Kapitel 16

ActiveX-Controls: Grundlagen

Wie jede andere Technologie auch, sollten Sie ActiveX-Controls im Auge behalten. Wenn Sie über längere Zeit verfolgen, was Microsoft von sich gibt, müßten Sie recht schnell zu der Überzeugung gelangen, daß ActiveX-Controls zwar nicht die wichtigste Technologie in Sachen Windows, Internet, Intranets, Client-Server-Entwicklung usw. ist, aber daß sie der Welt Frieden bringen, sowohl den Hunger als auch politische Korruption beseitigen und zu einer schnelleren Entwicklung des Reisens mit Überlichtgeschwindigkeit beitragen werden. Verfolgen Sie die Äußerungen anderer Unternehmen und gelehrter Köpfe, könnten Sie darauf aufmerksam werden, daß ActiveX-Controls von Grund auf unsicher, übermäßig komplex und nur mäßig standardisiert seien, und daß sie zu Gehirnerweichung, zum Dritten Weltkrieg und zur endgültigen Herrschaft von Microsoft führen würden, während Bill Gates fortfahren wird, jedes Software-Unternehmen dieser Welt aufzukaufen.

Ähem ...

An manchen Tagen beschleicht mich der Verdacht, daß die PR-Fritzen unserer Industrie schamlos Anleihen bei ihren Kollegen machen, die politische Kampagnen auf die Menschheit loslassen. Das Problem dieser Anleihen ist jedoch, daß sie einem Programmierer nicht gerade hilfreich bei der Entscheidung über einzusetzende Technologien sind. Ich denke es ist wichtig, das Thema ActiveX-Controls zunächst mit dem Versuch anzugehen, einige der heiß diskutierten Punkte ins rechte Licht zu rücken. Eines der schlimmsten Dinge, die einem Programmierer passieren können, ist die Investition von viel Aufwand und Zeit in eine Technologie, um dann früher oder später doch festzustellen, daß es die falsche Technologie und daher alle Mühe vergebens war.

Ich hoffe, daß dieses Kapitel solche Fehlentscheidungen zu vermeiden hilft, oder zumindest dazu verhilft, sie frühzeitig genug zu erkennen.

16.1 Was jeder Programmierer über ActiveX-Controls wissen sollte

Was ist ein ActiveX-Control? In den nächsten Kapiteln wird es um die technische Antwort auf diese Frage gehen. Doch ganz allgemein betrachtet, kennzeichnet ein ActiveX-Control folgendes aus:

1. Es kann alles, was auch ein In-Process-DLL-Server kann.
2. Es kann in einem Container plaziert werden. Das bedeutet, daß ein Control seine eigene, unabhängige Benutzeroberfläche hat, die im Fenster einer anderen Anwendung enthalten ist, sei es in einem Visual-Basic-Formular oder auf einer Webseite in einem Internet-Browser. Ein Control kann auf verschiedene Weise mit seinem Container interagieren.

3. Es unterstützt zur Design-Zeit:
   - Eigenschaften-Seiten
   - Die Möglichkeit, zur Design-Zeit gesetzte Eigenschaften-Werte an einer vom Container festgelegten Stelle zu speichern
   - Die Möglichkeit zur Interaktion mit einem gegebenenfalls vorhandenen Eigenschaften-Browser des Containers

Mit anderen Worten:

ActiveX-Controls = Code-Komponenten + Benutzeroberfläche + Container-Interaktion

Wir werden auf diese Punkte später noch wesentlich detaillierter eingehen. Der Grund, warum ich sie hier anführe, ist folgender: Sollte Ihre Komponente keine der nur ActiveX-Controls eigenen Eigenschaften und Fähigkeiten benötigen, könnte die Implementierung einer reinen Code-Komponente anstelle eines ActiveX-Controls die bessere Wahl darstellen. ActiveX-Controls sind komplexer als Code-Komponenten – warum sollten Sie sich also mehr Arbeit als nötig machen?

16.1.1 ActiveX-Controls sind plattformabhängig

Zum Zeitpunkt der Veröffentlichung dieses Buchs sind mit Visual Basic erstellte ActiveX-Controls auf folgenden Plattformen lauffähig.

- Windows 95 und Windows 98
- Windows NT 3.51 mit Service-Pack 5 oder höher
- Windows NT 4.0 oder höher

Das war's dann auch schon. Was bedeutet dies nun für Ihre Entscheidung über den Einsatz dieser Technologie? In folgenden Fällen können ActiveX-Controls die ideale Lösung darstellen:

- Sie erstellen Anwendungen für die bezeichneten Plattformen. Dies können Anwendungen auf der Grundlage von Visual Basic, Office 97, anderer ActiveX-Control-fähiger Container oder jeder anderen VBA-basierten Anwendung sein, die ActiveX-Controls unterstützt. Denken Sie daran, daß Microsoft mittlerweile VBA an andere Unternehmen lizenziert und daß ActiveX-Controls von vielen weiteren Entwicklungsumgebungen unterstützt werden.
- Sie entwickeln eine Web-Site, die sich an Anwender richtet, die diese Plattformen benutzen – oder Sie nehmen bei Anwendern mit anderen Plattformen geringere Funktionalität in Kauf. Microsoft hält es so mit der eigenen Web-Site.

- Sie entwickeln eine Web-Site für ein unternehmenseigenes Intranet und die besagten Plattformen sind in dem Unternehmen Standard – oder Sie nehmen auch hier geringere Funktionalität bei anderen Plattformen in Kauf.

Aus diesen Fällen ergeben sich diejenigen, in denen die Verwendung von ActiveX-Controls weniger angebracht ist:

- Ihre Anwendungen müssen noch 16-Bit-Plattformen unterstützen. In diesem Fall können Sie entweder Visual C++ einsetzen, um sowohl 16- als auch 32-Bit-Versionen von ActiveX-Controls zur Verwendung in Visual Basic 4 zu entwickeln, oder Sie verwenden Visual Basic 3 oder 4 zur Entwicklung von 16-Bit-Anwendungen, die auch unter 32-Bit-Windows laufen können.
- Sie wollen eine Web-Site entwickeln, die viele Plattformen unterstützt, einschließlich Apple, Unix, Windows for Workgroups und Windows NT 3.51 ohne Service-Pack 5.

In diesen Fällen könnte eine generische Java-Version (vorausgesetzt, es gibt eine solche) die bessere Wahl sein.

Gelegentlich hört man immer wieder das Gerücht, daß Microsoft die ActiveX-Technologie auch auf andere Plattformen bringen will. Mit allem Respekt gegenüber Microsoft – Sie sollten erst daran glauben, wenn dies auf dem Markt verfügbare Realität geworden ist. Ich vermute jedoch, daß es aus technischen Gründen ein paar Beschränkungen geben wird, sollte Microsoft das Vorhaben tatsächlich realisieren:

- Die Kompilierung in P-Code könnte erforderlich sein, falls nicht eine komplette Emulation von Intels Native-Code auf der betreffenden Plattform zur Verfügung stehen sollte.
- Die Unterstützung von API-Funktionen in einem Control könnte unterbleiben, die, wie Sie noch sehen werden, eine mächtige Technik zur Erstellung von ausgefeilten Controls darstellen.

Es kann natürlich sein, daß andere Anbieter ActiveX-Controls auf Nicht-Windows-Plattformen unterstützen werden. Aber auch hier gilt, daß das erst einmal Markt-Realität werden sollte, bevor man es glauben kann.

Ich möchte jedoch betonen, daß die Frage der Unterstützung von ActiveX-Controls auf Nicht-Windows-Plattformen eine gänzlich andere Sache ist, als die Unterstützung von ActiveX-Controls durch Nicht-Microsoft-Container. Ich schätze, daß immer mehr Windows-Versionen von Nicht-Microsoft-Produkten (einschließlich Web-Browsern) ActiveX-Controls recht gut unterstützen werden. Da es hierbei nicht um andersartige Betriebssysteme geht, gibt es eigentlich keine technischen Gründe, warum Windows-basierte Anwendungen oder Browser nicht ActiveX-Controls unterstützen können sollten. Natürlich mögen dabei einige industriepolitische und strategische Erwägungen in Betracht kommen – ich werde jedenfalls auf dieses Thema hier nicht näher eingehen.

16.1.2 ActiveX-Control-Container sind nicht immer gleich

In einer idealen Welt sollte jede Anwendung, die für sich in Anspruch nimmt, ein Container für ActiveX-Controls zu sein, mit jedem beliebigen ActiveX-Control umgehen können. In einer idealen Welt sollte jedes ActiveX-Control perfekt in jeder Anwendung funktionieren, die einen Container für ActiveX-Controls darstellt.

Es wird Sie nun wohl kaum überraschen, wenn ich Sie darüber informiere, daß wir leider nicht in einer idealen Welt leben.

Die Geschichte der ActiveX-Controls in Kürze

Was ich Ihnen hier erzählen möchte, hat nur wenig oder gar nichts mit der heutigen ActiveX-Control-Technologie oder der Entwicklung von ActiveX-Controls in Visual Basic zu tun. Sehen Sie es als Aussage von jemandem an, der sich damit ein tiefsitzendes traumatisches Erlebnis von der Seele reden möchte. Traumatisches Erlebnis? Ja – Sie müssen nämlich wissen, daß ich eine Quälerei durchstehen mußte, als wir Komponenten-Anbieter den Schritt von der VBX- zur OCX-Technologie während des Wechsels von Visual Basic 3 nach 4 zu bewältigen hatten.

