Kapitel 28

Eine Frage des Status

Bei dem »Kommentar«-Ansatz, nach dem ich in diesem Buch vorgegangen bin, fällt auf, daß die Anzahl der Seiten, die ich den einzelnen Themen gewidmet habe, im allgemeinen umgekehrt proportional zu der Informationsmenge ist, die Microsoft in seiner Visual-Basic-Dokumentation anbietet. Themen, die Microsoft ausführlich behandelt, erwähne ich normalerweise nur kurz. Themen, die kaum irgendwo erwähnt werden, nehmen bei mir ganze Kapitel ein.

Dieses Kapitel wurde durch die Kommentare in der Microsoft-Dokumentation verursacht, die Performance und Skalierbarkeit Ihrer IIS-Anwendung und der Microsoft-Transaction-Server-Komponenten könne verbessert werden, indem sie statuslos gemacht werden. Wenn man die Dokumentation von Microsoft liest, gewinnt man leicht den Eindruck, die statuslose Programmierung sei die allein seligmachende Methode der Zukunft, und jeder, der keine statuslosen Komponenten entwickelt, ist ineffizient und faul und sollte sofort gefeuert werden, egal welchen Job er hat.

Damit fragen wir normal Sterblichen natürlich, was es eigentlich bedeutet, eine Komponente oder ein Programm statuslos zu machen, welchen Einfluß das auf unsere Applikationen hat, ob die statuslose Programmierung wirklich die Zukunft darstellt, oder ob es nur eine Modeerscheinung ist, die im teuren Upgrade im nächsten Jahr völlig verworfen wird?

In diesem Kapitel versuche ich, Licht in dieses Thema zu bringen.

28.1 Was ist der Status?

Auch bei der Erklärung des Statuskonzepts beginnt man am besten mit einem ganz einfachen Beispiel. Angenommen, wir haben eine Applikation, mit deren Hilfe der Benutzer etwas bestellen kann. Um den Typ der Komponente wollen wir uns hier noch nicht kümmern. Er wird erst viel später wichtig. Eigentlich ist er ja auch später nicht wichtig, aber das wird lange Zeit nicht klar sein.

In dieser Bestell-Applikation durchläuft das Programm fünf Schritte:

1. Anmeldung des Benutzers im Bestellsystem.
2. Eingabe des zu bestellenden Artikels.
3. Eingabe der Artikelmenge.
4. Bestellung ausführen.
5. Abmeldung des Benutzers aus dem Programm.

Angenommen, wir erzeugen eine Komponente, die diese Operationen ausführt. Diese Komponente könnte fünf Methoden beinhalten, die einem Kunden die Ausführung dieser Operationen erlauben.

Es könnte eine Funktion geben, die den Benutzer am System anmeldet, und eine, die ihn abmeldet. Es könnte Funktionen geben, die den Artikel und die Menge abfragen, und eine weitere, die die Bestellung vornimmt.

Nicht alle Funktionen können zu jedem Zeitpunkt ausgeführt werden. Beispielsweise dürfen sie dem Kunden nicht erlauben, eine Bestellung auszuführen, wenn er nicht angemeldet ist. Er darf keine Bestellung ausführen, wenn er keinen Artikel und keine Menge eingegeben hat. Er kann sich nicht aus dem System abmelden, wenn er nicht angemeldet ist. Man könnte sagen, die erfolgreiche Ausführung der Funktionen zu beliebigen Zeitpunkten ist vom jeweiligen Status der Komponente abhängig.

Beachten Sie die Formulierung Status der Komponente.

Die Komponente muß offensichtlich beobachten, ob ein Benutzer angemeldet ist, um feststellen zu können, ob ihm erlaubt werden soll, einen Artikel auszuwählen. Sie muß beobachten, ob ein Artikel und eine Menge angegeben sind, um zu entscheiden, ob die Bestellung ausgeführt werden darf.

Die Komponente befindet sich zu jedem Zeitpunkt in einem bestimmten Status, abhängig davon, welche Funktionen zuvor dafür aufgerufen wurden. Nachdem beispielsweise ein Benutzer angemeldet ist, könnte man sagen, der Status der Komponente ist »Angemeldet«.

Viele Programmierer würden diese Komponente einfach mit ein paar `If...Then`-Anweisungen am Anfang jeder Funktion implementieren, um zu prüfen, ob sich die Komponente in einem gültigen Status für die jeweilige Funktion befindet. Für eine komplexe Komponente kann es jedoch schwierig sein, zu erkennen, ob jede mögliche Situation überprüft wurde.

Aus diesem Grund ist es häufig sinnvoll, ein spezielles Diagramm anzulegen, ein sogenanntes »Statusdiagramm«, um die Funktionalität der Applikation zu beschreiben. Man spricht in dem Zusammenhang auch von einem »Endlichen Automaten«, der eine Applikation oder Komponente beschreibt, die eine endliche Anzahl von Statuszuständen annehmen kann, wie es bei unserem Beispiel der Fall ist.

Abbildung 28.1 zeigt das Diagramm für den endlichen Automaten, der unsere Bestell-Komponente darstellt. Die Kreise enthalten die jeweiligen Statuszustände. Die Pfeile, die aus den Kreisen kommen, stellen die Operationen dar, die für diesen Status erlaubt sind. Zunächst ist die Komponente `idle` oder im Ausgangszustand. Kein Benutzer ist angemeldet, und die einzige zulässige Funktion ist die Anmeldung (Login). Nachdem der Benutzer angemeldet ist, kann er sich wieder abmelden (Logout), oder einen Artikel oder eine Menge angeben, aber noch keine Bestellung ausführen. Nachdem der Benutzer sowohl einen Artikel als auch eine Menge angegeben hat, ist er bereit für den Kauf. Der Benutzer kann den Artikel oder die Menge ändern und weiterhin bereit sein, die Bestellung auszuführen. Nach der Bestellung werden der aktuelle Artikel und die Menge gelöscht und die Komponente geht wieder in den Status Angemeldet (Logged in) über.

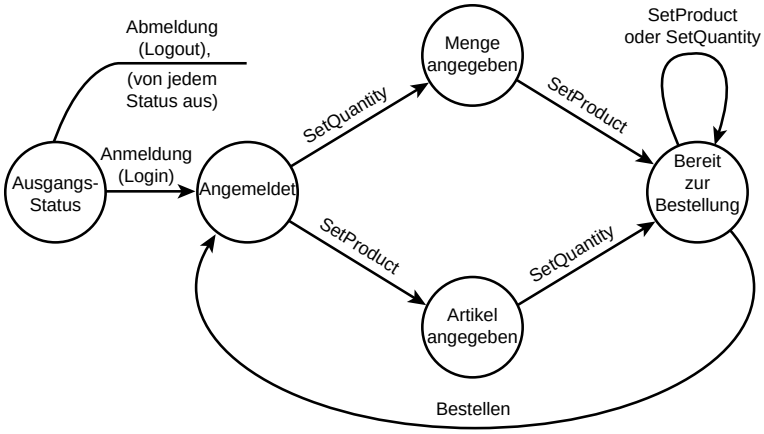

Abb. 28.1: Diagramm eines endlichen Automaten, der ein einfaches Bestellsystem darstellt

Ein Automatendiagramm ist ein äußerst leistungsfähiges Werkzeug für den Programmierer, der die Komponente entwickelt, um den endlichen Automaten zu implementieren. Es drückt ganz deutlich aus, welche Funktionen für jeden Status zulässig sind. Das Klassenmodul `BuyIt1.cls` implementiert diesen Automaten. Es verwendet vier Elementvariablen, in denen die vom Automaten verwendete Information verwaltet wird:

```
Private m_LoggedIn As Boolean   ' True, wenn der Benutzer angemeldet ist
Private m_ItemToBuy As String   ' Der zu bestellende Artikel
Private m_ItemQuantity As Long  ' Die Menge
Private m_CustomerName As String   ' Der Name des aktuellen Benutzers
```

Die ersten drei dieser Variablen sind ausreichend, um den Status der Komponente festzulegen. Die Variable `mLoggedIn` teilt Ihnen mit, ob ein Benutzer angemeldet ist. Der String `m_ItemToBuy` enthält den Namen des zu bestellenden Artikels, oder ist leer, wenn kein Artikel angegeben wurde. Das Feld `m_ItemQuantity` enthält die Artikelmenge, und ist Null, wenn keine Menge angegeben wurde.

Tabelle 28.1 zeigt die logischen Bedingungen für diese Variablen, die genutzt werden, um den Status der Komponente festzustellen.

