Kapitel 27

IIS-Anwendungen

Als ich die ersten Blicke auf Visual Basic 6 warf, war meine Begeisterung mäßig. Die Entwickler von Microsoft hatten hier und da einige interessante neue Funktionen eingefügt. Und es gibt interessante Informationen für diejenigen, die darauf gewartet haben, eigene Datenquellen anzulegen – meiner Meinung nach ein eher esoterisches Thema, weil ich eigentlich kein Datenbankexperte bin (ich weiß gerade genug, um sie effektiv zu bedienen, ohne mich jedoch großartig darum zu kümmern).

Dann sah ich aber einen Abschnitt über IIS-Anwendungen (auch als WebClasses bezeichnet). Mann oh Mann!

Aus meiner Sicht ist das die großartigste neue Technologie in Visual Basic 6.

Bevor wir jedoch über WebClasses sprechen, möchte ich den anderen neuen, interessanten Komponententyp erwähnen: DHTML-Applikationen.

27.1 Dynamic HTML

DHTML ist eine client-seitige Technologie, d.h. sie wird in einem Web-Browser ausgeführt.

Was bedeutet das eigentlich? Betrachten wir zunächst das normale HTML. HTML besteht aus Text und Format-Tags, die vom Web-Server an einen Browser geschickt werden.

Der Browser zeigt einfach den formatierten Text an. HTML kann auch einfache Benutzerelemente darstellen, beispielsweise Textfelder oder Schaltflächen, die auf Formularen gruppiert werden. Die Information aus diesen Formularen kann als Teil einer neuen Anforderung an einen Web-Server zurückgeschickt werden.

Dieser Ansatz weist einige Einschränkungen auf. Der Benutzer kann lediglich eine neue Server-Anforderung erzeugen. Es gibt keine Möglichkeit, Einträge zu überprüfen. Es gibt keine Möglichkeiten, andere Operationen auf dem Client auszuführen.

Microsoft und andere Hersteller haben lange Zeit nach dem Heiligen Gral der client-seitigen Programmierung gesucht. JavaScript war der erste Ansatz. Die Idee dabei war, eine allgemeine Skriptingsprache zu schaffen, die auf jeder Maschine ausgeführt werden konnte. Die Sprache sollte sicher sein – sie würde so zurechtgestutzt, daß sie auf jedem Client-System ausgeführt werden konnte, und (theoretisch) die Sicherheit oder Integrität dieses Systems nicht gefährdete. Microsoft zog nach mit VBScript, das (theoretisch) dasselbe machte. Beide Skriptingsprachen werden vom Web-Browser ausgeführt, der den in das HTML eingebetteten Skriptingcode interpretiert.

Skripting war ein Anfang, aber es wies immer noch Einschränkungen auf. Der Skriptingcode konnte erkennen, wenn der Benutzer bestimmte Operationen auf einer Web-Seite ausführte, aber er konnte kaum beeinflussen, was der Browser anzeigte.

Stellen Sie sich vor, was passieren würde, würden Sie Ihren Browser so abändern, daß er jedes Element auf einer Web-Seite lesen und in einzelne Objekte zerlegen könnte. Wir haben bereits gesehen, daß das geht. Der StockQuote-Server in Kapitel 15 zeigte eine einfache Methode, wie man eine HTML-Seite in einzelne Objekte zerlegt. Angenommen, Ihr Browser könnte das auf komplexere Weise ebenfalls, und Sie könnten den Inhalt dieser Objekte ändern, löschen oder der Seite hinzufügen, und den Browser veranlassen, das veränderte Ergebnis anzuzeigen. Jetzt könnten Sie durch Ihre Skriptingsprache erkennen, was der Benutzer auf der Seite gemacht hat, beispielsweise das Anklicken einer Schaltfläche oder eine Texteingabe, und die Web-Seite entsprechend anpassen und auf weitere Aktionen des Benutzers warten. Damit hätten Sie eine client-seitige Applikation, die in einem Web-Browser läuft. Und genau das ist – einfach ausgedrückt – Dynamic HTML.

Was hat das mit Visual Basic zu tun?

Was würde passieren, wenn Sie statt JavaScript oder VBScript, die auf einer Web-Seite eingebettet sind, eine Visual-Basic-Komponente oder ein Visual-Basic-Programm verwenden könnten, um den Browser zu steuern? Sie bzw. es würde Ereignisse vom Browser erhalten, wenn der Benutzer irgend etwas macht, und könnte dann direkt auf die Objekte auf der Seite zugreifen und sie beliebig abändern. Das ist eine Visual-Basic-DHTML-Applikation.

Das war sicher nicht Microsofts erster Versuch, eine sinnvolle Lösung für die client-seitige Programmierung zu bieten. Man entwickelte VBScript, das es mit JavaScript aufnehmen sollte, aber die meisten Leute verwenden immer noch JavaScript, weil es von mehr Web-Browsern unterstützt wird. Es scheint, daß ein großer Teil der Welt sich immer noch bockig den Nicht-Microsoft-Betriebssystemen widersetzt, insbesondere die Unix-Maschinen, die VBScript nicht oder nur wenig unterstützen. Obwohl es möglich ist, client-seitige Programme zu entwickeln, die VBScript für die Explorer-Leute und JavaScript für die Unix-Leute bereitstellen, kann bisher niemand wirklich begründen, warum man nicht einfach JavaScript verwendet, und damit jeden unterstützt.

Microsoft hat ActiveX-Steuerelemente entwickelt, die den Java-Applets (die auf der virtuellen Maschine von Java ausgeführt werden) Konkurrenz machen. Während ActiveX-Steuerelemente größten Erfolg als Applikationskomponenten haben, scheint es, daß die einzigen Leute, die sie auf Web-Seiten einsetzen, Microsoft selbst und ein paar andere Windows-spezifische Sites sind.

ActiveX-Dokumente war einfach nur ein weiterer Versuch, eine client-seitige Programmierung zu unterstützen – ich glaube, ich habe bereits alles zu diesem Thema gesagt.