Und es war eine Quälerei. Wie andere Software-Anbieter, die Controls zu Visual Basic beisteuerten, wurden auch wir recht früh über die neue Technologie informiert. Wir haben unmögliche Zeitpläne erhalten. Und dann kamen die Dinge ins schleudern. Im nachhinein betrachtet, dauerte der Übergang ein gutes Jahr länger als zuvor geschätzt.

Das Frustrierende aus unserer Sicht war, daß wir versuchten, Controls auf einer sich laufend ändernden Spezifikation mit sich fortwährend ändernden Tools für sich ständig ändernde Container zu entwickeln, für ein Betriebssystem, daß noch längst nicht erschienen war. Nirgendwo war auch nur eine einzige Zeile stabilen Codes in Sicht.

Es war keineswegs außergewöhnlich, wenn wir im Wochenrhythmus neue Software-Bröckchen erhielten, die in der Regel umfangreiches Neuschreiben von Code erforderten.

Als dann endlich der Zeitpunkt der Auslieferung kam, waren wir wohl alle sehr frustriert und ausgelaugt. Unsere Controls taten einigermaßen ordentlich ihren Dienst in Visual Basic 4, doch wer weitere Container unterstützen wollte, mußte häufig spezifischen Code für diese einfügen. So erschien beispielsweise Access 95 vor Visual Basic 4 und unterstützte nicht alle seiner ActiveX-Control-Merkmale.

Irgendwann erschien dann glücklicherweise Visual Basic 4 und so auch die Controls der Anbieter. Ich kann nicht für andere sprechen, aber ich war doch recht zufrieden mit der Stabilität und Zuverlässigkeit der unsrigen. Der ActiveX-Control-Standard hat sich mittlerweile immerhin ein wenig stabilisert.

Die verschiedenen Gesichter von ActiveX-Containern

Der ActiveX-Control-Standard ist leider kein Standard in der Form, wie man es erwarten würde. Und Sie wissen inzwischen genügend über ActiveX-Technologie, um zu verstehen, warum das so ist.

ActiveX-Technologie beruht auf COM-Objekten. ActiveX-Controls sind COM-Objekte, die bestimmte Interfaces unterstützen. Tatsächlich beruht die ActiveX-Control-Spezifikation im wesentlichen auf der Definition dieser Interfaces.

Das Problem ist dabei folgendes: Die Standards konzentrieren sich zumeist auf die Interfaces selbst – auf die Methoden und Eigenschaften jedes einzelnen Interfaces. Doch aus den Standards geht nicht klar hervor, welche dieser Interfaces ein Container zu unterstützen hat, um wirklich als ActiveX-Container gelten zu können. Es war üblich und nicht gerade selten, daß ein Container nur einen minimalen Satz von Interfaces unterstützte. So gibt es beispielsweise ein Interface namens `ISimpleFrame`, den ein Control implementieren kann, wenn es selbst als Container für andere Controls fungieren kann. Ein Beispiel hierfür ist die `PictureBox`, die andere Controls aufnehmen kann. Ein ActiveX-Container mußte dieses Interface nicht unterstützen – und einige taten es auch nicht.

Dies bedeutet, daß ein Control, das als Container für andere Controls gedacht war, nicht davon ausgehen konnte, daß es dazu auch die Erlaubnis von seinem eigenen Host-Container bekommen würde.

Microsoft unternahm einige Anstrengungen, zu spezifizieren, welche Interfaces notwendig sind und welche sowohl für Container als auch für Controls optional sind. Aber das ist bisher lediglich ein bewegliches Ziel geblieben.

Sie wissen bereits, daß ActiveX-Technologie über die Definition neuer Interfaces erweitert werden kann. Eines der jüngsten Beispiele dafür ist die Internet-Kompatibiliät von ActiveX-Controls – es mußten dazu nur ein paar neue Internet-bezogene Interfaces hinzugefügt werden.

Doch wenn einem Control ein neues Interface hinzugefügt wird, kann es nicht davon ausgehen, daß bereits früher existierende Container diese neuen Features auch unterstützen. Das bedeutet für Sie als Control-Entwickler, daß Sie davon ausgehen müssen, daß nicht alle Container einige der Features Ihres Controls unterstützen werden. Schlimmer ist noch, daß die die Unterstützung von ActiveX-Controls betreffenden Interfaces recht komplex sind, so daß es häufig unterschiedliche Verhaltensweisen bei verschiedenen Containern gibt (das ist eine schönfärberische Umschreibung für »Bugs«), selbst bei Containern vom gleichen Hersteller. Dies läßt Ihnen zwei Möglichkeiten offen: Entweder unterstützen Sie einfach diese Container nicht, oder Sie sorgen dafür, daß sich Ihr Control gnädig verhält und nicht die Container-Anwendung zum Absturz bringt, wenn es nicht funktioniert, indem Sie sorgfältig codieren und Fehlerbehandlungen einbauen.

Es gibt auch Umstände, in denen das Container-Verhalten nicht eindeutig definiert ist. Schauen Sie sich zum Beispiel das Control `ch16ctl1.ocx` im Ordner zu

diesem Kapitel auf der Buch-CD an. Registrieren Sie es und versuchen Sie, es in ein Projekt in Visual Basic 5 oder 6 einzufügen. Merken Sie sich, wie es zur Design-Zeit dargestellt wird. Nehmen Sie zur Kenntnis, daß sowohl die Eigenschaft `TestBool1` als auch die Eigenschaft `TestBool2` im Eigenschaften-Fenster erscheint.

Fügen Sie das Control nun in ein Visual-Basic-4-Projekt ein. Es sieht anders aus, nicht wahr? Und wo ist die Eigenschaft `TestBool1` abgeblieben? Das werden Sie später noch erfahren.

In diesem Buch werde ich versuchen, Container-spezifische Situationen aufzuzeigen. Bedenken Sie jedoch, daß auch ich nur mit einem Bruchteil der heute existierenden Container arbeiten konnte. Und wenn Sie dies hier nun lesen, werden schon wieder einige mehr auf dem Markt erschienen sein.

16.1.3 Mit Visual Basic erstellte Controls unterliegen Beschränkungen

Ich habe bereits erwähnt, daß Visual Basic die Windows-Programmierung vereinfacht hat, indem die Windows-Features hinter einer Basic-Syntax versteckt wurden. Programmieren mit Visual Basic ist längst nicht mehr so einfach wie früher. Doch das liegt in erster Linie am immer weiter wachsenden Umfang von Windows. Nach wie vor stellt Visual Basic einen der einfachsten Wege zum Erstellen von Windows-Anwendungen dar.

Dazu stellt Visual Basic auch einen der sichersten Wege der Windows-Programmierung dar. Programmierer, die Visual C++ oder andere ähnliche Sprachen verwenden, sind an ständig auftretende Speicherausnahmefehler gewöhnt. Visual Basic leistet dagegen hervorragende Arbeit bei dem Bemühen, solche Fehler zu verhindern. So können Speicherausnahmefehler eigentlich nur auf folgende Weise zustandekommen:

- Inkorrekte Aufrufe von API-Funktionen oder API-Erweiterungen der untersten Ebene
- Verwendung des `AddressOf`-Operators
- Bugs in Visual Basic

Vielleicht das schönste Merkmal von Visual Basic ist, daß es Ihnen die Wahl zwischen Leistungsfähigkeit und Sicherheit läßt. Sie können sich auf der sicheren und »einfachen« Seite bewegen und dabei den größten Teils des Windows-Leistungsumfangs nutzen. Sie können die Möglichkeiten von Visual Basic nach und nach und gezielt durch API-Aufrufe und Zusatz-Controls und -DLLs erweitern, um letztlich jede Aufgabenstellung zu lösen. Diese Erweiterungen gehen jedoch zu Lasten der Sicherheit, erhöhen das Risiko von Speicherausnahmefehlern und erhöhen zudem auch den Aufwand beim Erstellen der Programm-Installation, da die Zusatz-Controls und -DLLs auf sichere Weise mitvertrieben und installiert werden müssen.

Diese Problematik ist durch Visual Basics Implementierung zum Erstellen von ActiveX-Controls noch größer geworden.

Visual Basic erlaubt das Erstellen von ActiveX-Controls so einfach wie nie zuvor, indem viele der für ActiveX-Controls notwendigen Interfaces und Objekte von der Sprache und der Entwicklungsumgebung verborgen werden. Die Control-Entwicklung geht unter Visual Basic bemerkenswert schnell und sicher von der Hand.

Der Preis dafür ist jedoch die Opferung einiger Features, die in Visual C++ bei ActiveX-Controls möglich sind. Diese Beschränkungen betreffen im wesentlichen folgende Bereiche:

- Man kann keine 16-Bit-Controls erstellen.
- Nicht alle Ereignisse und Methoden von OLE-Controls stehen Visual-Basic-Programmierern zur Verfügung.
- Nicht-Dispatch-Interfaces werden nicht unterstützt.
- Eigenschaften, die nur zur Design-Zeit zur Verfügung stehen, können zur Laufzeit nicht modifiziert werden.