Ausgangsstatus	`Not m_LoggedIn`
Angemeldet	`m_LoggedIn`
Artikel angegeben, aber keine Menge	`m_LoggedIn And`
	`m_ItemToBuy<>"" And`
	`m_ItemQuantity = 0`

Tab. 28.1: Ermittlung des Status der Komponente

Menge angegeben, aber kein Artikel	`m_LoggedIn And`
	`m_ItemToBuy="" And`
	`m_ItemQuantity>0`

Bereit für die Bestellung	`m_LoggedIn And`
	`m_ItemToBuy<>"" And`
	`m_ItemQuantity>0`

Tab. 28.1: Ermittlung des Status der Komponente

Sie könnten vielleicht auch ohne die Auswertung von `m_LoggedIn` zu den drei Ergebnissen gelangen, wenn Ihre Komponente voraussetzt, daß `m_ItemToBuy` leer ist und `m_ItemQuantity` gleich Null, solange der Benutzer abgemeldet ist.

Listing 28.1 zeigt die Implementierung der gesamten Klasse. Die Komponente fügt Eigenschaften ein, um die Werte der internen Elementvariablen zu lesen, obwohl sie nur durch die Methoden der Komponente gesetzt werden können. Die Komponente verwendet die API-Konvention für die Fehlerüberprüfung, und gibt -1 zurück, wenn ein Fehler auftritt, und Null, wenn die Ausführung erfolgreich war.

```
' Einfache Klasse für die Bestellung
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private m_LoggedIn As Boolean ' True, wenn der Benutzer angemeldet ist
Private m_ItemToBuy As String ' Der zu bestellende Artikel
Private m_ItemQuantity As Long   ' Die zu bestellende Menge
Private m_CustomerName As String ' Der Name des aktuellen Benutzers

Public Property Get LoggedIn() As Boolean
   LoggedIn = m_LoggedIn
End Property

Public Property Get ItemToBuy() As String
   ItemToBuy = m_ItemToBuy
End Property

Public Property Get ItemQuantity() As Long
   ItemQuantity = m_ItemQuantity
End Property

Public Property Get CustomerName() As String
   CustomerName = m_CustomerName
End Property
```

```
Public Function Login(ByVal name As String) As Long
   If m_LoggedIn Or name = "" Then
      Login = -1
      Exit Function
   End If
   m_LoggedIn = True
   m_CustomerName = name
End Function

Public Function Logout() As Long
   If Not m_LoggedIn Then
      Logout = -1
      Exit Function
   End If
   m_LoggedIn = False
   m_CustomerName = ""
   m_ItemToBuy = ""
   m_ItemQuantity = 0
End Function

Public Function SetProduct(ByVal ProductName As String) As Long
   If Not m_LoggedIn Or ProductName = "" Then
      SetProduct = -1
      Exit Function
   End If
   m_ItemToBuy = ProductName
End Function

Public Function SetQuantity(ByVal Quantity As Long) As Long
   If Not m_LoggedIn Or Quantity <= 0 Then
      SetQuantity = -1
      Exit Function
   End If
   m_ItemQuantity = Quantity
End Function

Public Function Buy() As Long
   If Not m_LoggedIn Or m_ItemQuantity <= 0 Or m_ItemToBuy = "" Then
      Buy = -1
      Exit Function
   End If
   ' Hierhin kommt irgendeine Bestell-Funktionalität
   m_ItemQuantity = 0
   m_ItemToBuy = ""
End Function
```

Listing. 28.1: Eine einfache Klasse für die Bestellung

Nachdem Sie nun einen endlichen Automaten und seine Implementierung kennengelernt haben, möchte ich, daß Sie einen Moment lang überlegen, was es bedeutet, einen Automaten zu implementieren. Um den Automaten zu implementieren, mußte die Komponente zwei Dinge tun:

- Sie mußte Daten speichern, um den Status der Komponente zwischen den Methodenaufrufen zu verwalten. In unserem Beispiel legt die Komponente die Daten in den Klassenelementvariablen ab.
- Sie mußte die Abfolge steuern, indem sie die zulässigen Übergänge zwischen den verschiedenen Statuszuständen definierte.

Das ist ein sehr wichtiges Konzept. Die meisten Leute und Dokumentationen haben die Tendenz, Statuszustände so zu beschreiben, als handle es sich dabei um ein einziges Ding. Die Tatsache, daß die Statusverwaltung eigentlich aus zwei verschiedenen Problemen besteht (Aufbewahrung von Daten und Steuerung der Abfolge), hat weitreichende Einflüsse auf die Weise, wie Automaten mit Technologien wie IIS-Anwendungen oder Microsoft Transaction Server aufgebaut werden.

28.1.1 Implementierung eines einfachen Clients

In diesem Beispiel wurde der Automat als Komponente implementiert. Wenn Sie als Entwickler in einer Firma arbeiten, wo man auf aktuelle Schlagwörter steht, könnten Sie sich diese Komponente auch als Middle-Tier in einer Three-Tier-Anwendung vorstellen, die die Geschäftsregeln implementiert. Die unterste Ebene wäre die Datenbank und der eigentliche Bestellmechanismus (von dem Sie einfach annehmen müssen, daß er irgendwo ist, weil diese Beispielkomponente offensichtlich nichts Entsprechendes tut). Die oberste Ebene ist die Client-Applikation, die noch nicht geschrieben wurde.

Wenn Sie nicht in einer Firma arbeiten, oder den Schlagwörtern der Branche skeptisch gegenüberstehen, könnten Sie sich das Ganze auch als bodenständigen objektorientierten Ansatz vorstellen, wo die Komponente die Benutzeroberfläche von dem zugrundeliegenden Bestellmechanismus trennt.

Es gibt einige gute Gründe, den Automaten auf diese Weise zu implementieren.

- Sie können die Komponente zusammen mit den Komponenten für die eigentliche Bestellung auf einem Remote-System ablegen und reduzieren damit die Netzwerkbandbreite.
- Sie können Änderungen an der Implementation der Komponente vornehmen, ohne die Benutzeroberfläche ändern zu müssen, solange die COM-Schnittstelle dieselbe bleibt.
- Sie können die zugrundeliegende Datenbank und die Bestell-Komponenten ändern, ohne die Benutzeroberfläche ändern zu müssen.

Ich erwähne dies, um zu erklären, warum ich das Statuskonzept anhand einer Komponente erkläre, statt einen Automaten in einer Applikation zu entwickeln. Es ist leicht verständlich, wie diese Komponente einen Automaten mit fünf möglichen Statuszuständen implementiert. Nun wollen wir einige mögliche Entwürfe für Clients betrachten, die diese Komponente nutzen.

Der offensichtlichste Ansatz. Das Beispielprojekt `wiz1.vbp` demonstriert einen offensichtlichen Ansatz, wobei die Benutzeroberfläche den Automaten relativ genau nachbildet. Das Projekt setzt sich aus vier Formularen zusammen.

Das Anmeldeformular, `frmWiz1.frm`, enthält ein Textfeld, in das der Benutzer einen Kundennamen eingibt. Es besitzt eine einzige Schaltfläche, die die Anmeldeoperation ausführt, wie in Abbildung 28.2 gezeigt.

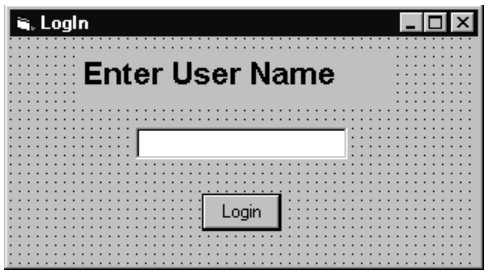

Abb. 28.2: Das Anmeldeformular

Das Formular verwendet den folgenden Code:

```
' Beispielprojekt wiz1
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub cmdLogin_Click()
   Dim res&
   res = BuyItObject.Login(txtLogin.Text)
   If res = -1 Then
      MsgBox "Error Occurred"
   Else
      frmWiz1B.Show
      Unload Me
   End If
End Sub

Private Sub Form_Load()
   Set BuyItObject = New BuyIt1
End Sub
```

Dies ist das erste Formular der Applikation. Beim Laden erzeugt es die Komponente `BuyIt1` und speichert sie in der globalen Variablen `BuyItObject` für die Applikation. Diese Variable ist in einem Standardmodul definiert, das von allen Formularen gemeinsam genutzt wird.

Die einzige Operation, die der Benutzer ausführen kann, wenn dieses Formular angezeigt wird, ist die Anmeldung. Wenn die Anmeldung erfolgreich ausgeführt wurde, wird das Formular aus dem Speicher entfernt und das Formular für die Artikeleingabe erscheint, wie in Abbildung 28.3 gezeigt. Dort gibt der Benutzer den Namen des Artikels ein.