Bleibt DHTML. Die gute Nachricht ist, daß DHTML ein Standard sein soll, d.h. es wird von jedem Browser-Hersteller (d.h. Netscape) auf jedem Betriebssystem auf dieselbe Weise unterstützt. Leider handelt es sich dabei um einen Standard, der noch in der Entwicklung ist, d.h. man hat sich noch nicht wirklich darauf

geeinigt. Es muß betont werden, daß DHTML nicht dasselbe ist wie die eine Visual-Basic-DHTML-Applikation. DHTML beschreibt eine Methode, eine Web-Seite als Objektmodell zu behandeln. Visual-Basic-DHTML-Applikationen stellen eine Methode dar, Komponenten zu erzeugen, die in den Internet Explorer 4.0 SP1 oder neuer »greifen«, um mit dem DHTML-Objektmodell zu arbeiten. Selbst wenn also DHTML zu einem echten Standard wird, sind VB-DHTML-Applikationen spezifisch für den Internet Explorer.

Damit kann man VB-DHTML-Applikationen in gewisser Weise in dieselbe Kategorie wie ActiveX-Steuerelemente und ActiveX-Dokumente einordnen:

- Für die Ausführung ist der Microsoft Internet Explorer erforderlich.
- Inwieweit sie korrekt ausgeführt werden, ist von der Client-Konfiguration abhängig. Beachten Sie, daß der Internet Explorer an der Spitze einer Technologiepyramide angeordnet ist, die beim Betriebssystem beginnt, über die COM- und OLE-Untersysteme und die Internet-Objekte zum Internet Explorer. Dabei sind Dutzende (Hunderte?) von Objekten und DLLs implementiert. Auf dem Client müssen alle diese Komponenten installiert und korrekt registriert sein, damit alles richtig funktioniert.
- Wenn ein Benutzer eine neue oder eine Betaversion des Internet Explorers installiert, die irgendwie nicht richtig mit Ihrer Komponente zusammenarbeitet, müssen Sie Ihre Komponente aktualisieren, um das Problem zu lösen. Nicht, daß Microsoft irgendwann versäumen würde, die Abwärtskompatibilität des Internet Explorers zu garantieren...

Damit sind wir also wieder dort, wo wir vor einem Jahr mit den ActiveX-Komponenten waren. DHTML-Applikationen stellen eine neue Technologie dar, die sich durchsetzen kann, oder auch nicht. Ich persönlich hoffe, sie wird mehr Erfolg haben.

Der andere Aspekt, den Sie bei Visual Basic DHTML-Applikationen nicht vergessen sollten, ist, daß ihre Programmierung ganz anders ist als bei allen anderen Arten von VB-Komponenten – so anders, daß ihr nicht nur ein Kapitel, sondern ein ganzes Buch gewidmet werden sollte. Zweifellos wird es sehr bald sehr viele Bücher über dieses Thema geben.

Weil DHTML eine neue Technologie mit unsicherer Zukunft ist, und ich nicht die Zeit habe, dem Thema gerecht zu werden, habe ich beschlossen, in dieser Auflage des Buchs überhaupt nicht darauf einzugehen. Ich werde darüber nachdenken. In der Zwischenzeit sollten Sie immer wieder auf der Web-Site `www.desaware.com` nachsehen. Es ist sehr wahrscheinlich, daß ich dort im Laufe der Zeit zusätzliche Artikel über dieses Thema veröffentliche.

27.2 Active Server Pages

Um IIS-Anwendungen verstehen zu können, müssen wir unsere Betrachtungsweise umkehren und uns dem Web-Server zuwenden. Ursprünglich handelte es sich bei einem Web-Server um eine sehr einfache Applikation. Er erhält Anforderungen von Seiten eines Web-Browsers, die im wesentlichen aus dem Namen einer Datei mit HTML-formatiertem Text bestehen. Der Web-Server erhält die Anforderung, liest die Datei an der angegebenen Position und sendet sie an den Browser. Wenn eine Web-Seite ein Formular enthält, wird das Formular als Teil der Befehlszeile an den Server geschickt. Ältere Server nehmen den Teil der Befehlszeile nach dem Dateinamen und schreiben ihn entweder in eine Datei auf der Platte, oder starten eine neue Applikation und senden die Information als Eingabe an die Datei. Die Ausgabe des Programms wird vom Web-Server gelesen und zurück zum Browser geschickt. Diese einfache Form server-seitiger Programmierung wird als CGI (Common Gateway Interface) bezeichnet.

CGI leidet unter diversen Nachteilen. In der Regel startet es für jede Anforderung einen neuen Prozeß, was relativ langsam ist. Es bedingt, daß Sie die gesamte Arbeit übernehmen, um die ankommenden Textinformationen auszuwerten, und das resultierende HTML zu erzeugen, was sehr viel Codierung erforderlich macht (in der Regel in einer Sprache namens PERL).

Den Internet Information Server von Microsoft gab es in mehreren Versionen, die CGI unterstützten, aber den eigentlichen Schwung hat er erst mit Version 3 und der Einführung der Active Server Pages (ASP) bekommen.

Das Konzept der Active Server Pages ist dem von DHTML sehr ähnlich, allerdings von der entgegengesetzten Seite her. Wie HTML Skripts beinhalten kann, die auf dem Browser ausgeführt werden, könnte es ebenfalls Skripts beinhalten, die vom Server ausgeführt werden, bevor er die Seite versendet. Diese Skripts könnten in VBScript oder JavaScript geschrieben werden. Welche Aufgaben müßte ein solches Skript erfüllen? Die wichtigsten davon sind im folgenden beschrieben:

- Es müßte in der Lage sein, die vom Browser gesendeten Parameterinformationen zu lesen.
- Es müßte in der Lage sein, HTML an den Browser zu senden.
- Es müßte in der Lage sein, Cookies mit dem Browser auszutauschen.

Active Server Pages stellen ein Objektmodell bereit, das es für die Skriptingsprache einfach macht, all dies und noch mehr zu tun. Die vom ASP-System bereitgestellten Objekte sind in Tabelle 27.1 aufgelistet.