Wir werden auf diese Beschränkungen in den nachfolgenden Kapiteln näher eingehen.

Das Gute an Visual Basic ist jedoch wieder, daß Sie auch hier über direkte API-Aufrufe und Zusatz-Tools Visual-Basic-Controls dazu bringen können, nahezu jede erdenkliche Aufgabe zu meistern.

Weiterhin haben mit Visual Basic erstellte Controls sogar einige Vorteile gegenüber mit Visual C++ erstellten Controls. So ist es unter Visual C++ eine ungemein schwierige Aufgabe, ein Control aus vorhandenen (»konstituierenden«) Controls zusammenzusetzen.

Es ist bestimmt nicht die Absicht dieses Abschnitts, der sich mit den Beschränkungen befaßt, Sie davon abzuhalten, mit Visual Basic ActiveX-Controls zu entwickeln. Im Gegenteil, ich denke, Sie werden mir zustimmen, daß Visual Basic das beste Werkzeug zum Erstellen von ActiveX-Controls ist. Die Absicht liegt vielmehr darin, ein paar Stolpersteine aus dem Weg zu räumen, so daß Sie sich der Technologie mit einem besseren Verständnis der Fähigkeiten und Beschränkungen nähern können.

16.2 Design-Zeit versus Laufzeit versus Design-Zeit versus Laufzeit

Wenn Sie mit ActiveX-Controls arbeiten, müssen Sie immer wissen, ob sich das Control gerade im Design-Zeit-Zustand befindet, der dazugehörende Designer also geöffnet ist, oder ob es sich im Break-Modus befindet, der Designer also

geschlossen ist. Der Break-Modus gehört auch zur Design-Zeit. Sie müssen daher zwischen dem Zustand der Design-Zeit aus der Sicht eines Control-Entwicklers und dem Zustand der Design-Zeit unterscheiden, in dem sich der Container befindet, der das Control enthält. In der Regel befindet sich das Control im Laufzeit-Zustand, wenn es sich aus der Sicht eines Anwenders, also eines Programmierers, der Ihr Control verwendet, im Design-Zeit-Zustand befindet. Denken Sie auch daran, daß sich konstituierende Controls im Laufzeit-Zustand befinden, ausgenommen, der Designer Ihres Controls ist gerade geöffnet.

Und bis jetzt war auch noch nicht die Rede davon, wenn der Anwender Ihres Controls sein Projekt in den Laufzeit-Zustand versetzt.

Falls Sie diesen Aspekt bereits verstanden haben sollten, ist das beeindruckend. Ich werde nämlich den Rest dieses Kapitels damit verbringen, genau das alles zu erklären.

16.2.1 Eine Frage der Perspektive

Wahrscheinlich ist das wichtigste, bevor Sie mit der Entwicklung von ActiveX-Controls in Visual Basic beginnen, ein grundlegendes Verständnis der verschiedenen Objekte, die Sie bei der Entwicklung von Controls verwenden, wie sie miteinander zusammenhängen, und die verschiedenen Modi, die sie unterstützen.

Ohne dieses Verständnis werden Sie rettungslos verloren sein, wenn Sie herauszufinden versuchen, welche Eigenschaft zu welchem Objekt gehört. Bezieht sich beispielsweise die Eigenschaft »Name« auf ein konstituierendes Control (und wenn ja, auf welches?), auf den Namen ihres Controls, oder auf den Namen, den ein Verwender des Controls diesem zuweist? Alle diese Namens-Eigenschaften können zur gleichen Zeit in Ihren Controls existieren und werden mit Sicherheit einen anderen Inhalt haben.

Vielleicht wird der eine oder andere sich bereits beim Lesen des Begriffs »konstituierende Controls« etwas verloren vorkommen.

Beginnen wir also mit einem etwas philosophischeren Blick auf ActiveX-Controls. Dann werden wir auf Ihren Kenntnissen von COM-Objekt und der ActiveX-Technologie aufbauen und sehen, wie sich eins zum anderen fügt.

Betrachten wir zunächst ein Standard-Control, etwa eine ListBox. Zunächst einmal ist ein in Visual Basic bereits eingebautes Controls (z.B. eine ListBox) dasselbe, wie ein von Ihnen erstelltes Control. Der einzige Unterschied liegt darin, daß so ein eingebautes Control (auch »intrinsic Control« genannt) in Visual Basic selbst statt in einer separaten OCX-Datei implementiert ist.

Wenn Sie eine ListBox auf ein leeres Formular zeichnen, haben Sie ein COM-Objekt eingefügt. Wie jedes andere Objekt auch, verfügt es über Eigenschaften, Methoden und über die Eigenschaft, Ereignisse auszulösen. So fügen Sie etwa ein Element in eine ListBox über folgende Anweisung ein:

```
List1.AddItem
```

Die Ereignisse der ListBox erscheinen nach dem Muster `List1_Ereignisname` - zum Beispiel:

```
Private Sub List1_Click()
'...
End Sub
```

Sie werden bemerkt haben, daß die Syntax eines Control-Ereignisses identisch zu der von Ereignissen einer ActiveX-Code-Komponente und zu der von über die `Implements`-Anweisung einem Objekt hinzugefügten Interface-Funktionen ist. Bei dieser Syntax wird sowohl der Interface-Name als auch der Name des Ereignisses angegeben. `List1_Click` bedeutet das Click-Ereignis des List1-Ereignis-Interfaces.

Nun möchte ich Ihnen eine Frage stellen. Wenn sich Ihr Formular im Design-Zeit-Zustand befindet, existieren dann die Eigenschaften der ListBox und deren Ereignisse? Die Antwort lautet: ja, selbstverständlich. Sie können jedoch nicht allzuviel damit anfangen. Dafür gibt es zwei Gründe: Zum einen Visual Basic führt zur Design-Zeit nichts vom Code Ihrer Anwendung aus. Somit geschieht in Ihrem Code nichts, wenn das ListBox-Control ein Ereignis auslöst. Zum anderen kann Ihr Code keine Eigenschaft der ListBox setzen, wenn er nicht ausgeführt wird.

Sie können jedoch die Eigenschaften der ListBox über das Eigenschaften-Fenster setzen. Soweit es das ListBox-Control selbst betrifft, bekommt es nur mit, daß eine Eigenschaft gesetzt wird. Es kann nicht unterscheiden, ob die Eigenschaft über das Eigenschaften-Fenster oder über Code gesetzt worden ist.

Allerdings weiß das Control, ob sich sein Container im Design-Zeit- oder im Laufzeit-Zustand befindet, da die ActiveX-Spezifikation einem Control einen Weg anbietet, auf dem es den Container fragen kann, in welchem Zustand er sich befindet. Das ListBox-Control kann daraufhin entscheiden, wie es den Zugriff auf die Eigenschaft in Abhängigkeit vom Zustand des Containers behandeln soll. Wenn Sie über das Visual-Basic-Eigenschaften-Fenster die Hintergrundfarbe einer ListBox setzen, wird die Änderung sofort sichtbar. Wenn Sie jedoch eine neue ListBox in ein Formular einfügen, enthält sie zur Design-Zeit den Namen des Controls, und zur Laufzeit überhaupt nichts (solange Sie nicht über die List-Eigenschaft im Eigenschaften-Fenster bereits zur Design-Zeit Elemente einfügen). Und es gibt Eigenschaften, wie etwa `MultiSelect`, die nur zur Design-Zeit gesetzt werden können.

Merken Sie sich, daß diese unterschiedlichen Verhaltensweisen von Controls nicht von Visual Basic festgelegt werden oder abhängen. Sie werden vom Control selbst bestimmt. Visual Basic weiß nur deswegen darüber Bescheid, welche Eigenschaften nur zur Design-Zeit gesetzt werden können, weil das Control die entsprechende Information geliefert hat. Das Control wiederum kann diese Information nur deswegen liefern, weil sein eigener Code ausgeführt wird.

Abbildung 16.1 stellt dieses Szenario aus der Sicht eines Verwenders eines Controls dar. Das Control läuft immer, kann jedoch sein Verhalten in Abhängigkeit vom Zustand des Containers ändern. Zur Visual-Basic-Design-Zeit läuft der größte Teil der Kommunikation mit dem Control über das Eigenschaften-Fenster ab. Zur Visual-Basic-Laufzeit und in kompilierten Anwendungen geht der Großteil der Kommunikation über den Code der Anwendung.

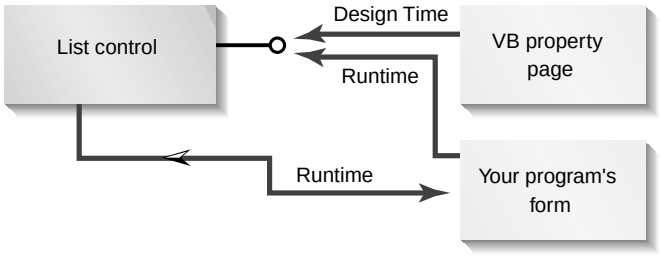

Abb. 16.1: Ein Control aus der Sicht eines Verwenders (Entwicklers)

16.2.2 Aus der Sicht des Autors

Als Autor eines Controls haben Sie eine andere Sichtweise.