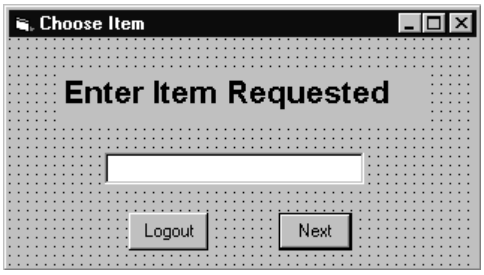

Abb. 28.3: Das Formular für die Artikeleingabe

Dieses Formular verwendet den folgenden Code:

```
' Beispielprojekt wiz1
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub cmdLogout_Click()
   Set BuyItObject = Nothing
   frmWiz1.Show
   Unload Me
End Sub

Private Sub cmdNext_Click()
   Dim res&
   res = BuyItObject.SetProduct(txtItem.Text)
   If res = -1 Then
      MsgBox "Error Occurred"
   Else
      frmWiz1C.Show
      Unload Me
   End If

End Sub
```

```
Private Sub Form_Load()
   txtItem.Text = BuyItObject.ItemToBuy
End Sub
```

Beachten Sie, daß es in diesem Formular zwei Möglichkeiten gibt. Er kann einen Artikel eingeben, dann wird das Formular geschlossen und das Formular für die Menge wird angezeigt. Der Benutzer hat außerdem die Möglichkeit, sich abzumelden, dann wird wieder das Anmeldeformular angezeigt.

Das Formular für die Mengenangabe (Laufzeit) ist in Abbildung 28.4 gezeigt.

Abb. 28.4: Das Formular für die Mengenangabe

```
' Beispielprojekt wiz1
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub cmdBack_Click()
   frmWiz1B.Show
   Unload Me
End Sub

Private Sub cmdLogout_Click()
   Set BuyItObject = Nothing
   frmWiz1.Show
   Unload Me
End Sub

Private Sub cmdNext_Click()
   Dim res&
   res = BuyItObject.SetQuantity(Val(txtQuantity.Text))
   If res = -1 Then
      MsgBox "Error Occurred"
   Else
      frmWiz1D.Show
      Unload Me
   End If
```

```
End Sub

Private Sub Form_Load()
   lblPrompt.Caption = "How many " & BuyItObject.ItemToBuy & " do you
want?"
   txtQuantity.Text = BuyItObject.ItemQuantity
End Sub
```

Dieses Formular bietet dem Benutzer drei Optionen. Der Benutzer kann eine Artikelmenge eingeben und weitergehen zum Formular für die Bestellbestätigung, sich abmelden oder zurückgehen und den eingegebenen Artikel ändern. Beachten Sie, daß beim Laden dieses Formulars und des Formulars für die Artikeleingabe die Komponente `BuyItObject` gelesen wird, um die Textfelder mit den aktuellen Datenwerten zu besetzen. Merken Sie sich das auch für das weitere Projekt.

Im letzten Formular wird die Bestellung bestätigt, wie in Abbildung 28.5 gezeigt. Dieses Formular bietet ebenfalls drei Optionen für den Benutzer. Der Benutzer kann die Bestellung bestätigen, einen Schritt zurückgehen oder sich abmelden.

Abb. 28.5: Das Formular für die Bestätigung der Bestellung

Dieses Formularmodul verwendet den folgenden Code:

```
' Beispielprojekt wiz1
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub cmdBack_Click()
   frmWiz1C.Show
   Unload Me
End Sub

Private Sub cmdBuy_Click()
   Dim res&
   res = BuyItObject.Buy()
   If res = -1 Then
      MsgBox "Error Occurred"
```

```
   Else
      MsgBox "Order confirmed"
      frmWiz1B.Show
      Unload Me
   End If
End Sub

Private Sub cmdLogout_Click()
   Set BuyItObject = Nothing
   frmWiz1.Show
   Unload Me
End Sub


Private Sub Form_Load()
   lblItem.Caption = BuyItObject.ItemQuantity & " x " & BuyItOb-
ject.ItemToBuy
End Sub
```

Nun wollen wir einen Augenblick lang über die Client-Applikation nachdenken. Handelt es sich dabei ebenfalls um einen endlichen Automaten? JA. Er besitzt vier Statuszustände. Er steuert die Übergänge zwischen diesen Statuszuständen. Er implementiert diesen Automaten unter Verwendung derselben Techniken wie in der Komponente. Er steuert die Abfolge und speichert die Statusinformation zwischen den einzelnen Operationen. In diesem Beispiel speichert er die Statusinformation in Form eines `BuyIt1`-Objekts, das in einer globalen Variablen für die Applikation abgelegt wird. Man könnte sagen, die Client-Applikation verwendet das `BuyIt1`-Objekt, um ihren eigenen Automaten zu implementieren.

Ist dieser Automat identisch mit dem, der in der Komponente implementiert wurde?

Nicht ganz. Er implementiert eigentlich eine Untermenge des `BuyIt1`-Automaten, wie in Abbildung 28.6 gezeigt. Dieser Automat beinhaltet keinen Übergang vom Anmelde-Status in den Elementanzahl-Status.

Jetzt müssen wir einen konzeptuellen Sprung ausführen. Offensichtlich ist die Applikation `wiz1.vbp` ein Automat – sie verwaltet den Status zwischen den einzelnen Operationen des Benutzers. Sie verwendet ein `BuyIt1`-Objekt, das ebenfalls ein Automat ist – es verwaltet den Status zwischen den Methodenaufrufen.

Betrachten Sie jedoch einmal das Formular für die Artikeleingabe, `frmWiz1B.frm`. Verwaltet dieses Formular den Status? Nein. Wenn der Benutzer mit der Eingabe auf dem Formular fertig ist, wird es geschlossen. Wenn der Benutzer zu dem Formular zurückkehrt, wird es neu erzeugt und sein Status wird aus einer äußeren Quelle wiederhergestellt (in diesem Beispiel ist das die globale Variable `BuyItObject`). Aber das Formularobjekt selbst speichert keine Informationen – es muß nicht weiterexistieren, wenn der Benutzer keine Operation zur

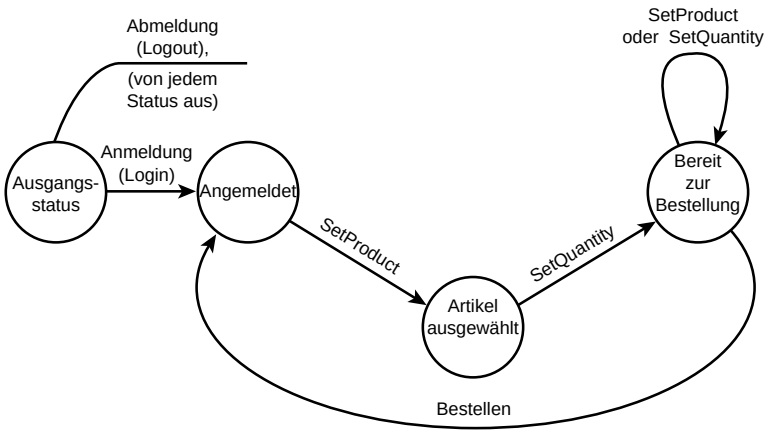

Abb. 28.6: Der Automat, den die Applikation wiz1 implementiert

Artikeleingabe ausführt. Das Formular kann durch die Applikation angezeigt werden, wann immer es benötigt wird, und danach sofort wieder verworfen werden.

Die Visual-Basic-Dokumentation sagt dazu, wenn Sie ein Objekt haben, das nach Bedarf erzeugt und zwischen den Operationen gefahrlos verworfen werden kann, und das intern keine Statusinformationen enthält, dann ist dieses Objekt statuslos.

Sie können Automaten unter Verwendung von statuslosen Objekten implementieren, wenn Sie nur einen Mechanismus bereitstellen, der den Status des Objekts außerhalb dieses Objekts speichert.

Das ist relativ kompliziert, Sie sollten also am besten diesen Abschnitt noch einmal lesen und darüber nachdenken. Bei diesem Beispiel haben wir uns darauf konzentriert, wie die Daten für die Verwaltung des Status gespeichert werden. Nun wollen wir betrachten, wie die Abfolge der Operationen gesteuert wird.

Ein zweiter Client. Das Beispiel `wiz1.vbp` sieht ganz wie ein typisches »Assistenten«-Programm aus, das den Benutzer schrittweise durch einen Prozeß führt. Assistenten sind hervorragend dafür geeignet, eine Abfolge zu steuern, deshalb werden sie auch für die Implementierung von Automaten verwendet.

Das ist alles ganz wunderbar, bis Sie irgendwann ein großes Design-Meeting haben, der Chef kommt herein, sieht Ihre elegant entworfene Applikation und sagt »Assistenten sind irgendwie dumm. Unsere Benutzer erwarten Dialoge auf Registerkarten. Sie brauchen die Flexibilität, schnell von einer Operation zur nächsten zu gelangen. Lösen Sie das irgendwie anders.«

Ihr erster Versuch könnte wie das in Abbildung 28.7 gezeigte Projekt `Tab1.vbp` aussehen.