ASP leidet nicht unter den Nachteilen, die zuvor für die clientseitigen Applikationen oder die CGI-Applikationen beschrieben wurden. Es bietet vielmehr die folgenden Vorteile:

Objekt	Beschreibung
Application	Wird genutzt, um Daten zwischen mehreren ASP-Seiten, die in einer Applikation gruppiert sind, gemeinsam nutzen zu können. Erlaubt die Initialisierung und das Beenden einer Applikation.
Request	Ermöglicht Ihnen, vom Browser bereitgestellte Informationen zu lesen, unter anderem den Inhalt von beliebigen Feldern eines Formulars, Benutzerdaten und Cookies.
Response	Ermöglicht Ihnen, HTML-Daten an den Browser zurückzuschicken, die angezeigt werden sollen. Erlaubt außerdem, Cookie-Werte zu setzen.
Server	Erlaubt Ihnen, die Operation des Servers zu kontrollieren und neue Objekte zu erzeugen.
Session	Wird genutzt, um Informationen für einen bestimmten Client zu speichern, der die Applikation verwendet. Damit ist es Ihnen möglich, Informationen zu verwalten, während sich der Benutzer innerhalb der Applikation zwischen den Seiten bewegt.
ObjectContext	Wird genutzt, um Transaktionen zu steuern, wenn Sie die Funktionen des Microsoft Transaction Servers in Ihrer ASP-Applikation verwenden.

Tab. 27.1: Objekte, die durch Active Server Pages bereitgestellt werden

- Es ist unabhängig vom Browser und vom Betriebssystem.
- Es ist nicht von einer korrekten Konfiguration des Client-Systems abhängig.
- Es kann nicht durch Änderungen auf dem Client-System gestört werden, es sei denn, Sie übergeben explizit HTML-Code, für den bestimmte Funktionen im Client-Browser notwendig sind.
- Es ist effizienter als CGI, weil für die Verarbeitung eingehender Anforderungen keine neuen Prozesse gestartet werden müssen.

Aber nichts ist perfekt, und auch ASP hat ein paar Nachteile:

- Es verschiebt den Rechenaufwand auf den Server, und es fallen mehr Anforderungen auf dem Server an, als es bei einem clientseitigen Ansatz der Fall wäre.
- ASP-Skritps werden interpretiert, und Interpreter sind in der Regel langsam.
- Für ASP ist es erforderlich, Server-Skripts und Client-Text und Skripts in derselben Datei zu kombinieren. ASP-Dateien können schnell unverständlich und damit nicht mehr verwaltet werden.
- ASP-Skripts sind immer schwierig zu debuggen. Und besonderes kompliziert ist das Debugging, wenn Sie Objekte aus dem Skript erzeugen und verwenden.

Visual Basic 6 führt einen neuen Komponententyp ein, die sogenannte IIS-Anwendung oder WebClass. Dabei scheint ein völlig neues Skripting stattzufinden – bis Sie genauer hinsehen. Wenn Sie auf eine WebClass zugreifen, wird der Browser auf eine ASP-Seite weitergeleitet. Hier folgt ein Beispiel für eine ASP-Seite für die Applikation `withstate.vbp`, die Sie in Kapitel 28 kennenlernen werden.

```
<%
Response.Buffer=True
Response.Expires=0

If (VarType(Application("~WC~WebClassManager")) = 0) Then
    Application.Lock
    If (VarType(Application("~WC~WebClassManager")) = 0) Then
        Set Application("~WC~WebClassManager") = Server.CreateOb-
ject("WebClassRuntime.WebClassManager")
    End If
    Application.UnLock
End If

Application("~WC~WebClassManager").ProcessRetainInstanceWebClass "WSPro-
ject.WithState", _
        "~WC~WSProject.WithState", _
        Server, _
        Application, _
        Session, _
        Request, _
        Response
%>
```

Der Text zwischen den Symbolen <% und %> ist ein serverseitiges Skript. Was macht dieses Skript? Wenn es zum ersten Mal ausgeführt wird, erzeugt es ein Objekt des Typs `WebClassRuntime.WebClassManager`. Dieses Objekt wird im Applikationsobjekt abgelegt, so daß es von jedem genutzt werden kann, der diese spezielle IIS-Anwendung einsetzt. Bei jedem Zugriff auf das Skript lädt sie dieses Objekt und ruft eine Methode dafür auf, die Parameter wie beispielsweise den Objektnamen Ihrer WebClass oder Verweise auf Server-, `Application`-, `Session`-, `Request`- und `Response`-Objekte erhält. Man kann voraussetzen, daß dieses `WebClassManager`-Objekt Ihre WebClass erzeugt und den Verweis an die anderen ASP-Objekte übergibt.

Es stellt sich also heraus, daß die IIS-Anwendungen von Visual Basic eigentlich nichts anderes sind, als eine Möglichkeit, Komponenten zu erzeugen, die von ASP ausgeführt werden. Statt Server-Skripts in Ihren HTML-Seiten zu erzeugen, delegiert ASP die Arbeit an Ihre WebClass-Komponente.

Nichts besonders. Außer daß...

- Nachdem Sie das ASP-Skript ausgeführt haben, wird Ihr gesamter Servercode zu kompiliertem Visual-Basic-Code, statt eines interpretierten VBScript oder JavaScript.
- Sie können vertrauten Visual-Basic-Code und seine Konstrukte verwenden, statt in einer anderen Skriptingsprache arbeiten zu müssen.
- Sie können den modernen, objektorientierten Ansatz in Ihrem VB-Programm verwenden, statt einer unübersichtlichen Kombination von Skripts in einer HTML-Datei.
- Sie können den clientseitigen Code einfach vom serverseitigen Code abtrennen, so daß Ihre Server-Programme leichter lesbar werden und besser unterstützt werden können.
- Sie können statische HTML-Seiten als Schablonen benutzen und eine einfache Textersetzung vornehmen, statt das erforderliche HTML selbst zu codieren.
- Sie haben Zugriff auf einen relativ einfach zu benutzenden Designer, der Ihnen ermöglicht, Seitenelemente zu verwalten und sie für die gemeinsame Nutzung durch IIS-Anwendungen zur Verfügung zu stellen.
- Und außerdem ist die Ausführung und das Debugging einer IIS-Anwendung so einfach wie das Anklicken der Schaltfläche STARTEN in VB. Sie haben vollständigen Zugriff auf alle Debugging-Werkzeuge, die in Visual Basic eingebaut sind, unter anderem Haltepunkte, Überwachungsfenster usw.

Sind die IIS-Anwendungen von Visual Basic also einfach nur ein Trick, um VB-Komponenten in das normale Active Server Pages-System des Internet Information Servers von Microsoft zu transplantieren?

Ja.

Aber es ist ein erstaunlich guter Trick! David Copperfield ist nichts dagegen, weil die Microsoft-Entwickler hiermit ein großes Problem einfach verschwinden haben lassen.

