.12 | Verschachtelte Schleifen in Java | XI |
| C.13 | Dateioperationen in C++ | XI |
| C.14 | Dateioperationen in C# | XII |
| C.15 | Dateioperationen in Java | XIII |
| C.16 | Objektorientierung in C++ | XIV |
| C.17 | Objektorientierung in C# (parent) | XV |
| C.18 | Objektorientierung in C# (sub1) | XV |

| | |
|----------------------------------------------------|------|
| C.19 Objektorientierung in C# (sub2) | XV |
| C.20 Objektorientierung in C# (sub3) | XVI |
| C.21 Objektorientierung in Java (parent) | XVI |
| C.22 Objektorientierung in Java (sub1) | XVI |
| C.23 Objektorientierung in Java (sub2) | XVII |
| C.24 Objektorientierung in Java (sub3) | XVII |