Abb. 28.7: Das Beispiel Tab1.vbp

Dieses Projekt wird mit einem einzigen Formular implementiert, das vier Rahmen-Steuerelemente enthält. Die Rahmen-Steuerelemente (Frames) dienen als Container für die Textfelder – für jede der Operationen unserer Bestell-Applikation eines. Einer enthält einen Anmeldebildschirm, auf einem kann ein Artikel eingegeben werden, auf einem weiteren die Menge, und auf einem kann der Benutzer die Bestellung bestätigen. Die Abmeldung erfolgt, wenn der Benutzer zurück zum Anmelde-Frame wechselt.

Das Formular erzeugt beim Laden durch das `Form_Load`-Ereignis ein `BuyIt1`-Objekt. Dieses Ereignis zeigt gleichzeitig den ersten Frame an, indem es das Ereignis `TabStrip1_Click` auslöst, wie hier gezeigt:

```
' Beispiel Tab1
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private Sub Form_Load()
   With TabStrip1
      .Top = 0
      .Left = 0
      .Width = frmTab1.ScaleWidth
      .Height = frmTab1.ScaleHeight
   End With
   Set BuyItObject = New BuyIt1
   TabStrip1_Click
End Sub
```

Die beiden Hauptereignisse sind `TabStrip1_BeforeClick` und `TabStrip1_Click`. Das Ereignis `TabStrip1_BeforeClick` tritt auf, wenn der Benutzer auf eine andere als die angezeigte Registerkarte klickt. Das ist für die Applikation das Signal, die Operation für den aktuellen Frame auszuführen. Ist die Operation erfolgreich, erlaubt die Applikation dem Benutzer, zur angeklickten Registerkarte

zu wechseln. Andernfalls wird eine Fehlermeldung angezeigt und der Wechsel wird verhindert. Der Code für dieses Ereignis sieht wie folgt aus:

```
Private Sub TabStrip1_BeforeClick(Cancel As Integer)
   Dim res&
   Select Case TabStrip1.SelectedItem.Index
      Case 1   'logon
               res = BuyItObject.Login(txtLogin.Text)
      Case 2   ' item
               res = BuyItObject.SetProduct(txtItem.Text)
      Case 3   ' quantity
               res = BuyItObject.SetQuantity(Val(txtQuantity.Text))
   End Select
   If res = -1 Then
      MsgBox "Error occurred"
      Cancel = True
   Else
      TabFrame(TabStrip1.SelectedItem.Index).Visible = False
   End If
End Sub
```

Das Ereignis `TabStrip1_Click` tritt auf, wenn eine neue Registerkarte angezeigt wird. Der Code für dieses Ereignis ist unten gezeigt. Die Applikation lädt existierende Informationen aus `BuyItObject`, die dann angezeigt werden. Sie führt eine Abmeldung aus, falls der Anmelde-Frame gewählt wurde.

```
Private Sub TabStrip1_Click()
   TabFrame(TabStrip1.SelectedItem.Index).Visible = True
   TabFrame(TabStrip1.SelectedItem.Index).Move TabStrip1.ClientLeft, _
       TabStrip1.ClientTop, TabStrip1.ClientWidth, TabStrip1.Clien-
tHeight
   Select Case TabStrip1.SelectedItem.Index
      Case 1   'logon
               If BuyItObject.LoggedIn Then BuyItObject.Logout
      Case 2   ' item
               txtItem.Text = BuyItObject.ItemToBuy
      Case 3   ' quantity
               lblPrompt.Caption = "How many " & BuyItObject.ItemToBuy _
                      & " do you want?"
               txtQuantity.Text = BuyItObject.ItemQuantity
      Case 4   ' purchase
               lblItem.Caption = BuyItObject.ItemQuantity & " x " & _
                      BuyItObject.ItemToBuy
   End Select
End Sub
```

Die einzige Schaltfläche ist die BUY-Schaltfläche bei der Bestellbestätigung. Der Code für diese Operation folgt. Wenn die Bestellung erfolgreich war, wird die Registerkarte mit dem Frame für den Artikeleintrag angezeigt.

```
Private Sub cmdBuy_Click()
   Dim res&
   res = BuyItObject.Buy
   If res = -1 Then
      MsgBox "Error occurred"
   Else
      MsgBox "Order confirmed"
      Set TabStrip1.SelectedItem = TabStrip1.Tabs(2)
   End If

End Sub
```

Versuchen Sie, diese Applikation auszuführen. Sie sehen schnell, daß es vielleicht der schlechteste Code ist, den Sie je ausprobiert haben.

Das Problem dabei ist, daß diese Applikation versucht, einen Automaten zu implementieren, für den eine Ablaufsteuerung erforderlich ist, aber eine Benutzeroberfläche verwendet, die als solche keine Ablaufsteuerung unterstützt. Nicht nur das Ergebnis ist schrecklich, sondern das Ganze funktioniert auch nicht besonders gut. Versuchen Sie, einen Benutzernamen einzugeben und sich dann auf die Registerkarte für die Bestellung zu bewegen. Der Frame erlaubt Ihnen zehn Stück Nichts zu bestellen. Selbstverständlich wird ein Fehler aufgeworfen, wenn Sie versuchen, die Bestellung auszuführen, aber das ist eine sehr unfreundliche Benutzeroberfläche.

Was können Sie dagegen tun?

Eine Lösung wäre die Beispielapplikation `Tab2.vbp`, die Sie in Abbildung 28.8 sehen.

Abb. 28.8: Im Beispiel Tab2.vbp werden unzulässige Operationen verhindert

Dieses Beispiel erlaubt dem Benutzer, jederzeit auf jede dieser Registerkarten zu gehen. Der Beispielcode von `TabStrip1_Click` wurde jedoch so abgeändert, daß er prüft, ob der angeforderte Frame einen zulässigen Status darstellt. Andernfalls wird ein spezieller Frame angezeigt, der dem Benutzer mitteilt, daß die gewünschte Operation zu diesem Zeitpunkt nicht möglich ist. Wenn Sie also versuchen, vor der Anmelde-Registerkarte eine andere Registerkarte anzuwählen, sehen Sie einen Frame, der anzeigt, daß diese noch nicht zur Verfügung steht. Diesen Code sehen Sie hier:

```
Private Sub TabStrip1_Click()
   Dim invalid As Boolean
   ' Zuerst die Auswertung
   Select Case TabStrip1.SelectedItem.Index
      Case 1   'Anmeldung ist immer erlaubt

      Case 2   ' Artikel
               If Not BuyItObject.LoggedIn Then invalid = True
      Case 3   ' Menge
               If Not BuyItObject.LoggedIn Then invalid = True
      Case 4   ' Bestellung
               If Not BuyItObject.LoggedIn Then invalid = True
               If BuyItObject.ItemQuantity <= 0 Or BuyItObject.ItemToBuy _
                           = "" Then invalid = True
   End Select

   If invalid Then
      InvalidFrame.Visible = True
      InvalidFrame.Move TabStrip1.ClientLeft, TabStrip1.ClientTop, _
                     TabStrip1.ClientWidth, TabStrip1.ClientHeight
      Exit Sub
   End If

   TabFrame(TabStrip1.SelectedItem.Index).Visible = True
   TabFrame(TabStrip1.SelectedItem.Index).Move TabStrip1.ClientLeft, _
       TabStrip1.ClientTop, TabStrip1.ClientWidth, TabStrip1.ClientHeight
   Select Case TabStrip1.SelectedItem.Index
      Case 1   'Anmeldung
               If BuyItObject.LoggedIn Then BuyItObject.Logout
      Case 2   ' Artikel
               txtItem.Text = BuyItObject.ItemToBuy
      Case 3   ' Menge
               lblPrompt.Caption = "How many " & BuyItObject.ItemToBuy _
& " do you want?"
               txtQuantity.Text = BuyItObject.ItemQuantity
      Case 4   ' Bestellung
               lblItem.Caption = BuyItObject.ItemQuantity & " x " & _
                     BuyItObject.ItemToBuy
```

```
    End Select
End Sub
```

Sie haben vielleicht den Eindruck, daß auch dies eine schlechte Benutzeroberfläche ist, aber Sie sehen so etwas doch andauernd. Betrachten Sie die Menüeinträge für die verschiedenen Visual-Basic-Menüs. Sie sehen, daß viele davon ausgegraut (deaktiviert) sind. Führen Sie ein Visual-Basic-Programm aus; Sie sehen, daß jetzt andere Menüeinträge deaktiviert wurden. Was Sie hier sehen, ist vom Konzept her dasselbe, womit wir es zu tun haben. Bestimmte Menüeinträge sind zu bestimmten Zeitpunkten erlaubt – mit anderen Worten, bestimmte Menüeinträge sind aktiviert, wenn sich die Visual-Basic-Umgebung in einem bestimmten Status befindet.

Sie sehen, fast jede nicht-triviale Applikation braucht die Fähigkeit, Status zu verwalten und die Abfolge zu steuern. Der Versuch, eine Benutzeroberfläche einzusetzen, die keine Abfolgesteuerung unterstützt, bedingt, daß Sie die verfügbare Funktionalität jederzeit selbst steuern.