Meiner bescheidenen Meinung nach werden die IIS-Anwendungen von Visual Basic die normale ASP-Entwicklung ersetzen. Sie könnten es jedenfalls. Meine Firma wird die Web-Site ab sofort für alle neuen Entwicklungen auf diese Technologie umstellen, und wir werden auch einige unserer älteren ASP-Skripts so bald wie möglich konvertieren. Es ist einfach zu cool!

In diesem Buch werden die IIS-Anwendungen nicht detailliert beschrieben. Dafür gibt es zwei Gründe:

- Auch dieses Thema verdient ein ganzes Buch.
- Ich weiß immer noch nicht genug darüber, um dem Ganzen gerecht zu werden.

Die letzte Begründung mag ein bißchen fadenscheinig sein, aber überlegen Sie doch selbst einmal. Um mit einer Technologie vertraut zu werden, müssen Sie diese über längere Zeit in einem echten Szenario einsetzen. In meinem ersten Buch konnte ich über die Verwendung des Win32 API in Visual Basic schreiben, weil ich jahrelange Erfahrung mit dem Win32 API hatte, noch bevor ich überhaupt mit Visual Basic arbeitete. Ich konnte über ActiveX-Komponenten und Steuerelemente schreiben, als VB5 veröffentlicht wurde, weil ich neben der Entwicklung von Komponenten in Visual Basic auch unzählige Stunde damit verbrachte, mir die Hörner an der Technologie abzustoßen, zuerst mit MFC und später mit ATL.

Aber WebClasses sind neu.

Und Sie wissen so gut wie ich, daß die Unmenge an Technologie-Information, die uns Microsoft präsentiert (andere Quellen gar nicht zu erwähnen), viel zu umfangreich für einen Einzelnen ist, als daß er sich auf jedem Gebiet auskennen würde. Ich bin noch kein Experte für WebClasses, aber in ein paar Monaten werde ich es sein. Mein Verleger will aber nicht bis dahin warten, und ich bin ihm nicht böse darum – wenn ich vor jedem Buch alles über jedes Thema lernen wollte, dann hätte ich nie eines veröffentlicht (und sei es nur, weil während der Lernphase eine neue Beta von irgend etwas aufgetaucht ist, die mein Wissen veralten hätte lassen).

Ich konnte Ihnen nicht all zu viel über Active Server Pages erzählen, aber dieses Buch hat auch ein anderes Thema. Ich könnte eine passable Einführung für IIS-Anwendungen schreiben, aber dieses Buch will ja die Microsoft-Dokumentation ergänzen, nicht sie wiederholen.

Ich wollte aber unbedingt etwas über diese großartige neue Technologie schreiben. Den ersten Teil haben Sie bereits gelesen: Einen Ausblick, in welcher Beziehung IIS-Anwendungen zu ASP-Seiten stehen.

Es gibt noch drei allgemeine Bereiche, die ich zu diesem frühen Zeitpunkt ansprechen sollte.

- Zunächst möchte ich Ihnen WebClasses vorstellen. Ich weiß, ich habe gesagt, ich will die Handbücher nicht wiederholen. Aber offen gesagt, ich habe lange gebraucht, bis ich verstanden habe, was die Dokumentation über HTML-Schablonen und WebItems sagte, und in welchem Verhältnis sie zueinander stehen. Deshalb will ich auch etwas dazu sagen. Nicht daß ich glaube, ich würde es unbedingt besser als die Handbücher machen, aber ich glaube, hier handelt es sich um eine Situation, wo eine andere Perspektive Ihnen helfen könnte, die Konzepte schneller und vielleicht besser zu verstehen.
- Anschließend werde ich einige der Tricks beschreiben, die von Microsoft nicht erwähnt werden.

- Im nächsten Kapitel finden Sie eine detaillierte Beschreibung der Statusverwaltung. Dieses Thema wird im Kontext von WebClasses häufig erwähnt, aber wie Sie sehen werden, hat es sehr viel mehr in sich, als auf den ersten Blick deutlich wird.

27.3 WebClasses – aus einer anderen Perspektive

In Kapitel 9 habe ich viel Zeit auf das Thema »Designer« verwendet und dabei verdeutlicht, daß ein Designer ein Visual-Basic-Werkzeug ist, das Ihnen hilft, verschiedene Arten von Applikationen zu erstellen, und daß hinter jedem Designer ein Codemodul steht, das Ihren Code enthält. Bis jetzt gab es eine enge Beziehung zwischen dem Designer und dem Benutzeroberflächenelement für die Applikation. Ein Formular-Designer erzeugt ein sichtbares Formular. Ein UserControl-Designer erzeugt ein sichtbares UserControl. Ein ActiveX-Dokument-Designer erzeugt ein sichtbares ActiveX-Dokument. Es war, als erzeugte der Designer die Benutzeroberfläche und stellte Links zum Codemodul bereit. Das ist in Abbildung 27.1 dargestellt.

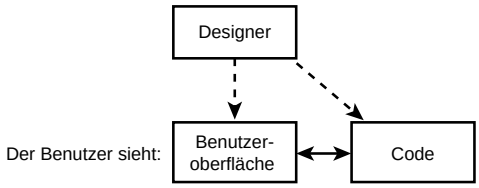

Abb. 27.1: Der Designer im Verhältnis zu einer Standardapplikation

Der Designer existiert nur in der Entwurfszeitumgebung von Visual Basic. Nachdem Sie Ihre Applikation kompiliert und weitergegeben haben, gibt es den Designer nicht mehr.

Bei IIS-Anwendungen müssen Sie das Ganze etwas anders betrachten. Der Designer entspricht nicht mehr einem Element der Benutzeroberfläche, wie Sie in Abbildung 27.2 sehen. Die Benutzeroberfläche für eine IIS-Anwendung besteht aus HTML-Text, der an den Browser geschickt wird. Dieser HTML-Text wird von zwei Objekttypen erzeugt, HTML-Schablonen-WebItems und WebItems. Ich möchte betonen, daß beide Objekte letztlich dazu da sind, dasselbe zu tun – HTML-Ausgaben zu erzeugen. Das HTML-Schablonen-WebItem verwendet eine ihm zugeordnete HTML-Datei, die es direkt übergeben oder vor dem Versenden an den Browser abändern kann. Ein WebItem erzeugt HTML-Daten im Code. Sowohl HTML-Schablonen-WebItems als auch normale WebItems sind mit dem WebClass-Klassenmodul verbunden. Der Designer existiert ebenfalls nur, solange die WebClass in der Visual-Basic-Umgebung erzeugt wird. Wie Sie jedoch sehen, gibt es keine direkte Verbindung zwischen dem Designer und dem, was letztlich an den Benutzer gesendet wird.