Ihre Aufgabe ist, ein Control zu erstellen, das Eigenschaften, Methoden und Ereignisse zur Verfügung stellt, wenn es ausgeführt wird. Es liegt an Ihnen, den Container zu fragen, ob er sich im Design-Zeit-Zustand oder im Laufzeit-Zustand befindet, und davon abhängend die Aktionen Ihres Controls entsprechend anzupassen. Der Entwickler, der Ihr Control verwendet, kann nur auf die Eigenschaften, Methoden und Ereignisse zurückgreifen, die Sie als öffentlich deklariert haben.

Diese Terminologie folgt den Vorgaben von Microsofts Dokumentation. Sie sind der *Autor* des Controls. Ein *Entwickler* ist derjenige, der Instanzen Ihres Controls auf einem seiner Formulare anlegt.

Controls entwickeln sich natürlich nicht von alleine aus dem Nichts. Aus Ihrer Sicht als Autor kann sich Ihr Control sowohl im Design-Zeit-Zustand als auch im Laufzeit-Zustand befinden. Controls sind in dieser Hinsicht jedoch ein wenig komplizierter als Anwendungen.

In Kapitel 9, »Komponenten erstellen und testen«, habe ich einiges Aufhebens über Designer gemacht und zwischen Designern, in denen Sie die Benutzeroberfläche eines Objekts gestalten, etwa einem Formular, und Code-Fenstern unterschieden, in denen Sie den Code eines Objekts bearbeiten. Das hatte unter anderem auch den Hintergrund, daß das Verhalten eines ActiveX-Controls davon abhängt, ob sein Designer geöffnet oder geschlossen ist.

Ist der Designer eines Controls geöffnet, ist es für seine Clients nicht erreichbar. Ist der Designer geschlossen, können Sie das Control in ein Formular in der glei-

chen Visual-Basic-Instanz einfügen. Sie können Haltepunkte im Code Ihres Controls setzen und im Debug-Modus sowohl durch den Code des Controls als auch durch den Code, der das Control verwendet, durchlaufen.

Der erste Schritt zum Testen Ihres Controls lautet also: Schließen Sie den Designer des Controls. Damit wird die Ausführung des Controls gestartet. Dies bedeutet, daß Sie den Designer schließen können, einen Haltepunkt im Code Ihres Controls setzen können, das Control auf einem Formular plazieren können und daß Sie im Debug-Modus durch den Code des gerade ausgeführten Controls gehen können, obwohl sich der Container im Design-Zeit-Zustand befindet!

Wie legen Sie Methoden, Eigenschaften und Ereignisse in einem Control an? Und wie unterscheiden Sie zwischen den Methoden und Eigenschaften, die Sie nach außen hin offenlegen, und denen, die Sie zum Erstellen des Controls benötigen?

16.2.3 Ein ganzer Haufen Objekte

Eine ganze Reihe verschiedener Objekte ist am Entstehen eines Controls beteiligt. Die wichtigsten sind:

- Ihr Control-Objekt
- Das UserControl-Objekt
- Das Ambient-Objekt
- Das Extender-Objekt
- Konstituierende Control-Objekte

Schauen wir uns diese der Reihe nach an, indem wir als Beispiel ein einfaches Control anlegen. Dieses Control soll ein privates Control sein, das in eine Anwendung direkt eingebunden wird. Sie können davon keine OCX-Datei erstellen, doch das ist der einzige Unterschied zwischen einem privaten und einem öffentlichen ActiveX-Control.

Öffnen Sie ein neues Standard-EXE-Projekt. Fügen Sie ein UserControl-Objekt hinzu. Der Designer des Controls wird geöffnet. Ändern Sie den Namen des Controls von `UserControl1` in `MyControl`. Ändern Sie den Namen des Projekts von `Project1` in `Ch16CtlA`.

Klicken Sie im Designer auf die Fläche des Controls. Die Eigenschaften, die daraufhin im Eigenschaften-Fenster angezeigt werden, sind die des UserControl-Objekts Ihres Controls. Die Klasse `MyControl`, die Sie angelegt haben, um Ihr Control zu implementieren, enthält ein UserControl-Objekt, mit dessen Hilfe Sie Ihr Control erstellen.

Mit einem Doppelklick in den Designer öffnen Sie das Code-Fenster des Controls. Fügen Sie eine Eigenschaft über den folgenden Code ein:

```
Dim m_MyProp As String

Public Property Get MyProp() As String
   MyProp = m_MyProp
End Property

Public Property Let MyProp(vNewValue As String)
   m_MyProp = vNewValue
End Property
```

Öffnen Sie das Formular des Projekts. Ändern Sie den Namen des Formulars in `MyForm`. Dieses Formular werden Sie zum Testen des Controls verwenden. In der Werkzeugsammlung wird das Symbol für Ihr Control gesperrt (grau) dargestellt.

Schließen Sie nun den Designer des Controls. Das Symbol des Controls wird wieder vollständig sichtbar. Legen Sie eine Instanz des Controls in dem Formular an. Der vorgegebene Name der neuen Control-Instanz wird aus dem Namen des UserControl-Objekts abgeleitet, er lautet hier also `MyControl1`. Dies ist auch der Name des Objekt-Typs.

Im Eigenschaften-Fenster sehen Sie die zuvor angelegte Eigenschaft `MyString`. Sie sehen auch eine ganze Menge weiterer Eigenschaften. Diese Eigenschaften werden Extender-Eigenschaften genannt, da sie von einem weiteren, dem Extender-Objekt stammen, das vom Container (hier ist es Visual Basic) zur Verfügung gestellt wird.

Ohne den Formular-Designer zu schließen, öffnen Sie mit einem Doppelklick im Projekt-Fenster erneut den Designer des Controls. Im Formular wird das Control mit einer diagonalen Schraffur versehen, die anzeigt, daß das Control im Moment nicht verfügbar ist. Holen Sie wieder das Code-Fenster des Controls in den Vordergrund und geben Sie folgenden Ereignis-Code ein:

```
Event Click() ' Im (Allgemein)-Bereich des Code-Fensters

Private Sub UserControl_Click()
    RaiseEvent Click
End Sub
```

Schließen Sie den Designer wieder. Fügen Sie im Code-Fenster des Formulars `MyForm` den folgenden Code ein:

```
Private Sub MyControl1_Click()
   MsgBox "MyControl was clicked"
End Sub
```

Starten Sie nun das Projekt (Vergewissern Sie sich, daß `MyForm` das Start-Objekt des Projekts ist). Klicken Sie auf das Control im Formular. Die MessageBox wird erscheinen, wenn das Click-Ereignis des Controls ausgelöst wird.

Stoppen Sie das Projekt und öffnen Sie erneut den Designer des Controls. Plazieren Sie eine Schaltfläche (CommandButton) im Control. Fügen Sie folgenden Code im Code-Fenster des Controls ein:

```
Event ButtonClick() ' Im (Allgemein)-Bereich

Private Sub Command1_Click()
   RaiseEvent ButtonClick
End Sub
```

Starten Sie das Projekt wieder. Klicken Sie nun sowohl auf das Control selbst als auch auf die Schaltfläche, die sich darauf befindet. Aus Ihrer Sicht als Autor handelt es sich um zwei verschiedene Click-Ereignisse von zwei verschiedenen COM-Interfaces, das eine vom UserControl-Objekt, das andere vom `CommandButton`-Objekt, das ein konstituierendes Control ist. Daraufhin lösen Sie zwei verschiedene Ereignisse über das Ereignis-Interface Ihres Controls aus. Der Entwickler als Verwender sieht beide Ereignisse als Ereignisse von `MyControl1`.

Öffnen Sie wieder den Designer des Controls. Plazieren Sie ein Label-Control auf dem Control. Fügen Sie folgenden Code in die Ereignis-Prozedur `UserControl_Paint` ein:

```
Private Sub UserControl_Paint()
  If Ambient.UserMode Then
    Label1.Caption = "Container is in Run mode"
  Else
    Label1.Caption = "Container is in Design mode"
  End If
End Sub
```

Schließen Sie nun den Designer und sehen Sie sich das Formular zur Design-Zeit an. Starten Sie nun wieder das Projekt. Dies demonstriert, wie das Control sein Verhalten abhängig davon ändern kann, ob sich sein Container im Design-Zeit-Zustand oder im Laufzeit-Zustand befindet. Dazu verwendet das Control das Ambient-Objekt, das Informationen über die Umgebung des Controls, also über den Container, anbietet.

Schauen wir uns diese fünf Objekte näher an.

Ihr Control-Objekt

Der einzige Teil des ActiveX-Controls, den Sie als Autor selbst anlegen, ist das Control-Objekt selbst, hier das `MyControl`-Objekt. Dieses dient zweierlei Zwecken: Hier werden die öffentlichen Methoden, Eigenschaften und Ereignisse definiert, die zur Verwendung Ihres Controls zur Verfügung stehen, und hier wird das Verhalten des Controls definiert.