Ein letzter Client. Das Projekt `OneForm.vbp` zeigt noch einen Ansatz, der für einfache Applikationen verwendet werden kann. Alle erforderlichen Informationen werden auf einem einzigen Formular untergebracht, wie in Abbildung 28.9 gezeigt.

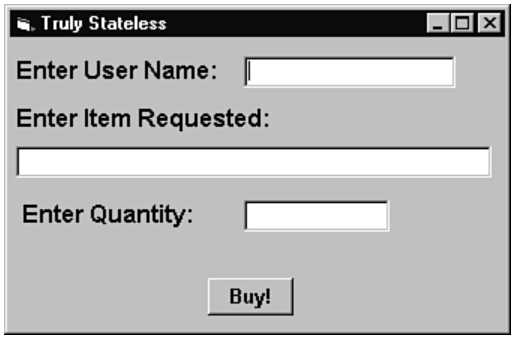

Abb. 28.9: Der Ansatz mit einem einzigen Formular

Dieses Formular enthält eine einzige Schaltfläche, mit der die Bestellung ausgeführt wird. Es enthält Textfelder, in die Sie den Kundennamen, den Artikel und die Menge eingeben. Wenn Sie die Schaltfläche anklicken, wird ein neues `BuyIt1`-Objekt erzeugt. Der Code ruft dann nacheinander die Methoden des Objekts auf, um Sie anzumelden, den Artikel anzugeben, die Menge anzugeben und schließlich die Bestellung auszuführen. Danach wird das Objekt gelöscht.

```
' Beispiel OneForm.vbp
' Copyright © 1998 by Desaware Inc. all Rights Reserved

Option Explicit
```

```
Private Sub Command1_Click()
   Dim res&
   Dim BuyItObject As New BuyIt1
   res = BuyItObject.Login(txtLogin.Text)
   If res = 0 Then
      res = BuyItObject.SetProduct(txtItem.Text)
   End If
   If res = 0 Then
      res = BuyItObject.SetQuantity(Val(txtQuantity.Text))
   End If
   If res = 0 Then
      res = BuyItObject.Buy()
   End If
   If res = 0 Then
      MsgBox "Purchase successful"
   Else
      MsgBox "Error occurred"
   End If

End Sub
```

Das Beispiel `OneForm.vbp` ist irgendwie ganz anders als die vorherigen Beispiele. Das Formular `OneForm` ist wirklich statuslos: Es gibt keine Abfolge, und es ist nicht erforderlich, Daten zu speichern. Das Formular verwendet einen Automaten, aber die Applikation selbst ist kein Automat.

28.1.2 Abfolge, Persistenz und Skalierbarkeit

Rekapitulieren wir:

Sie haben gesehen, daß die meisten nicht-trivialen Applikationen einen Automaten der einen oder anderen Form verwenden, obwohl die meisten Programmierer sich nicht damit aufhalten, die möglichen Statuszustände der Applikation formal zu definieren. Eine komplexe Applikation kann so viele mögliche Statuszustände haben, daß es gar nicht möglich ist, sie alle zu definieren und zu testen (überlegen Sie doch einmal: viele Fehler resultieren daraus, daß der Benutzer eine unerwartete Operationsabfolge ausführt, d.h. der Benutzer führt eine Operation aus, die im Automaten nicht definiert ist).

Sie haben auch gesehen, daß für die Implementierung eines Automaten zwei verschiedene Aufgaben erledigt werden müssen. Sie müssen die Abfolge steuern, und Sie müssen Statusinformationen speichern.

Die einfachste Methode zur Abfolgesteuerung ist die Verwendung einer Benutzeroberfläche oder Objektschnittstelle, die die Abfolge genau steuern, indem sie entweder einen Fehler zurückgeben, wenn versucht wird, eine unzulässige Operation auszuführen, oder indem alle unzulässigen Operationen von vornherein verhindert werden.

Am einfachsten verwaltet man Statuszustände, indem man das Objekt, das den Automaten implementiert, die gesamte Lebensdauer des Automaten über gültig sein läßt, und die Statusinformation einfach in den Variablen des Objekts ablegt.

Was spricht dagegen, es immer so einfach zu machen?

Probleme mit der Abfolge. Die Beispiele mit den Registerkarten haben gezeigt, daß man möglicherweise aufgrund der Wünsche eines uneinsichtigen Vorgesetzten eine Benutzeroberfläche einsetzen muß, die die Abfolge nicht unterstützt. Ein häufigerer Grund für die Verwendung einer solchen Oberfläche ist, daß Sie eine Applikation über das Internet oder im Firmen-Intranet unter Verwendung eines Web-Browsers implementieren möchten.

Hier wollen wir für den Moment die Tatsache ignorieren, daß es sehr vernünftig ist, verteilte Applikationen mit Hilfe komplexer Client-Applikationen zu implementieren, die die gesamte Abfolge mit DCOM oder anderen Remote Automation-Technologien steuern. Offensichtlich gibt es viele Fälle, wo es sinnvoll wäre, eine Applikation mit Hilfe eines Web-Browsers zu implementieren (beispielsweise ein Online-Kaufhaus), und auch viele Fälle, wo es wenig Sinn macht, aber die Leute es trotzdem tun.

Das einzige Problem mit einem Web-Browser ist, daß der Benutzer direkt in eine URL eintreten oder ein Lesezeichen dafür setzen kann, mit allen Parametern für diese URL. Das entspricht in etwa dem Programm mit den Registerkarten, wo es dem Benutzer möglich ist, jederzeit an jede Stelle der Applikation zu gehen. Mit anderen Worten, wenn es um die Abfolgesteuerung geht, ist ein Web-Browser ein schreckliches Werkzeug. Damit haben Sie drei Alternativen:

- Sie bitten den Benutzer, artig zu sein. Häufig sieht man Sites, die den Benutzer bitten, die ZURÜCK-Schaltfläche nicht zu benutzen, oder nur ein Lesezeichen für die Homepage der Site zu setzen. Wenn der Benutzer diesen Bitten nicht entspricht, dann können einfache Fehlermeldungen auftreten, Verwirrungen entstehen, aber auch unbeabsichtigte Aufträge für 50000 getupfte, innen angerauhte Schlafanzüge vorgenommen werden, die mit Luftpost an Bill Gates geschickt und dem Bundesgerichtshof in Rechnung gestellt werden.
- Sie werfen jede Menge Fehler auf. Immer wenn die Site eine Anforderung erhält, die keinen Sinn macht, geben Sie einfach einen Fehler zurück, der anzeigt, daß die Seite nicht existiert, und/oder daß der Benutzer ein Idiot ist, weil er Ihrer Bitte nicht Folge geleistet hat.
- Sie entwickeln eine intelligente Site. Eine solche Site würde dem Beispiel `tab2.vbp` entsprechen, wo der Server den Status der einzelnen Benutzer beobachtet und nur Seiten sendet, die gültige Operationen darstellen. Wenn der Benutzer versucht, direkt auf eine Seite zu springen, die für den aktuellen Status unzulässig ist, reagiert der Server mit der korrekten Seite für den aktuellen Status.

Sie sind sicher nicht überrascht, daß ich Ihnen die dritte Option empfehle. Glücklicherweise bieten die IIS-Anwendungen von Visual Basic eine geeignete Umgebung, die diesen Ansatz unterstützt.

Probleme mit der Persistenz. Die Persistenz bedeutet, daß ein Objekt Daten speichern oder Statusinformationen verwalten kann. Die einfachste Methode dafür war, das Objekt während der gesamten Ausführung der Applikation beizubehalten; warum sollte man dann überhaupt etwas anderes tun?

Wenn Sie einen Automaten innerhalb einer Applikation erzeugen, wie Sie es im Beispiel WIZ1.VBP gesehen haben, dann wären Sie ein Narr, würden Sie nicht die einfache Methode verwenden. Legen Sie einfach die Objekte an, die Sie brauchen, und löschen Sie sie, wenn Sie sie nicht mehr brauchen.

Wenn Sie eine WebClass oder ein MTS-Objekt anlegen, können Sie das Objekt ebenfalls am Leben erhalten. Für WebClass-Objekte setzen Sie die StateManagement-Eigenschaft einfach auf WCRETAININSTANCE, dann bleibt das Objekt während der gesamten Sitzung des Benutzers existent. Wenn es sich um ein MTS-Objekt handelt, erlauben Sie nicht, das Objekt zu deaktivieren. Die Objekte existieren weiter, bis sie am Ende der Applikation freigegeben werden. Alle ihre internen Variablen bleiben existent, und somit kann ihr Status zwischen den Methodenaufrufen verwaltet werden.

In einigen Fällen ist die einfache Methode jedoch nicht die beste Methode. Um zu erklären, warum das so ist, muß ich über die Skalierbarkeit sprechen.