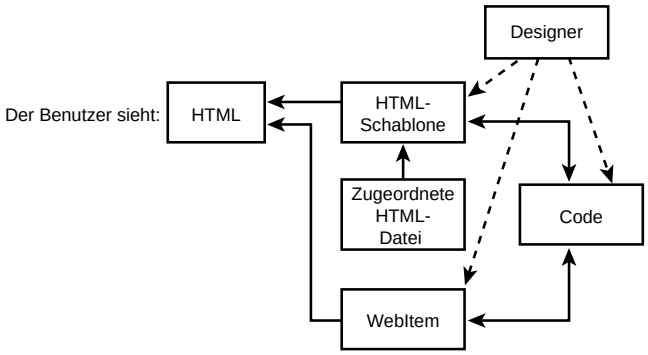

Abb. 27.2: Der Designer im Verhältnis zu einer IIS-Anwendung

27.3.1 Der Designer bei der Arbeit

Wenn Sie ein neues Projekt anlegen, dann speichern Sie die Applikation erst einmal (nachdem Sie ihr einen anderen Namen als Webclass1 gegeben haben). Das passiert, weil WebClasses nicht ganz im Speicher ausgeführt werden können; sie brauchen verschiedene Dateien in einem gemeinsamen Verzeichnis, um die WebClass erzeugen und testen zu können. In diesem Fall wollen wir das Projekt `gtpWebClassTest` und die WebClass `gtpWebClass1` nennen.

Der Designer zeigt die WebClass und zwei Ordner, einen für HTML-Schablonen-WebItems, und den anderen für normale WebItems.

Wir beginnen damit, ein neues HTML-Schablonen-WebItem zu erzeugen. Ein HTML-Schablonen-WebItem braucht eine HTML-Schablonendatei. Diese HTML-Datei kann auf beliebige Weise erzeugt werden. Ich verwende dazu FrontPage.

Die Seite enthält den folgenden Text:

```
This is a test page
It has a single hyperlink
```

Zunächst können Sie den Hyperlink beliebig setzen. Ich habe ihn hier so gesetzt, daß er auf sich selbst verweist. Es ist egal, wohin er zeigt, weil er sehr bald geändert wird.

Das HTML für dieses Seite sieht wie folgt aus:

```
<html>

<head>
<title>New Page 1</title>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
</head>
```

```
<body>

<p>This is a test page</p>

<p>It has a single <a href="newpage1.htm">hyperlink</a></p>

<p> </p>
</body>
</html>
```

Speichern Sie die Datei irgendwo; das muß nicht im Projektverzeichnis sein. Hier habe ich sie unter dem Dateinamen `ATemplate.htm` abgelegt. Als nächstes habe ich ein HTML-Schablonen-WebItem angelegt, das diese Seite verwendet. Ich benannte es in *MainPage* um. Der WebClass-Designer erscheint jetzt wie in Abbildung 27.3 gezeigt.

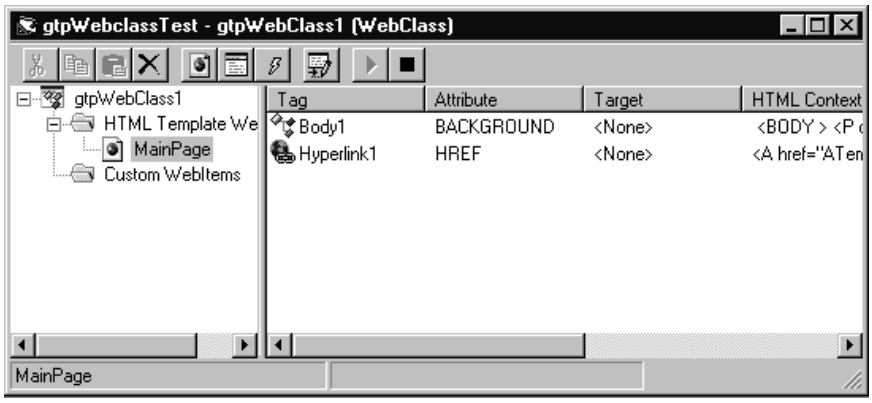

Abb. 27.3: Der WebClass-Designer

Beim Speichern des Projekts werden Sie feststellen, daß Visual Basic eine Kopie der HTML-Schablonendatei angelegt und als `ATemplate1.htm` in Ihrem Projektverzeichnis abgelegt hat. Dieser Ansatz ermöglicht, daß mehrere HTML-Schablonen-WebItems gemeinsam dieselbe HTML-Schablonendatei verwenden.

Als erstes müssen wir festlegen, was dem Benutzer mitgeteilt wird, wenn diese Seite angefordert wird. Eine leere WebClass enthält den folgenden Schablonencode:

```
Private Sub WebClass_Start()

    'Antwort an den Benutzer schreiben
    With Response
        .Write "<html>"
        .Write "<body>"
```

```
        .Write "<h1><font face=""Arial"">WebClass1's Starting Page
                            </font></h1>"
        .Write "<p>This response was created in the Start event of
                            WebClass1.</p>"
        .Write "</body>"
        .Write "</html>"
    End With

End Sub
```

Das Ereignis `WebClass_Start` wird aufgerufen, wenn der Benutzer Ihre IIS-Anwendung zum ersten Mal startet – die Sitzung, die alle WebItems enthält, aus denen sich Ihre Applikation zusammensetzt. Eine Sitzung dauert so lange, wie sich ein Benutzer auf den Seiten Ihrer Applikation bewegt und bis das folgende passiert:

- Sie beenden die Sitzung explizit.
- Ein vorgegebenes Zeitintervall ist abgelaufen, seit der Benutzer zuletzt auf die Applikation zugegriffen hat.

Das von Microsoft bereitgestellte Beispiel zeigt, wie Sie reines HTML direkt an den Browser senden. Wir werden es dabei belassen.