Neben den Methode und Eigenschaften, die Sie implementieren, haben Sie Zugriff auf die Eigenschaften des UserControl-, des Extender- und des Ambient-Objekts und aller konstituierenden Controls, die Sie auf Ihrem Control plaziert haben.

Das UserControl-Objekt

Aus Ihrer Sicht als Autor und Programmierer des Controls ist das UserControl-Objekt ein Unterobjekt Ihres Controls. Sie greifen auf Methoden des UserControl-Objekts über die Syntax `UserControl.Methode` zu. Das UserControl-Objekt kann Ereignisse in Ihrem Control-Objekt auslösen, auf die Sie im Code des Controls reagieren können. Beachten Sie, das damit keine Ereignisse in den Instanzen Ihres Controls ausgelöst werden, die ein Entwickler bei der Verwendung Ihres Controls anlegt. Wollen Sie, daß ein Ereignis des UserControl-Objekts, wie etwa das Ereignis `UserControl_Click`, dem Verwender zur Verfügung stehen soll, müssen Sie in Ihrem Control-Objekt ein neues Ereignis definieren und das Ereignis im Code Ihres Controls selbst auslösen.

Aus der Perspektive einer Visual-Basic-Anwendung sieht es umgekehrt so aus, als ob Ihr Control ein Unterobjekt des UserControl-Objekts wäre. Dies rührt daher, daß es das UserControl-Objekt ist, das Visual Basic kennt und von dem es weiß, wie es in einem Container einzufügen ist. Visual Basic und Windows leiten Maus- und Tastatureingaben an das UserControl-Objekt weiter, die dort verarbeitet werden und in Ihrem Control-Code beliebig behandelbare Ereignisse auslösen. Das UserControl-Objekt ist zuständig für das Fenster des Controls und für die Behandlung der Aktivierung, des Eingabe-Fokus und aller übrigen Belange der Benutzeroberfläche. Alle diese Aktivitäten werden Ihrem Control-Objekt über Ereignisse und Eigenschaften zur Verfügung gestellt.

Das UserControl-Objekt ist das Standard-Objekt ihres Controls. Dies ist genauso, als wenn Sie im Code eines Formulars die Eigenschaft `BackColor` verwenden. Hierbei wird standardmäßig die `BackColor`-Eigenschaft des Formulars angesprochen. Legen Sie jedoch in dem Formular eine neue Eigenschaft namens `BackColor` an, wird diese neue `BackColor`-Eigenschaft die von dem Formular zur Verfügung gestellte Standard-Eigenschaft überdecken. Dies entspricht exakt den Regeln zum Gültigkeitsbereich von Variablen, die Sie in Kapitel 10, »Code und Klassen«, kennengelernt haben. Lokal definierte Eigenschaften und Methoden überdecken globale Eigenschaften und Methoden des Standard-Objekts. Sie können natürlich nach wie vor auf die `BackColor`-Eigenschaft des Formulars zugreifen, indem Sie das Formular explizit angeben, beispielsweise `Form1.BackColor`. Genauso verhält es sich beim UserControl-Objekt. Sie können in Ihrem Control die Eigenschaft BackColor definieren und dann über `UserControl.BackColor` auf die Standard-Eigenschaft `BackColor` des UserControl-Objekts weiterhin zugreifen.

Das Ambient-Objekt

Die Eigenschaften des Ambient-Objekts stellen Informationen über Fähigkeiten des Containers zur Verfügung, die Sie beliebig benutzen können. Weiter oben haben Sie bereits die Eigenschaft `UserMode` kennengelernt, anhand derer Sie feststellen können, ob sich der Container im Design-Zeit-Zustand oder im Laufzeit-Zustand befindet. Ein anderes Beispiel ist die `BackColor`-Eigenschaft, die die Hintergrundfarbe des Containers liefert. Diese brauchen Sie, wenn Sie Ihr Control automatisch dem Hintergrund des Containers anpassen wollen. Das UserControl-Objekt bietet das Ereignis `AmbientChanged`, das Sie über die Änderung einer Eigenschaft des Ambient-Objekts benachrichtigt.

Container können Ihre eigenen Ambient-Eigenschaften definieren und müssen nicht unbedingt diejenigen implementieren, die Sie im Visual-Basic-Objekt-Katalog und in der Visual Basic-Dokumentation finden. Damit das Leben jedoch etwas einfacher wird, entdeckt das Ambient-Objekt fehlende Eigenschaften des Containers und gibt dann einen Standard-Wert für die betreffende Eigenschaft zurück.

Heißt das, daß ein Container nicht unbedingt eine Eigenschaft wie `UserMode` bieten muß? Ja, genau so ist es. Wenn diese Eigenschaft jedoch nicht implementiert ist – wie kann Ihr Control denn feststellen, ob es sich im Design-Zeit-Zustand oder im Laufzeit-Zustand befindet? Das kann es dann eben nicht. Es gibt Container, bei denen wird nicht zwischen Design-Zeit und Laufzeit unterschieden. Diese Container brauchen auch keine UserMode-Eigenschaft anzubieten. In diesem Fall würde das Ambient-Objekt immer True zurückgeben, was besagt, daß sich der Container immer im Laufzeit-Zustand befindet.

Das Extender-Objekt

Es gibt einige Eigenschaften von Controls, die nichts mit der Funktionalität des Controls selbst zu tun haben. So hat beispielsweise jedes Control in Visual Basic die Eigenschaft `Name`, die vom Container selbst verwaltet wird. Andere Eigenschaften, die in diese Kategorie fallen, sind die Eigenschaften `Visible` und die Positions-Eigenschaften wie `Left` und `Top`.

Wenn ein Entwickler Ihr Control verwendet, fügt der Container diese Eigenschaften automatisch zum Interface Ihres Controls hinzu, so daß der Entwickler auf sie zugreifen kann. Sie werden Extender-Eigenschaften genannt. Der Entwickler kann nicht zwischen den Eigenschaften, die vom Container stammen, und den von Ihrem Control selbst stammenden Eigenschaften unterscheiden. Doch Sie, der Control-Autor, können das. Zum einen brauchen Sie die Extender-Eigenschaften nicht selbst zu implementieren – sie könnten es auch gar nicht. Doch Sie können auf diese Container-seitigen Extender-Eigenschaften über das Extender-Objekt zugreifen.

Dieses Objekt hat jedoch einen Haken: Es gibt keine Möglichkeit festzustellen, über welche Eigenschaften ein Container verfügt. Obwohl es ein paar Standard-Eigenschaften gibt, die die meisten Container unterstützen, müssen Sie immer die

Existenz einer Eigenschaft vorher prüfen (eine entsprechende Fehlerbehandlung einbauen), ehe Sie darauf zugreifen, damit Ihr Control in jedem Container verwendet werden kann.

Da Extender-Objekte nicht im voraus bestimmt werden können, ist der Zugriff darauf immer spät gebunden.

Konstituierende Control-Objekte

Diese Objekte sind am einfachsten zu handhaben. Die Arbeit mit ihnen ist weitgehendst dieselbe wie in einem Formular. Sie zeichnen sie auf Ihrem Control wie auf einem Formular ein. Sie greifen auf genau die gleiche Weise zu und sie stellen genauso ihre Ereignisse zur Verfügung. Denken Sie jedoch daran, daß ein Verwender Ihres Controls – anders als bei einem Formular – keinen Zugriff auf die Eigenschaften, Methoden und Ereignisse eines konstituierenden Controls hat, solange Sie es nicht ausdrücklich offenlegen.

Es gibt ein paar interessante Seiteneffekte bei der Verwendung von konstituierenden Controls in einem ActiveX-Control. Diese Seiteneffekte hängen mit dem Fokus, mit Zugriffstasten und dem Fokus-Wechsel über die Tabulator-Taste zusammen. Doch das ist ein Thema für das folgende Kapitel.

Konstituierende Objekte in einem ActiveX-Control haben eine grundlegende Beschränkung. Diese resultiert aus der Frage, wann sich ein konstituierendes Control im Laufzeit-Zustand und wann es sich im Design-Zeit-Zustand befindet.

Tabelle 16.1 stellt die möglichen Zustände eines ActiveX-Controls, einer Client-Anwendung und von konstituierenden Controls dar. Sie sehen, daß der Code Ihres Controls nicht ausgeführt werden kann, wenn sein Designer geöffnet ist (Zustand: Control-Design-Zeit). In diesem Zustand können die Design-Zeit-Eigenschaften von konstituierenden Controls über das Visual-Basic-Eigenschaften-Fenster gesetzt werden. Sobald der Designer geschlossen wird, wird Ihr Control ausgeführt und die konstituierenden Controls befinden sich im Laufzeit-Zustand, unabhängig vom Zustand des Containers.

Dies bedeutet, daß in Visual Basic die Design-Zeit-Eigenschaften im Code Ihres Controls nicht gesetzt werden können. Sie haben dort nur Zugriff auf die Laufzeit-Eigenschaften eines konstituierenden Controls. So könnten Sie beispielsweise die Eigenschaft `MultiSelect` einer ListBox nicht mehr ändern, selbst dann nicht, wenn sich der Container im Design-Zeit-Zustand befindet. Es gibt mehrere Möglichkeiten, dieses Problem zu umgehen, wie Sie später noch sehen werden.