Skalierbarkeit. Das Problem ist, daß die Skalierbarkeit zu einem dieser Wischiwaschi-Ausdrücke geworden ist, und jeder glaubt, sie wäre extrem wichtig für jeden, und es sei erforderlich, jede Menge Zeit und Geld in die Softwareentwicklung und neue (teuere) Skalierbarkeits-Software zu investieren, um die Skalierung zu verbessern.

Wenn Sie der Chef-Entwickler von Amazon.com sind und Zehntausende von Benutzern Ihre Bestell-Applikation gleichzeitig nutzen, dann ist die Skalierbarkeit natürlich ein großes Problem. Und selbst wenn Sie nur ein Objekt hätten, das die Applikation bedient (und für eine komplexe Abbildung wie die von Amazon.com sind zweifellos mehrere Objekte erforderlich, ebenso wie Datenbankverbindungen), wird durch Tausende dieser Objekte im Speicher selbst die schnellste Maschine gebremst. Sie wären sicher an Methoden interessiert, die Anzahl dieser Objekte zu reduzieren, um Ihre Applikation skalierbarer zu machen. Dasselbe gilt, wenn Sie eine Multi-Tier-Applikation entwickeln, die nicht auf einem Browser basiert – Sie können auch dann mit zu vielen Objekten auf Ihren Servern enden.

Sind Sie dagegen Chef-Entwickler für Amazonians.com (die neue Site, die die Dienste von Wonder Woman und anderer Superhelden vermarktet), haben Sie vielleicht ein paar Dutzend Leute pro Monat, die Ihre Site ausführen, abhängig davon, wie sehr das Böse die Welt gerade im Griff hat. Der Entwickler dieser Site muß sich kaum Gedanken um die Skalierbarkeit machen. Ein paar Dutzend exi-

stierender Objekte fällt niemals auf, und wenn, dann setzt man einfach eine neue CPU ein und spendiert ein paar Hundert Megabyte mehr Speicher.

Es ist sehr wahrscheinlich, daß Sie irgendwo zwischen diesen beiden Extremen liegen. Sie sind vielleicht versucht, einfach von Anfang an Skalierbarkeit in Ihrer Applikation vorzusehen, falls Ihre Site die beliebteste im ganzen Internet werden sollte. Bevor Sie jedoch anfangen, sollten Sie eines berücksichtigen: Die Techniken, die die Skalierbarkeit verbessern, reduzieren in der Regel die Performance Ihrer Site für eine kleinere Anzahl an Benutzern. Warum? Weil die Techniken selbst einen zusätzlichen Overhead darstellen. Das gilt auch für mittelgroße Firmen. Sie haben vielleicht 1000 Benutzer auf Ihrem System, aber wie viele davon verwenden eine bestimmte Applikation gleichzeitig? Stellen Sie sich vor, die Hälfte davon ist in der Kaffeepause, krank, in Meetings oder wirklich bei der Arbeit. Sie brauchen in Wirklichkeit vielleicht nur 20 oder 30 Benutzer gleichzeitig in Ihrer Applikation zu unterstützen. Die Reduzierung der Objektanzahl ist in so einem Fall vielleicht den Aufwand nicht wert.

Wie verbessern Sie die Skalierbarkeit? Indem Sie die Anzahl der auf dem Server existierenden Objekte reduzieren. Wie können Sie die Anzahl der Objekte reduzieren? Indem Sie der Applikation erlauben, sie zu erzeugen, wenn sie sie braucht, und sie zwischen den Methodenaufrufen zu verwerfen, oder indem Sie der Applikation erlauben, existierende Objekte mit einer anderen Datenmenge wiederzuverwenden. Beide Ansätze werden von IIS-Anwendungen und vom Microsoft Transaction Server unterstützt.

Beide Ansätze machen es aber auch erforderlich, daß das Objekt selbst seinen Status nicht innerhalb des Objekts speichert. Es muß die Aufgabe, Persistenz für Statusdaten bereitzustellen, an einen anderen Mechanismus delegieren. Das Objekt selbst wird damit »statuslos«. Beachten Sie, daß der Begriff »statuslos« in diesem Kontext nichts mit der Ablaufsteuerung zu tun hat. Das Objekt kann weiterhin, abhängig von seinem aktuellen Status, bestimmte Statusübergänge verweigern. Es bedeutet einfach, daß zwischen den Zeiten, wo das Objekt bestimmte Operationen ausführt (also zwischen den Methodenaufrufen oder Web-Seiten-Besuchen), es vielleicht nicht existiert, und interne Statusdaten müssen an anderer Stelle abgelegt werden. Später in diesem Kapitel werden wir einige der Techniken kennenlernen, die es zum Speichern von Statusinformationen gibt (siehe Abschnitt »Wie man den Status persistent macht«).

Offensichtlich bedeutet das Speichern des Status außerhalb des Objekts einen zusätzlichen Aufwand von Ihrer Seite her. Das ist auch die Quelle für den zusätzlichen Overhead bei der Verwendung von statuslosen Objekten. Microsofts Vorschlag, alle Ihre Komponenten statuslos zu machen, ist meiner Meinung nach eine zu starke Vereinfachung. Angenommen, der Status Ihrer Komponenten kann den Programmieraufwand wesentlich erleichtern (und damit die Entwicklungskosten für die Applikation reduzieren). Microsoft's Strategie, Windows NT in ein Enterprise-System umzuwandeln, scheint die vielen kleineren oder mittleren Firmen zu vergessen, für die diese Lösungen und Empfehlungen einfach Overkill wären. Sie

sollten sorgfältig überlegen, wie viele Leute Ihre server-seitigen Applikationen gleichzeitig nutzen werden. Wenn das sehr wenige sind, können Sie die meisten Empfehlungen zur Skalierbarkeit ignorieren und die einfachere Lösung verwenden, die Komponenten während der Lebensdauer der Applikation aktiviert zu halten.

Wie klein ist klein?

Ich weiß es nicht. Benchmarking ist schwarze Magie. Ich habe genug gearbeitet, um zu sehen, daß die Techniken für die Skalierbarkeit die Performance für eine kleine Anzahl gleichzeitiger Benutzer reduzieren, aber es ist nicht klar, wo man die Grenze setzen soll. Wenn Sie Tausende von gleichzeitigen Benutzern erwarten, dann ist es sehr wahrscheinlich, daß Ihnen die Skalierbarkeit am Herzen liegt. Wenn Sie nur ein paar Dutzend gleichzeitiger Benutzer erwarten, dann ist sie möglicherweise völlig irrelevant. Alles was dazwischen liegt, müssen Sie für sich selbst ausprobieren (besuchen Sie unsere Site unter `www.desaware.com`, dort finden Sie sicher in der Zukunft immer wieder Artikel über dieses Thema).

Die Schlüsselwörter sind hier gleichzeitige Benutzer. Wenn ein Benutzer seine Aufgabe erledigt hat, würde man hoffen, daß die Objekte, die er verwendet, freigegeben werden, so daß die Anzahl der existierende Objekte nicht ins Unendliche wächst. Selbst auf Web-Sites, wo status-beobachtende Objekte vom Server für eine bestimmte Zeit aktiv bleiben, ist das Anlegen von Objekten auf die Seiten beschränkt, die Teil der IIS-Anwendung sind. Wenn Sie nicht Ihre gesamte Site als IIS-Anwendung aufbauen (was für die meisten Sites ebenfalls zu viel des Guten wäre), stellen Sie vielleicht fest, daß nur ein Bruchteil Ihrer Besucher den Teil mit der IIS-Anwendung Ihrer Site benutzt.

28.2 Status in IIS-Anwendungen

IIS-Anwendungen bieten einige interessante Herausforderungen bei der Implementierung von Automaten. Als erstes wollen wir die Beispielapplikation `wiz1.vbp` in eine einfache IIS-Anwendung portieren, die nicht statuslos ist.

Das Projekt `withstate.vpb` im Web-Verzeichnis für Kapitel 28 (`ch28web`) ist eine ganz einfache Web-basierte Implementierung unserer Bestell-Applikation. Sie verwendet vier Schablonen, eine für die Anmeldung, eine für die Eingabe des Artikelnamens, eine für die Angabe der Artikelmenge, und eine für die Bestätigung der Bestellung.