Wie veranlassen Sie, daß die MainPage-Schablone angezeigt wird? Sie müssen sich nach dem Start-Ereignis dorthin bewegen. Das erfolgt durch den folgenden Code:

```
Set NextItem = MainPage
```

Daran erkennt die WebClass, daß sie die aktuelle Anforderung von dem angegebenen WebItem fortsetzen soll. Was bedeutet das für dieses WebItem? Es wirft das Respond-Ereignis auf. Während des Respond-Ereignisses für ein HTML-Schablonen-WebItem ist Ihre gebräuchlichste Aktion, die HTML-Schablonendatei mit der Funktion `WriteTemplate` auf den Browser zu schreiben. Das passiert wie folgt:

```
Private Sub MainPage_Respond()
   MainPage.WriteTemplate
End Sub
```

Versuchen Sie, das Projekt auszuführen. Sie sehen folgendes:

- *WebClass1's Starting Page*
- This response was created in the Start event of WebClass1.
- This is a test page.
- It has a single hyperlink.

Der Browser sah zuerst die Ausgabe von dem Start-Ereignis, dann die von der MainPage.

Jetzt gehen Sie zum `Hyperlink1`-Eintrag im rechten Feld des Designers und klikken mit der rechten Maustaste darauf. Ordnen Sie ihn einem benutzerdefinierten Ereignis zu. Damit wird der erste Hyperlink automatisch auf die Seite für Ihren Ereignis-Handler gesetzt. Geben Sie dem Hyperlink im linken Feld den Namen *FirstLink*. Klicken Sie jetzt mit der rechten Maustaste auf den Eintrag `MainPage` im linken Feld des Designers und klicken Sie auf den Menüeintrag HTML-SCHABLONE BEARBEITEN. Wenn Ihr System korrekt konfiguriert ist, sollten Sie damit zu Ihrem bevorzugten HTML-Editor gelangen, wo die Seite angezeigt wird. Betrachten Sie den HTML-Code. Sie sollten folgendes sehen:

```
<html>

<head>
<title>This is a test page</title>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
</head>

<body>

<p>This is a test page</p>

<p>It has a single <a href="gtpWebClass1.ASP?WCI=MainPage&amp;WCE=
              FirstLink&amp;WCU">hyperlink</a></p>

<p> </p>
</body>
</html>
```

Der Link wurde geändert! Was bedeutet `gtpWebClass1.ASP?WCI=MainPage&WCE=FirstLink&WCU`?

Zerlegen wir es in seine Komponenten:

`gtpWebClass1.asp` ist der Name der Seite. Dies ist die ASP-Datei, die Sie früher in diesem Kapitel gesehen haben. Immer wenn der Benutzer auf Ihre Applikation zugreift, egal, ob es sich um einen Hyperlink oder ein Formular handelt, gelangt er zu dieser ASP-Datei. Die ASP-Datei ruft einfach die WebClassManager-Datei auf, die erzeugt wurde, als der Benutzer in die Applikation eingetreten ist. `WCI`, `WCE` und `WCU` werden als erste Parameter übergeben.

Der Parameter `WCI` gibt den Namen der Seite an. Der Parameter `WCE` gibt den Namen des aufzurufenden Ereignisses an. `WCU` ist ein Parameter, mit dessen Hilfe Sie zusätzliche Benutzerdaten spezifizieren können, was wir aber hier nicht genauer beschreiben werden.

Logisch wäre, wenn durch Anklicken dieses Hyperlinks ein Ereignis namens `Mainpage_FirstLink` in der WebClass ausgelöst würde. Gehen Sie in den Designer, und doppelklicken Sie auf das Ereignis `FirstLink`. Genau, da haben wir das Ereignis!

Fügen Sie dem Ereignis den folgenden Code hinzu:

```
Private Sub MainPage_FirstLink()
    With Response
        .Write "<html>"
        .Write "<body>"
        .Write "<h1><font face=""Arial"">I've been clicked!</font></h1>"
        .Write "</body>"
        .Write "</html>"
    End With
End Sub
```

Versuchen Sie, die WebClass erneut auszuführen, und klicken Sie jetzt auf den Hyperlink.

Die Antwort:

I've been clicked!

Jetzt erzeugen Sie ein benutzerdefiniertes WebItem für die Webclass, SecondPage. Fügen Sie seinem `Respond`-Ereignis den folgenden Code hinzu:

```
Private Sub SecondPage_Respond()
   With Response
      .Write "<html>"
      .Write "<body>"
      .Write "<p>This is our second page.</p>"
      .Write "<p>It contains a <a href=""gtpWebClass1.ASP?WCI=MainPage "">
                        hyperlink</a> back to the first page</p>"
      .Write "<p> </p>"
      .Write "</body>"
      .Write "</html>"
   End With

End Sub
```

Woher habe ich diesen Code? Ich habe in FrontPage eine Seite angelegt und das HTML von dort kopiert. Ich habe das `href`-Attribut manuell geändert, so daß es auf MainPage zeigt, nicht auf das `MainPage_FirstLink`-Ereignis.

Aber wie gelangen wir zu diesem WebItem? Bearbeiten Sie die MainPage und fügen Sie einen neuen Hyperlink ein. Wenn Sie Ihren Editor schließen, fragt Visual Basic Sie, ob die Änderungen berücksichtigt werden sollen. Stimmen Sie zu. Sie sehen den zweiten Hyperlink im rechten Feld des Designers. Klicken Sie mit der rechten Maustaste auf hyperlink2 und verbinden Sie ihn mit dem WebItem SecondPage (kein Ereignis).

Führen Sie das Beispiel aus. Die MainPage sieht jetzt wie folgt aus:

- *WebClass1's Starting Page*
- This response was created in the Start event of WebClass1.
- This is a test page.
- It has a single hyperlink.
- New link to second page.

Wenn Sie auf den zweiten Hyperlink klicken, bringt er Sie zu dem WebItem SecondPage. Der Referenzcode für diesen Link ist `gtpWebClass1.ASP?WCI=SecondPage&WCU`. Weil kein `WCE`-Parameter mit einem Ereignis angegeben wurde, wird damit das `Respond`-Ereignis ausgelöst, das den zuvor gezeigten Code sendet, um die folgende Ausgabe zu erzeugen:

- This is our second page.
- It contains a hyperlink back to the first page.