Denken Sie auch daran, daß Sie in Visual Basic 6.0 Controls dynamisch zur Laufzeit anlegen können. Auf dieses Thema sind wir bereits in Kapitel 11, »Ereignisse«, gestoßen. Controls, die auf einem UserControl zur Laufzeit angelegt werden, entsprechen konstituierenden Controls und werden direkt im Laufzeit-Zustand mit den Vorgabe-Werten der Eigenschaften angelegt.

ActiveX-Designer	Client-Anwendung, die das Control verwendet	Zustand konstituierender Controls
Geöffnet – Control wird nicht ausgeführt	Control ist inaktiv	Design-Zeit
Geschlossen – Control wird ausgeführt bzw. befindet sich im Unterbrechungs-Modus	Design-Zeit	Laufzeit
Geschlossen – Control wird ausgeführt bzw. befindet sich im Unterbrechungs-Modus	Laufzeit oder Unterbrechungs-Modus	Laufzeit – Control wird ausgeführt bzw. befindet sich im Unterbrechungs-Modus

Tab. 16.1: *Ausführungszustände von ActiveX-Controls und konstituierenden Controls*

Abbildung 16.2 faßt das Objekt-Modell von ActiveX-Controls zusammen. Die graue Fläche links oben repräsentiert Ihr Control. Ihr Control umschließt das UserControl-Objekt und alle konstituierenden Controls. Die zweite graue Fläche enthält die Container-seitigen Objekte. Das Ambient-Objekt und das Extender-Objekt werden von Visual Basic zur Verfügung gestellt und ermöglichen den Zugriff auf Informationen über den Container.

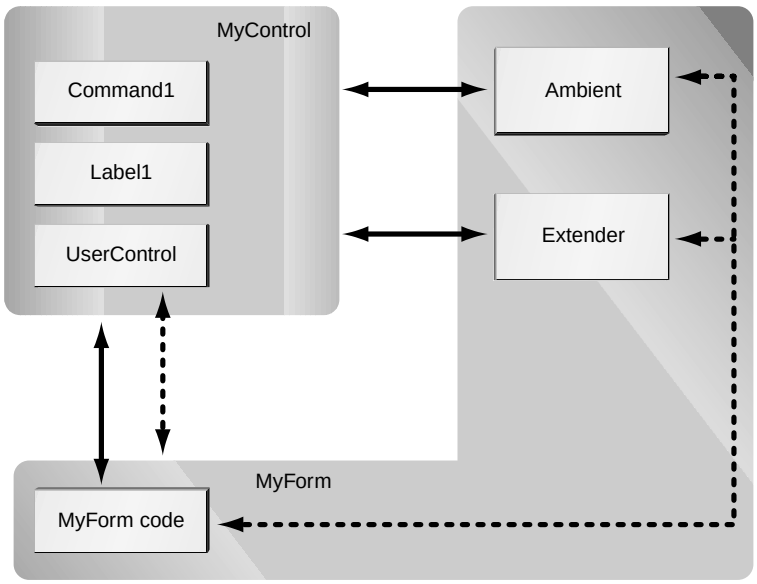

Abb. 16.2: Objekt-Beziehungen in einem ActiveX-Control

16.3 Die ~~drei~~ vier Modelle der Control-Entwicklung

Visual Basic unterstützt vier Modelle, nach denen ActiveX-Controls angelegt werden können. Ja, ich bin mir bewußt, daß Microsoft lediglich von drei Modellen spricht. Doch es gibt ein viertes, sehr wichtiges Modell. Es ist ziemlich komplex und erfordert einiges an Erfahrung, so daß ich deswegen Microsoft keinen Vorwurf machen kann, daß man es übergangen hat.

Die Entscheidung für ein Modell ist eine der grundlegendsten Entscheidungen beim Entwurf eines Controls. In diesem Abschnitt werden wir uns die verschiedenen Ansätze ansehen, samt deren Vor- und Nachteilen. Dazu werden wir noch einmal auf einige der fundamentalen Konzepte eingehen, die wir in diesem Kapitel bereits angeschnitten haben.

16.3.1 Ein vorhandenen Control erweitern

Nehmen wir einmal an, Sie verwenden bereits seit Jahren CommandButtons und stellen plötzlich fest, daß irgend etwas fehlt. Nur ein einziges Feature fehlt diesen CommandButtons, und wenn dieses vorhanden wäre, wären Sie auch rundherum mit Visual Basic zufrieden. Das einzige, was Sie vermissen, ist ein `Click`-Ereignis, das zugleich in seinen Parametern die Maus-Koordinaten des Klicks mitliefert.

Als erfahrener Visual-Basic-Programmierer wissen Sie, daß Sie diese über das `MousUp`-Ereignis erhalten können. Sie wissen jedoch auch, daß das nicht für jedermann, der sich ein modifiziertes `Click`-Ereignis wünschen würde, eine einfache und klare Angelegenheit ist. Doch letztlich ist das egal. Der Punkt ist, daß wohl nahezu jeder Visual-Basic-Programmierer bei dem einen oder anderen Control gedacht hat: »Wenn es doch bloß dieses eine Feature mehr hätte ...«

Im ActiveX-Control-Designer können Sie andere Controls (sogenannte »Konstituierende Controls«) auf Ihrem ActiveX-Control plazieren. Sie können ebenso öffentliche Eigenschaften und Ereignisse definieren, über die die Eigenschaften und Ereignisse eines konstituierenden Controls erreichbar werden.

Die Projekt-Gruppe `BtnTest` im Ordner zu Kapitel 16 auf der Buch-CD demonstriert dies an einem einfachen Beispiel. Sie enthält das Standard-EXE-Projekt `BtnTest` zum Testen des Controls aus dem ActiveX-Control-Projekt `ch16Button`. Dieses Control entstand, indem ein `CommandButton`-Control auf dem `UserControl`-Designer plaziert wurde und indem mit dem ActiveX-Schnittstellen-Assistenten Eigenschaften und Ereignisse Ihrer Control-Klasse mit denen des CommandButton-Controls verknüpft wurden.

Sie kennen bereits die Art von Code, bei dem Eigenschaften und Ereignisse verknüpft werden. Hier sehen Sie beispielsweise den Code zur Verknüpfung der `Font`-Eigenschaft:

```
Public Property Get Font() As Font
   Set Font = Command1.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
   Set Command1.Font = New_Font
   PropertyChanged "Font"
End Property
```

Und dies hier ist ein Beispiel für eine Ereignis-Verknüpfung:

```
Private Sub Command1_KeyPress(KeyAscii As Integer)
   RaiseEvent KeyPress(KeyAscii)
End Sub
```

So wird das Click-Ereignis Ihres Objekts modifiziert:

```
Event Click(ByVal X As Single, ByVal Y As Single)
Event MouseDown(Button As Integer, _
 Shift As Integer, X As Single, Y As Single)
Private mousex As Single, mousey As Single
Event MouseMove(Button As Integer, _
 Shift As Integer, X As Single, Y As Single)
Private Sub Command1_MouseDown(Button As Integer, _
 Shift As Integer, X As Single, Y As Single)
   mousex = X: mousey = Y
   RaiseEvent MouseDown(Button, Shift, X, Y)
End Sub

Private Sub Command1_MouseMove(Button As Integer, Shift As Inte-
ger, X As Single, Y As Single)
   mousex = X: mousey = Y
   RaiseEvent MouseMove(Button, Shift, X, Y)
End Sub

Private Sub Command1_Click()
   RaiseEvent Click(mousex, mousey)
End Sub
```

Die Position des Mauszeigers wird in den Ereignis-Prozeduren `Command1_MouseDown` und `Command1_MouseMove` in zwei privaten Variablen zwischengespeichert, damit die Werte als Parameter mit dem Click-Ereignis übergeben werden können.

Ebenso ist wichtig, daß das `CommandButton`-Control die gesamte Fläche Ihres Controls ausfüllt. Dies wird im Ereignis `UserControl_Resize` gewährleistet:

```
Private Sub UserControl_Resize()
   ' Control füllt das Fenster aus
```

```
    With UserControl
       Command1.Move 0, 0, .ScaleWidth, .ScaleHeight
    End Width
End Sub
```

Dieses Beispiel zeigt ein paar Punkte auf, die Sie bedenken sollten. Zunächst sollte man eigentlich keine Standard-Ereignisse wie das `Click`-Ereignis umdefinieren. Dies könnte Anwender Ihres Controls irritieren. Legen Sie besser einen neuen Ereignis-Namen fest. Das gilt gleichermaßen auch für Eigenschaften.

Dieses Control-Beispiel umfaßt keine vollständige Implementierung eines `CommandButtons`. Es übernimmt nur einige der Eigenschaften und Ereignisse, um den Lösungsweg für diese Aufgabe aufzuzeigen.

Bei diesem Control sind die meisten seiner öffentlichen Eigenschaften und Ereignisse mit denen des `CommandButton`-Controls verknüpft. Es werden keine Eigenschaften und Ereignisse des `UserControl`-Objekts verwendet. Warum nicht?