Die WebClass erzeugt ein einziges `BuyIt1`-Objekt, wenn sie zum ersten Mal initialisiert wird. Weil die Eigenschaft StateManagement gesetzt ist, um die WebClass-Instanz beizubehalten, müssen wir uns nicht darum kümmern, ob das Objekt zerstört wird; die WebClass wird erst nach dem Ende der Sitzung des Benutzers zerstört. Wenn die Applikation gestartet wird, zeigt die WebClass die Anmeldeseite an, `ws1UserName`. Der WebClass-Code ist im folgenden gezeigt:

```
' Beispielprojekt WithState
' Copyright © 1998 by Desaware Inc. All Rights Reserved

Option Explicit
Option Compare Text

Private BuyItObject As BuyIt1


Private Sub WebClass_Initialize()
    Set BuyItObject = New BuyIt1
End Sub


Private Sub WebClass_Start()
   Set WebClass.NextItem = ws1Username
End Sub

Private Sub WebClass_Terminate()
   Set BuyItObject = Nothing
End Sub
```

Der Code für die Schablone der Anmeldeseite folgt. Drei Ereignisse werden unterstützt. Während des `ProcessTag`-Ereignisses wird ein existierender Benutzername angezeigt, zur Bequemlichkeit, falls sich die Person, die sich abgemeldet hat, wieder anmelden möchte. Während des `Respond`-Ereignisses wird die Schablone angezeigt. Das Formular versucht, eine Anmeldung vorzunehmen. Falls diese fehlschlägt, wird eine Meldung angezeigt, und danach wieder der Anmeldebildschirm. Andernfalls wechselt die Applikation zum Bildschirm für die Eingabe des Artikels:

```
Private Sub ws1Username_ProcessTag(ByVal TagName As String, TagContents
_
                As String, SendTags As Boolean)
   If TagName = "WC@UserName" Then
      TagContents = Replace(TagContents, "currentuser", _
                BuyItObject.CustomerName)
   End If
End Sub

Private Sub ws1Username_Respond()
   ws1Username.WriteTemplate
End Sub

Private Sub ws1Username_UserForm()
   Dim res&
   res = BuyItObject.Login(Request.Item("UserNameField"))
   If res < 0 Then
```

```
      Response.Write "<P>Illegal user name specified<P>"
      Set NextItem = ws1Username
   Else
      Set NextItem = ws1EnterItem
   End If
End Sub
```

Listing 28.2 zeigt den Rest des Beispielprogramms `withstate.vbp`. Der Code, der die restlichen Schablonen verarbeitet, ist ganz einfach und folgt demselben Muster wie der Code für den Anmeldebildschirm.

```
Private Sub ws1ConfirmPurchase_Confirm()
   Dim res&
   res = BuyItObject.Buy()
   If res = 0 Then
      Response.Write "<P>Purchase confirmed<P>"
   Else
      Response.Write "<P>Purchase failed<P>"
   End If
   Set NextItem = ws1EnterItem
End Sub

Private Sub ws1ConfirmPurchase_ProcessTag(ByVal TagName As String, _
                    TagContents As String, SendTags As Boolean)
   Select Case TagName
      Case "WC@Quantity"
         TagContents = BuyItObject.ItemQuantity
      Case "WC@Item"
         TagContents = BuyItObject.ItemToBuy
   End Select

End Sub

Private Sub ws1ConfirmPurchase_Respond()
   ws1ConfirmPurchase.WriteTemplate
End Sub

Private Sub ws1EnterItem_Hyperlink1()
   Call BuyItObject.Logout
   Set NextItem = ws1Username
End Sub

Private Sub ws1EnterItem_ItemForm()
   Dim res&
   res = BuyItObject.SetProduct(Request.Item("ItemRequestedField"))
   If res < 0 Then
      Response.Write "<P>Illegal item specified<P>"
      Set NextItem = ws1EnterItem
   Else
      Set NextItem = ws1EnterQuantity
```

```
   End If

End Sub

Private Sub ws1EnterItem_ProcessTag(ByVal TagName As String, _
          TagContents As String, SendTags As Boolean)
   If TagName = "WC@ItemRequested" Then
      TagContents = Replace(TagContents, "itementered", BuyItOb-
ject.ItemToBuy)
   End If
End Sub

Private Sub ws1EnterItem_Respond()
   ws1EnterItem.WriteTemplate
End Sub

Private Sub ws1EnterQuantity_Hyperlink1()
   Call BuyItObject.Logout
   Set NextItem = ws1Username
End Sub

Private Sub ws1EnterQuantity_ProcessTag(ByVal TagName As String, _
                  TagContents As String, SendTags As Boolean)
   Select Case TagName
      Case "WC@Item"
         TagContents = BuyItObject.ItemToBuy
      Case "WC@ItemQuantity"
         TagContents = Replace(TagContents, "itemquantitynum", _
                       BuyItObject.ItemQuantity)
   End Select
End Sub

Private Sub ws1EnterQuantity_QuantityForm()
   Dim res&
   res = BuyItObject.SetQuantity(Request.Item("ItemQuantityField"))
   If res < 0 Then
      Response.Write "<P>Illegal quantity specified<P>"
      Set NextItem = ws1EnterQuantity
   Else
      Set NextItem = ws1ConfirmPurchase
   End If
End Sub

Private Sub ws1EnterQuantity_Respond()
   ws1EnterQuantity.WriteTemplate
End Sub
```

Listing. 28.2: Das Projekt withstate.vbp

28.2.1 Zurück zur Ablaufsteuerung

Sie denken vielleicht, daß durch Beibehaltung der WebClass (d.h. nicht statuslos) alle Probleme dieser Applikation gelöst wären. Versuchen Sie, das Projekt auszuführen und einen Artikel zu bestellen. Anschließend verwenden Sie die Liste der zuletzt besuchten Sites, um auf beliebige andere Seiten zu springen. Überraschung! Sie finden sich schnell in einem Zustand, wo es fast unmöglich ist, irgend etwas zu tun. Versuchen Sie beispielsweise, sich abzumelden, und dann direkt auf den Bildschirm für die Artikeleingabe zu springen. Sie werden feststellen, daß Sie keinen Artikel eingeben können, weil Sie nicht angemeldet sind!

Wenn die WebClass den Status verwaltet, wird das Problem der Beibehaltung von Statusinformationen gelöst, aber nicht das Problem der Ablaufsteuerung. Web-Browser sind wie der Registerkartendialog aus unseren früheren Beispielen – der Benutzer kann jederzeit an eine beliebige Stelle springen. Es bedeutet einen zusätzlichen Aufwand, sicherzustellen, daß die Abfolge richtig verarbeitet wird. Das Projekt `withstate2.vbp` demonstriert einen Ansatz, der dieses Problem lösen soll.

Alle `Respond`-Ereignisse der Schablone wurden verändert, wie im folgenden gezeigt:

```
Private Sub ws1ConfirmPurchase_Respond()
   ' Abfolge überprüfen
   If BuyItObject.ItemQuantity = 0 Or BuyItObject.ItemToBuy = "" Then
      Set NextItem = ws1Username
      Exit Sub
   End If
   ws1ConfirmPurchase.WriteTemplate
End Sub

Private Sub ws1EnterItem_Respond()
   If Not BuyItObject.LoggedIn Then
      Set NextItem = ws1Username
      Exit Sub
   End If
   ws1EnterItem.WriteTemplate
End Sub

Private Sub ws1EnterQuantity_Respond()
   ' Abfolge überprüfen
   If Not BuyItObject.LoggedIn Or BuyItObject.ItemToBuy = "" Then
      Set NextItem = ws1Username
      Exit Sub
   End If
   ws1EnterQuantity.WriteTemplate
End Sub
```

```
Private Sub ws1Username_Respond()
   ' Abmeldung erzwingen, wenn diese Seite erreicht wird
   BuyItObject.Logout
   ws1Username.WriteTemplate
End Sub
```

Bevor die Schablone geschrieben wird, prüft die Applikation, ob der Status für den angeforderten Bildschirm stimmt. Wenn nicht, meldet die Applikation den Benutzer ab (falls er angemeldet ist), und zeigt das Anmeldeformular an. Natürlich wäre es besser, das Programm würde den Grund für diese Umleitung erklären, statt einfach nur seltsame Fehlermeldungen anzuzeigen, wie es hier der Fall ist.

Dieser Code demonstriert übrigens ebenfalls wieder, warum Microsoft empfiehlt, während des `Respond`-Ereignisses immer die `WriteTemplate`-Methode aufzurufen. Die `WriteTemplate`-Methode wirft selbst kein `Respond`-Ereignis auf, wenn man also die Eigenschaft `NextItem` verwendet, um eine Schablone anzuzeigen, haben wir letztlich eine zentrale Position für die Ausführung der Ablaufsteuerung definiert.

28.2.2 Statuslose IIS-Komponenten

Wenn Sie die StateManagement-Eigenschaft auf `wcNoState` setzen, haben Sie der Applikation nur mitgeteilt, daß sie das WebClass-Objekt zwischen den Anforderungen verwerfen kann. In den meisten Fällen hat dies, wie auch in unserem Beispiel, keine Auswirkungen auf die Ablaufsteuerung für die Gesamtapplikation. Es wird dadurch jedoch unmöglich für uns, das `BuyIt1`-Objekt in der WebClass zu speichern. Es würde zusammen mit dem WebClass-Objekt zwischen den Anforderungen freigegeben.