Wenn Sie auf den Hyperlink klicken, bringt er Sie zurück zur ersten Seite und zeigt den folgenden Bildschirm an:

- This is a test page.
- It has a single hyperlink.
- New link to second page.

Was ist mit dem Startseiten-Abschnitt von `WebClass1` auf der ersten Seite geworden? Es war nie Teil der MainPage. Es wurde nur vom Start-Ereignis erzeugt, und wenn Sie zurück zur ersten Seite gehen, dann haben Sie das Start-Ereignis bereits hinter sich.

Es gibt natürlich sehr viel mehr über WebClasses zu sagen, aber das meiste davon hat mit den Funktionen für die Arbeit mit ASP-Objekten und nicht mit den eigentlichen WebClasses zu tun. Und ich will gar nicht behaupten, Ihnen etwas über diese Objekte zu erzählen, obwohl Sie im nächsten Kapitel ein paar Beispiele dafür kennenlernen werden.

Das wichtigste ist jetzt, daß Sie sich ein paar Schlüsselkonzepte merken, die Ihnen Ihre Arbeit mit WebClasses leichter machen werden:

- Der WebClass-Designer hat nichts mit der Benutzeroberfläche zu tun. Es handelt sich dabei nur um ein Werkzeug, das Ihnen helfen soll, Ihre WebItem-Objekte zu organisieren. Objekte? Ja – ich weiß, Sie sind daran gewöhnt, daß sich jedes Objekt in einer separaten Klassendatei befindet, die in Ihrem Projektfenster erscheint, aber bei IIS-Anwendungen werden alle Objekte Ihrer WebClass vom WebClass-Designer verwaltet und in derselben `.DSR`-Datei abgelegt.

- Der einzige wirkliche Unterschied zwischen einem HTML-Schablonen-WebItem und einem normalen WebItem ist, daß das HTML-Schablonen-WebItem ein paar zusätzliche Funktionen bietet, die ihm ermöglichen, eine ihm zugeordnete HTML-Datei zu verwalten. Es kann diese Datei durchlaufen und so verändern, daß Hyperlinks und Formular-Objekte zurück zu Ihrer WebClass führen und im angegebenen WebItem Ereignisse auslösen. Es kann außerdem dynamisch Text ersetzen, wie Sie gleich sehen werden.
- Der ganze Trick bei den WebClasses ist, daß jeder Hyperlink in der Applikation zur selben ASP-Seite zurückgeleitet wird, und daß diese Seite die Aktionen an die verschiedenen WebItems verteilt. Nachdem Sie im WebItem sind, können Sie Visual Basic nutzen, um die wahre Leistungsfähigkeit des ASP-Systems auszunutzen.

27.3.2 Einige Tricks mit IIS-Anwendungen

Hier zeige ich Ihnen ein paar Tricks, die ich während meiner Arbeit entdeckt habe.

Tags und Textersatz. Eine der wichtigsten Funktionen von HTML-Schablonen-WebItems ist ihre Fähigkeit, die Schablonendatei auszuwerten und dynamisch Text zu ersetzen. Sie können ein Tag-Präfix spezifizieren (der Standard ist `WC@`, aber das kann im WebItem-Eigenschaftenfenster geändert werden). Stellen Sie den Text, den Sie ersetzen möchten, einfach zwischen die Tags <WC@yourid> und </WC@yourid>.

Das MainPage-WebItem hat auch ein Textfeld und eine SUBMIT-Schaltfläche. Damit wird der Seite der folgende Text hinzugefügt:

```
<p>Here's a text box</p>

<form method="POST" action="--WEBBOT-SELF--">
  <!--webbot bot="SaveResults" startspan U-File="_private/form_results.txt"
  S-Format="TEXT/CSV" S-Label-Fields="TRUE" --><!--webbot bot="SaveResults"
                        endspan --><p>mytextfield
<input type="text" name="T1" size="20" value="replacethistext"></p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
```

Dann habe ich `WC@`-Tags hinzugefügt, um zwei Elemente auf der Seite einzufügen.

```
<p>Here's a text box</p>

<form method="POST" action="--WEBBOT-SELF--">
  <!--webbot bot="SaveResults" startspan U-File="_private/form_results.txt"
  S-Format="TEXT/CSV" S-Label-Fields="TRUE" --><!--webbot bot="SaveResults"
                endspan --><p><WC@firstfield>mytextfield</wc@firstfield>
```

```
<WC@secondfield>  <input type="text" name="T1" size="20" value=
                "replacethistext"></WC@secondfield></p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
```

Die Seite erscheint im FrontPage-Editor wie in Abbildung 27.4 gezeigt. Die Quadrate mit den Fragezeichen sind HTML-Markup-Tags – so zeigt FrontPage Tags an, die es nicht versteht, wie etwa diejenigen, die ich dem Text hinzugefügt habe. Beachten Sie, daß ich dem Textfeld auch Standardtext hinzugefügt habe. Das ist sehr wichtig, wie Sie gleich sehen werden.