Wenn ein konstituierendes Control angeklickt wird, wird dessen `Click`-Ereignis ausgelöst, hier das des `CommandButton`-Controls. Das `Click`-Ereignis des `UserControl`-Objekts wird nur ausgelöst, wenn seine »nackte« Oberfläche angeklickt wird. Dies können Sie am Beispiel des oben beschriebenen Controls `Ch16CtlA` sehen, bei dem zwischen Ereignissen zweierlei Ursprungs unterschieden wurde. In diesem Beispiel hier benötigen wird das `Click`-Ereignis des `UserControl`-Objekts nicht, da das `CommandButton`-Control dessen Fläche vollständig abdeckt – das `Click`-Ereignis des `UserControl`-Objekts kann daher niemals ausgelöst werden.

Das gleiche gilt für Eigenschaften des `UserControl`-Objekts, wie etwa die `Font`-Eigenschaft. Die `Font`-Eigenschaft tritt in diesem Control-Beispiel dreimal auf: beim `CommandButton`-Control, beim `UserControl`-Objekt und in Ihrer Control-Klasse:

- Die `Font`-Eigenschaft des `CommandButton`-Controls legt das Erscheinungsbild der Beschriftung der Schaltfläche fest.
- Die `Font`-Eigenschaft des `UserControl`-Objekts würde das Erscheinungsbild von direkt auf dessen Oberfläche gezeichneten Texts festlegen. Doch da hier in diesem Beispiel nichts auf die Oberfläche des `UserControl`-Objekts gezeichnet wird, hat diese Eigenschaft keinerlei Effekt.
- Die `Font`-Eigenschaft Ihrer Control-Klasse ist die öffentliche Eigenschaft, auf die ein Verwender Ihres Controls zugreifen kann. Er hat auf die beiden anderen `Font`-Eigenschaften keinen Zugriff, solange Sie ihm diesen nicht ausdrücklich gewähren.

Es ist leicht ersichtlich, welche der `Font`-Eigenschaften ein Verwender Ihres Controls zu sehen bekommt. Doch welche sehen Sie als Autor innerhalb des Control-Projekts?

Ändern Sie versuchsweise den Code in der Ereignis-Prozedur `Command1_KeyPress` wie folgt:

```
Private Sub Command1_KeyPress(KeyAscii As Integer)
   RaiseEvent KeyPress(KeyAscii)
   If KeyAscii = Asc("A") Then
      Debug.Print Font.Name
   End If
   If KeyAscii = Asc("B") Then
      Debug.Print UserControl.Font.Name
   End If
End Sub
```

Ändern Sie im Test-Programm den `Font`-Namen der Design-Zeit-Instanz Ihres Controls über das Visual-Basic-Eigenschaften-Fenster – (doppel)klicken Sie dazu dort auf die Eigenschaft, um den Font-Dialog zu öffnen. Wenn Sie das Programm dann starten und die Tasten »A« oder »B« drücken, werden Sie sehen, daß die `Font`-Eigenschaft, auf die Sie innerhalb Ihres Control-Programms zugreifen, diejenige ist, die Sie selbst für das Control definiert haben und die an die `Font`-Eigenschaft des `CommandButton`-Controls delegiert wird.

Wenn Sie keine öffentliche `Font`-Eigenschaft Ihrer Control-Klasse definieren, greifen Sie innerhalb der Control-Klasse auf die `Font`-Eigenschaft des `UserControl`-Objekts zu, da dieses das Standard-Objekt eines Controls darstellt.

Wie Sie sehen, sind die Gültigkeitsbereichsregeln bei ActiveX-Controls von großer Bedeutung, da bei mehreren verwendeten Objekten gleichnamige Eigenschaften auftreten. Und mit diesen Betrachtungen über die Font-Eigenschaft haben Sie lediglich erst die Spitze des Eisbergs gesehen. Wir werden in Kapitel 19, »Wunderbare Welt der Eigenschaften«, noch weiter darauf eingehen.

Vor- und Nachteile der Erweiterung vorhandener Controls

Der größte Vorteil der Erweiterung vorhandener Controls ist deren Einfachheit. Der ActiveX-Schnittstellen-Assistent kann Ihnen ein großes Stück Arbeit bei der Verknüpfung der öffentlichen Eigenschaften, Methoden und Ereignisse mit denen von konstituierenden Controls und des `UserControl`-Objekts abnehmen.

Dieser Ansatz hat aber auch Nachteile:

- Da das Verhalten und die Darstellung eines konstituierenden Controls bereits festgelegt ist, bleiben Ihnen nur wenige Möglichkeiten zur Modifizierung des Controls. Über SubClassing-Techniken können Sie zwar mehr Einfluß auf das Verhalten eines konstituierenden Controls nehmen, doch sind dies recht fortgeschrittene Techniken, die ein gutes Verständnis des Win32-APIs und viele Experimente erfordern, um herauszufinden, wie das Control auf die verschiedenen Windows-Nachrichten reagiert.

- Wie Sie bereits gelesen haben, befinden sich konstituierende Controls immer im Laufzeit-Zustand, wenn Ihr Control-Objekt aktiv ist. Daher lassen sich Design-Zeit-Eigenschaften des konstituierenden Controls selbst dann nicht mehr beeinflussen oder ändern, wenn sich Ihr Control-Objekt im Design-Zeit-Zustand befindet.
- Wenn Sie andere als die in Visual Basic eingebauten oder von Microsoft mit Visual Basic ausgelieferten Controls als konstituierende Controls verwenden wollen, werden Sie von Lizenz- und Copyright-Problemen betroffen. Damit Ihr neues Control ordnungsgemäß in einer Visual-Basic-Entwicklungsumgebung verwendet werden kann, müssen alle darauf befindlichen konstituierenden korrekt lizenziert sein. Darauf werden wir in Kapitel 26, »Lizenzierung und Vertrieb«, weiter eingehen.

16.3.2 Control aus konstituierenden Controls zusammensetzen

Dieses Modell wird von Microsoft als eines der drei möglichen Modelle beschrieben. Eigentlich ist es ein der einfachen Erweiterung übergeordnetes Modell. Die Unterschiede lauten jedoch:

- Anstatt die öffentlichen Eigenschaften, Methoden und Ereignisse Ihrer Control-Klasse mit einem einzigen konstituierenden Control zu verknüpfen, werden sie mit den verschiedenen entsprechenden Elementen einiger oder aller konstituierenden Controls verknüpft. Es ist schließlich auch möglich, ein und dieselbe Eigenschaft mit mehr als einem konstituierenden Control zu verknüpfen.
- Hierbei wird es eher vorkommen, daß Sie auf Eigenschaften des `UserControl`-Objekts zugreifen werden.

Dieses Modell wird in der `BtnTest`-Projekt-Gruppe anhand des Controls `Ch16DualButton` gezeigt. Dieses Control enthält zwei CommandButtons und zeigt eine Reihe verschiedener Techniken der Verknüpfung von Eigenschaften auf.

Sie können mehrere Ereignisse mit einem einzigen verknüpfen. So werden hier die `Click`-Ereignisse der konstituierenden CommandButtons und des `UserControl`-Objekts mit einem einzigen mit Parametern versehenen `Click`-Ereignis wie folgt verknüpft:

```
Event Click(ByVal ButtonNum%)
Private Sub Command1_Click()
   RaiseEvent Click(1)
End Sub

Private Sub Command2_Click()
   RaiseEvent Click(2)
End Sub
```

```
Private Sub UserControl_Click()
   RaiseEvent Click(0)
End Sub
```

Starten Sie das Projekt und klicken Sie auf beide Schaltflächen und auf den Raum dazwischen. Eine Nachricht im Direkt-Fenster informiert Sie darüber, welches Objekt angeklickt worden ist.

Sie können auch eine einzelne Eigenschaft Ihrer Control-Klasse mit Eigenschaften mehrerer Controls verknüpfen:

```
Public Property Get Font() As Font
   Set Font = UserControl.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
   Set Command1.Font = New_Font
   Set UserControl.Font = New_Font
   Set Command2.Font = New_Font
   PropertyChanged "Font"
End Property
```

In diesem speziellen Fall ist die Verknüpfung der öffentlichen `Font`-Eigenschaft mit der des `UserControl`-Objekts ein wenig übertrieben, da auf die Oberfläche des `UserControl`-Objekts selbst kein Text gezeichnet wird. Und warum geben wir in der `Property Get`-Prozedur die `Font`-Eigenschaft des `UserControl`-Objekts zurück? Eigentlich ist es egal. Alle Objekte referenzieren dasselbe `Font`-Objekt, so daß Sie dieses von überall herholen können, und es wird funktionieren.

Dieses Modell hat genau die gleichen Vor- und Nachteile wie der Ansatz der einfachen Erweiterung. Gleichfalls gilt hier, daß das `UserControl`-Objekt selbst nicht den Fokus erhalten kann – nur die konstituierenden Controls können ihn erhalten.