Das Projekt `NoState.vbp` demonstriert einen der vielen Ansätze für die Datenpersistenz, wenn WebClass gelöscht wird. Das `BuyIt1`-Objekt (jetzt in der Datei `BuyIt2.cls`) wird so geändert, daß es öffentlich erzeugt werden kann. Das Objekt wird während des ersten `BeginRequest`-Ereignisses für die WebClass erzeugt. Während dieses Ereignisses prüft das Programm, ob das Objekt bereits im `Session`-Objekt gespeichert ist. Wenn nicht, wird ein neues angelegt. Das Objekt wird mit Hilfe der Methode `Server.CreateObject` erzeugt. Damit ist es dem Server möglich, das Threading-Modell für die Klasse festzulegen, wobei immer das Apartment-Modell verwendet werden soll.

Am Ende der Anforderung (das durch das EndRequest-Ereignis der WebClass angezeigt wird) wird das Objekt im Session-Objekt gespeichert, falls dieses existiert. Der Code dazu ist im folgenden dargestellt:

```
Private BuyItObject As BuyIt1


Private Sub WebClass_BeginRequest()
```

```
   On Error Resume Next
   Set BuyItObject = Session("MyObject")
   If BuyItObject Is Nothing Then
      Set BuyItObject = Server.CreateObject("NSProject1.BuyIt1")
   End If
End Sub

Private Sub WebClass_EndRequest()
   Set Session("MyObject") = BuyItObject
   Response.Expires = 0
End Sub

Private Sub ws1Username_Respond()
   ' Force a logout just in case any time you reach this page
   BuyItObject.Logout
   ws1Username.WriteTemplate

   ' Das Objekt muß an dieser Stelle nicht persistent sein
   Set BuyItObject = Nothing

End Sub
```

In einer Änderung gegenüber dem vorherigen Beispiel wird die Variable `BuyItObject` gelöscht, wenn der Benutzer am Ende einer Anforderung abgemeldet wird. Das reduziert die Systemlast, weil sichergestellt wird, daß keine zusätzlichen Objekte mehr vorhanden sind, wenn die Applikation normal beendet wird (durch Abmeldung des Benutzers). Außerdem wird das Debugging dadurch einfacher, weil VB angibt, daß das Objekt noch verwendet wird, wenn Sie die Applikation anhalten, solange noch vom `Session`-Objekt darauf verwiesen wird.

28.2.3 Wie man den Status persistent macht

Die Visual-Basic-Dokumentation beschreibt einige Ansätze, wie man Statusinformationen für Ihre IIS-Anwendungen beibehält. Hier folgt meine kurze Liste mit Ansätzen, sowie Abwägungen, die für jeden dieser Ansätze getroffen wurden.

Legen Sie das Objekt in der Session-Variablen ab. Dies ist der Ansatz, der im Beispiel `Nostate.vbp` verwendet wird. Wenn Sie ein Apartment Model Threaded Object in der `Session`-Variablen speichern, wird die Sitzung immer in dem Thread ausgeführt, der das Objekt erzeugt hat. Es ist unwahrscheinlich, daß dadurch ein Problem entsteht, weil andere Sitzungen (d.h. andere Benutzer) weiterhin in unabhängigen Threads ausgeführt werden können. Achten Sie darauf, keine single-threaded-Komponente im `Session`- oder `Application`-Objekt zu speichern, weil Sie damit Ihren ganzen Server auf einen einzigen Thread beschränken könnten.

Speichern Sie Daten in der Session-Variablen. Sie können auch andere Arten von Variablen in der Session-Variablen ablegen, beispielsweise Strings oder Felder.

Diese haben den Vorteil, daß Sie Ihre Sitzung nicht auf einen einzigen Thread beschränken. Ein sinnvoller Ansatz, wenn Sie nicht zu viele Daten zu speichern haben.

Speichern Sie ein Byte-Feld, das aus einem `PropertyBag` erzeugt wurde, in der Session-Variablen oder an einer anderen Position. Wenn Sie mehrere Klassen haben, könnten Sie sie, statt alle Objekte zu speichern, alle persistent machen, indem Sie sie in einem `PropertyBag` ablegen und dann das Bytefeld aus dem `PropertyBag` speichern. Dieser Ansatz ist zwar bequem, aber mit Sicherheit langsamer, als nur die Objekte zu speichern.

Speichern Sie Daten im `Application`-Objekt. Daten, die im `Application`-Objekt gespeichert werden, können von allen Clients gemeinsam genutzt werden, die die aktuelle Applikation verwenden. Deshalb werden Sie das `Application`-Objekt sehr viel wahrscheinlicher für andere Aufgaben verwenden, die für die Applikation anfallen, als den Status für einen Client dort abzulegen.

Verwenden Sie einen Sitzungs-Cookie. Cookies, die kein Ablaufdatum haben, werden im Speicher im Browser abgelegt und nicht auf der Platte des Clients. Wenn die Datenmenge, die Sie speichern möchten, sehr klein ist, können die Cookie-Daten allein genug sein. Größtenteils verwenden Sie jedoch den Cookie sicher als Schlüssel, um die Daten an anderer Stelle auf dem Server zu suchen. Gebräuchliche Plätze, wo Daten auf dem Server abgelegt werden, sind Datenbanken, Dateien auf der Platte, die Registrierung sowie OLE-strukturierte Speicherdateien. Weitere Informationen über die Abwägungen zwischen diesen Ansätzen finden Sie in meinem Artikel »Persistence of Data« unter `www.desaware.com`.

Verwenden Sie UserData. UserData ist Information, die mit der http-Anforderung an die Applikation geschickt wird. Dieser Ansatz ist gut für kleine Datenmengen geeignet. Sie können diese Daten auf dieselbe Weise benutzen wie einen Cookie. Das hat den Vorteil, daß es auch funktioniert, wenn der Browser Cookies deaktiviert hat. Beachten Sie, daß Sie keine Daten in der Session-Variablen speichern können, wenn der Browser keine Cookies unterstützt.

28.2.4 Fazit

Damit beschließen wir unsere Statusbetrachtungen. Ich bin dabei kaum auf den Status für Komponenten eingegangen, die für den Microsoft Transaction Server entwickelt wurden; die hier beschriebenen Konzepte gelten jedoch auch für MTS. Sie sehen, daß es einfach war, das Konzept der Automaten auf IIS-Anwendungen anzuwenden – der Entwurf der Applikationen hat sich kaum von dem der Standalone-Applikationen unter Visual Basic unterschieden. Dasselbe gilt für MTS-Komponenten. MTS-Komponenten ermöglichen Ihnen eine genauere Kontrolle über die Lebenszeit von Objekten, weil die Komponente in der Regel selbst entscheidet, wann sie deaktiviert werden kann. MTS stellt außerdem ein Speichersystem für gemeinsam genutzte Eigenschaften bereit, das es Objekten ermöglicht, ganz einfach Daten auszutauschen. Wenn ich die MTS-Komponenten größtenteils

ignoriert habe, dann nur, weil der Microsoft Transaction Server ein Thema ist, das ein eigenes Buch verdient, das ich aber sicher nicht in der näheren Zukunft schreiben werde.

28.3 Abschließender Kommentar

Damit kommen wir zum Ende der zweiten Auflage von Dan Developing COM/ ActiveX Components with Visual Basic 6. Ich bin mir sicher, irgendwann wird es Visual Basic 7.0 geben, und ob es eine dritte Auflage dieses Buchs geben wird, hängt hauptsächlich davon ab, ob ich meine Aufgabe hier zufriedenstellend erledigt habe – und ob Ihnen das Buch bei Ihrer Arbeit hilft.

Nur noch eine letzte Bitte. Visual Basic hat sich von einer kleinen und einfachen Programmierumgebung zu einer großen, komplexen Umgebung entwickelt, die zwar nicht ganz einfach ist, aber immer noch einfacher, als alles andere. Ich bin sicher, es gibt einige Bereiche, die ausführlicher erklärt werden hätten können, als hier geschehen. Um zu erkennen, welche Themen für zukünftige Auflagen interessant sind, brauche ich ein Feedback von Ihnen. Welche Themen verstehen Sie noch nicht? Wo brauchen Sie zusätzliche Erklärungen oder Beispiele? Schreiben Sie mir ein paar Worte unter `dan@desaware.com`. Ich kann nicht versprechen, auf jede Nachricht zu antworten (ich bekomme sehr viele E-Mails), aber ich werde alle Kommentare zur Kenntnis nehmen und sie sorgfältig auswerten.

Unter `www.desaware.com` finden Sie auch Korrekturen. Kein Buch ist perfekt. Zum Zeitpunkt der Drucklegung bin ich mir keines Fehlers bewußt, aber irgendwo haben sich sicher ein paar eingeschlichen. Leser der Übersetzung finden zudem Hilfe unter `support@mut.de` und weitere Informationen auf der Webseite des Markt & Technik Werks: `www.mut.de`.

Daniel Appleman

Juli 1998