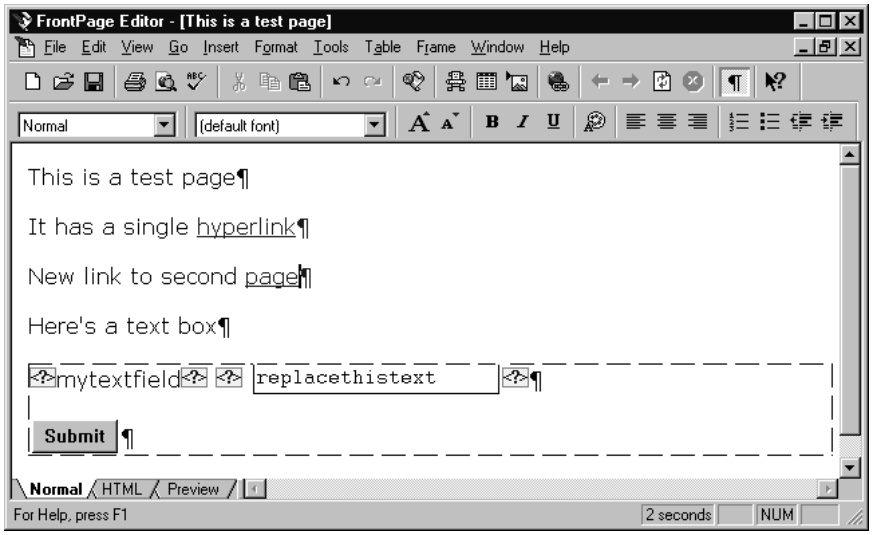

Abb. 27.4: MainPage, wie sie im FrontPage-Editor angezeigt wird

Wenn eine `WriteTemplate`-Operation ausgeführt wird, wird das `ProcessTag`-Ereignis ausgeführt, sobald ein Tag mit dem Präfix `WC@` gefunden wird. Das vollständige Tag erscheint im `TagName`-Parameter. Der `TagContents`-Parameter enthält den aktuellen Text zwischen den Tags.

```
Private Sub MainPage_ProcessTag(ByVal TagName As String, TagContents As _
            String, SendTags As Boolean)
   Select Case TagName
      Case "WC@firstfield"
         TagContents = "added in program"
      Case "wc@secondfield"
         TagContents = Replace(TagContents, "replacethistext", "current text")
   End Select
End Sub
```

Für das Tag `WC@firstfile` sieht man ganz einfach, was passiert. Der Text in der Schablonendatei (das ist hier `mytextfield`) wird durch den Satz *added in program* ersetzt.

Für das Tag `WC@secondfield` ist es etwas komplizierter. Ich will nur den Text im Textfeld ersetzen. Warum setze ich es dann so, daß das gesamte Eingabefeld enthalten ist, wie hier gezeigt?

```
<WC@secondfield>  <input type="text" name="T1" size="20" value=
                "replacethistext"></WC@secondfield></p>
```

Der Grund dafür ist, daß FrontPage ein ernsthaftes Problem mit der Verarbeitung von Tags innerhalb von Tags hat, und das gesamte Eingabefeld, das das Textfeld spezifiziert, ist ein einzelnes Tag. Um FrontPage bei Laune zu halten, verwende ich die Tags außerhalb. Wie kann ich dann genau den Text ersetzen, an dem ich interessiert bin? Ganz einfach! Ich habe den neuen Replace-Befehl verwendet, um einen einfachen Textersatz innerhalb des Strings vorzunehmen. Das Wort *replacethistext* wird in diesem Fall durch die Wörter *current text* ersetzt und wie in Abbildung 27.5 gezeigt ausgegeben.

Weitere Vorschläge. Verwenden Sie HTML-Schablonenelemente, statt WebItems, wann immer das möglich ist. Ich hasse es, HTML-Ausgabe fest zu codieren. Alle Ihre Web-Site-Editoren müssen dann Programmierer sein. Es könnte etwas weniger effizient sein, aber ich glaube, es ist viel besser, viele kleine HTML-Dateien als Schablonen zu verwenden, als eine große Menge fest codierten HTMLs innerhalb von WebItems.

Wenn Sie FrontPage einsetzen und Probleme mit dem Erkennen von Formulareingaben haben, sollten Sie sicherstellen, daß die Formulareigenschaften so gesetzt werden, daß die Daten an ein CGI-, ISAPI- oder ASP-Skript geschickt werden, und daß die Einstellung die Daten an das richtige WebItem weiterleitet. Visual Basic soll selbst dafür sorgen, aber es ist mir ein- oder zweimal passiert, wo das nicht der Fall war.

Microsoft empfiehlt, statt `WriteTemplate` besser `SetNextItem` zu verwenden, um von einem WebItem zum nächsten zu gelangen. Das ist wirklich ein guter Vorschlag. Wenn Sie `WriteTemplate` aufrufen, wird das Respond-Ereignis für das WebItem nicht ausgelöst. Sie möchten, daß Ihre Ausgabe zentralisiert wird, um die Abfolge steuern zu können. Das erscheint Ihnen vielleicht jetzt noch keinen Sinn zu machen, aber am Ende des nächsten Kapitels werden Sie genau verstehen, was ich meine.

Damit kommen wir zu einem Thema, das sehr wichtig für die Arbeit mit WebClasses ist, auch wenn es wenig mit WebClasses zu tun hat. Verwirrt? Das ist nur ein vorübergehender Geisteszustand, und als nächstes werden wir genau über den Übergang von einem Status in einen anderen sprechen.

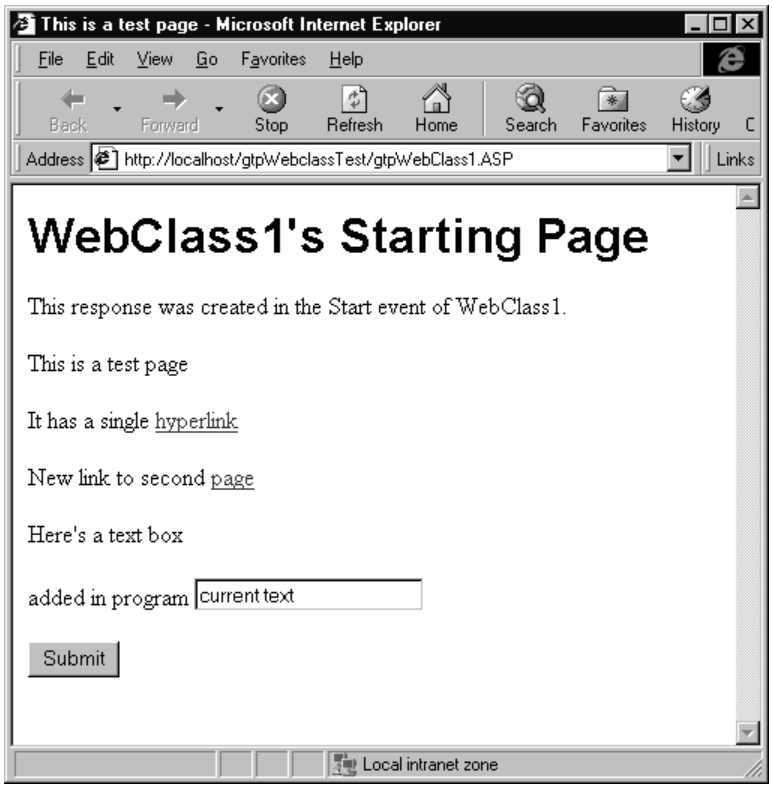

Abb. 27.5: Die Seite, wie sie vom MainPage-WebItem angezeigt wird