16.3.3 Selbstzeichnende Controls

Selbstzeichnende Controls stellen einen der interessantesten Ansätze der Control-Entwicklung dar, auch wenn Sie um ein vielfaches komplexer sind. Bei diesem Ansatz werden Sie in erster Linie mit den Eigenschaften, Methoden und Ereignissen des `UserControl`-Objekts arbeiten. Sie können bei Bedarf unsichtbare konstituierende Controls verwenden. Doch sobald Sie ein sichtbares konstituierendes Control hinzufügen, ändert sich das Verhalten Ihres Control-Objekts dahingehend, daß es den Fokus nicht mehr erhalten kann.

Die Beispiel-Anwendung `ClkTest` enthält ein privates Control namens `ClkTest.ctl`, dessen Code Sie als Beispiel für ein selbstzeichnendes Control in Listing 16.1 sehen. Das Control erscheint beim Anlegen zunächst als grünes Rechteck und ändert seine Farbe in Blau und Rot, wenn es den Eingabe-Fokus

erhält bzw. wieder verliert. Wenn Sie das Projekt starten, sehen Sie das Control auf dem Test-Formular `clkForm`. Springen Sie mit der Tabulator-Taste zu dem Control und wieder von diesem weg, und klicken Sie es an – Sie werden sehen, daß das `Click`-Ereignis funktioniert.

```
Option Explicit

Event Click()

Private Sub UserControl_Click()
   RaiseEvent Click
End Sub

Private Sub UserControl_GotFocus()
   UserControl.BackColor = vbBlue
End Sub

Private Sub UserControl_Initialize()
   UserControl.BackColor = vbGreen
End Sub

Private Sub UserControl_LostFocus()
   UserControl.BackColor = vbRed
End Sub
```

Abb. 16.1: Ein einfaches selbstzeichnendes Control

Sie werden das Control in der Regel im `UserControl_Paint`-Ereignis zeichnen. Über das Zeichnen von Controls werden Sie in Kapitel 17 mehr erfahren, in dem es um die verschiedenen Methoden und Eigenschaften des `UserControl`-Objekts gehen wird.

Vor- und Nachteile selbstzeichnender Controls

Der größte Vorteil selbstzeichnender Controls liegt in deren Flexibilität. Da Sie das Verhalten und das Erscheinungsbild des Controls während seiner gesamten Lebensdauer selbst in der Hand haben, gibt es nur wenige Grenzen für die Aufgabenstellungen für das Control.

Der größte Nachteil selbstzeichnender Controls ergibt sich aus folgender Tatsache: Sie *können* nicht nur Verhalten und Erscheinungsbild des Controls selbst bestimmen, sondern Sie *müssen* es sogar. Einige der Techniken, die Sie für ein selbstzeichnendes Control verwenden können, werden im folgenden Kapitel behandelt. Doch da es unzählige Möglichkeiten gibt, wäre es vermessen zu behaupten, daß ich Ihnen mehr als nur eine solide Einführung darin geben könnte.

Denken Sie daran, daß Sie bei einem selbstzeichnenden Control nicht auf die Visual-Basic-eigenen Fähigkeiten beschränkt sind. Sie haben vielmehr Zugriff

auf alle Win32-API-Funktionen. Und dieses bietet einige leistungsfähige Grafik- und Textausgabe-Funktionen, die weit über das hinausgehen, was Visual Basic zu bieten hat (und obendrein auch noch schneller sind). Lesen Sie dazu mein Buch *Visual Basic Programmer's Guide to the Win32 API.*

Schauen Sie sich auch nach kommerziellen Angeboten von Controls um, die mit Quell-Code ausgeliefert werden. Damit haben Sie dann nicht nur hochwertige Controls zur Verfügung, die Sie in Ihren Anwendungen einsetzen können, sondern Sie können auch viel über fortgeschrittene Control-Entwicklung lernen.

16.3.4 Eigene Fenster erstellen

Das Windows-Betriebssystem definiert Standard-Klassen für Fenster wie ListBoxen, TextBoxen, Strukturansicht-Fenster, Standard-Dialog-Fenster u.v.m.

Viele ActiveX-Controls und die in Visual Basic eingebauten Controls sind eigentlich Oberklassen (»Superklassen«) dieser Standard-Fenster. So legt das Visual-Basic-eigene ListBox-Control tatsächlich ein Windows-ListBox-Fenster an. Es fängt dessen Windows-Nachrichten ab und leitet sie in Eigenschaften und Ereignisse des Controls um. Auf diese Weise kann ein verhältnismäßig einfaches Control in den vollen Genuß der Leistung der ins Betriebssystem eingebauten Controls kommen.

Wenn Sie konstituierende Controls verwenden, legen Sie lediglich eine weitere Schicht über diese bereits vorhandene Hierarchie, wie Sie in Abbildung 16.3 sehen können.

Die Schwierigkeit mit diesem Ansatz besteht darin, daß sich Ihr Control nicht über Beschränkungen hinwegsetzen kann, die auf einer tieferen Ebene dieser Hierarchie eventuell bereits eingebaut worden sind. Nicht alle ActiveX-Controls implementieren die volle Funktionalität der zugrundeliegenden Windows-Fenster-Klasse. So bietet die Visual-Basic-ListBox keine Möglichkeit zum Setzen von Tabulatoren, in der Basis-Implementierung des Betriebssystems ist diese Möglichkeit jedoch enthalten. Bei mit Visual Basic erstellten ActiveX-Controls müssen Sie außerdem damit leben, daß Design-Zeit-Eigenschaften zur Laufzeit nicht mehr geändert werden können.

Das liegt daran, daß etwa bei einem Fenster der `LISTBOX`-Klasse verschiedene Merkmale, wie etwa die Möglichkeit der Mehrfachauswahl, bereits beim Anlegen des Fensters festgelegt werden müssen. Um ein Merkmal wie dieses zu ändern, muß das Fenster zuerst zerstört und erneut angelegt werden. Mit Visual Basic erstellte Controls können nicht nach Belieben ein konstituierendes Control zerstören und neu anlegen lassen, was zum Ändern eines Merkmals wie eben der Mehrfachauswahl notwendig wäre.

Doch es gibt eine Lösung. Anstatt eine Visual-Basic-ListBox zu verwenden, können Sie auch die Windows-eigene Klasse direkt verwenden und Ihr eigenes Fenster der Klasse `LISTBOX` auf dem `UserControl`-Objekt anlegen. Da Sie das Fenster selbst angelegt haben, können Sie es auch wieder zerstören und neu

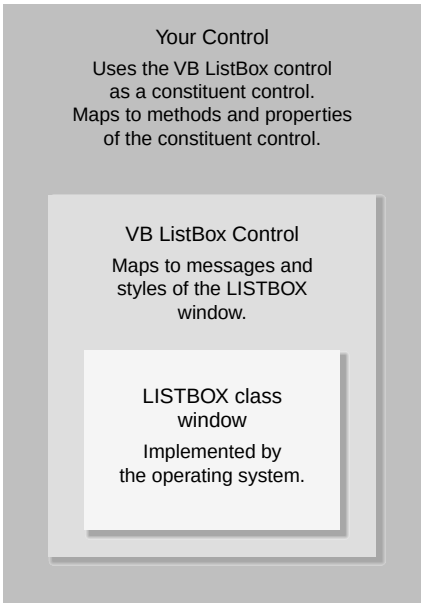

Abb. 16.3: Hierarchie von Controls

erstellen. Dann können Sie jederzeit die ursprünglich fixen Design-Zeit-Merkmale und sogar die Laufzeit des Containers beliebig ändern.

Der Wert dieses Ansatzes liegt darin, daß Sie in den vollen Genuß der Leistungsfähigkeit der Fenster-Klassen kommen, die im Betriebssystem vorhanden sind. Es gibt jedoch auch hierbei wiederum einige Nachteile zu berücksichtigen:

- Dieser Ansatz erfordert ein tiefgehendes Verständnis von Windows-API-Techniken und der Funktionsweise von Fenster-Klassen.
- Der Ansatz kann sehr aufwendig werden, vor allem bei komplexeren Controls.

Eines sollten Sie jedoch nicht aus dem Auge verlieren. Wenn Sie ein Control für Ihren eigenen Bedarf anlegen, werden Sie sicher nicht die gesamte Spannweite der vom Betriebssystem angebotenen Funktionalität in Eigenschaften, Methoden und Ereignisse umzusetzen brauchen. Als kommerzieller Entwickler werden Sie jedoch bestrebt sein, so viele Features des zugrundeliegenden Controls so vielen Programmierern wie möglich zur Verfügung zu stellen. Wenn Sie das Control nur für den eigenen Gebrauch entwickeln, können Sie sich auf die Funktionalität konzentrieren, die Sie wirklich brauchen, und keine Zeit darauf verschwenden, alles übrige zu implementieren und zu testen. Daher kann dieser Ansatz unter Umständen weitaus praktikabler sein, als es auf den ersten Blick scheinen mag. Diesen Ansatz werden wir weiter in Kapitel 22 behandeln.

Nun haben Sie die Grundlagen der ActiveX-Control-Entwicklung mit Visual Basic kennengelernt, so daß es nun an der Zeit ist, tiefer in die Materie einzutauchen und sich mit dem zentralen Herz der Maschinerie zu befassen: mit dem `UserControl`-Objekt.