Kapitel 45

Notification Services

In diesem Kapitel:

Stellen Sie sich vor, Sie wären für die Weiterentwicklung einer *großen* Internet-Shop-Applikation zuständig, die von *vielen* Kunden genutzt wird. Ihre Geschäftsleitung hat sich dazu entschlossen, die Kommunikation mit den Kunden durch verschiedene Maßnahmen zu verbessern. Dazu gehören unter anderem:

- Ein Benachrichtigungsverfahren über den Zustand der Bestellungen. Dieses Order-Tracking soll die Kunden per E-Mail oder SMS über die Bearbeitung Ihrer Aufträge informieren.
- Kunden sollen automatisch und kurzfristig über Sonderangebote informiert werden können. Und zwar nur in den von Ihnen ausgewählten Warengruppen.
- Kunden sollen sich als Abonnenten für Infomailings eintragen können, über die Sie sich über neue Produkte informieren lassen können. Die Mailings können als HTML-Mails oder auch als PDF-Flyer bezogen werden.

Wie es sich für ein echtes EDV-Projekt gehört, soll die Umsetzung dieser Funktionen möglichst rasch vonstatten gehen. Die Zeit- und Kostenbudgets sind nicht gerade unglaublich üppig und dennoch werden die üblichen hohen Standards in Bezug auf Qualität, Performance und Ausfallsicherheit der Lösung gesetzt. Die ursprüngliche *netShop*-Anwendung soll so wenig wie möglich geändert werden müssen. Was tun?

Eine Antwort auf diese Frage könnte der Einsatz der *SQL Server Notification Services* (*SQL*NS) sein. Diese stellen ein Framework für die Implementierung von Benachrichtigungslösungen zur Verfügung. Der Schwerpunkt liegt dabei auf einer standardisierten Ablaufumgebung in Form von automatisch generierten Tabellen, Sichten, Gespeicherten Prozeduren, Diensten und Performance-Objekten. Das entlastet einen Entwickler bereits an vielen Stellen. Eine Benachrichtigungsanwendung, die mit der Hilfe der SQLNS entwickelt wird, lässt sich zu großen Teilen über Konfigurationsdateien deklarativ einrichten oder unter Einsatz des Objektmodells der Notification Services Management Objects (NMO) programmieren. Keine Sorge: Auch bei Einsatz der deklarativen Verfahren bleibt je nach Anwendungsfall noch einiges an Arbeit übrig. Dies betrifft vor allen Dingen die Schnittstellen zu anderen Systemen, aus denen Ereignisse bezogen werden sollen. Aber auch die Protokolle, die dem Verschicken von Informationen dienen, sind ein dankbares Gebiet für eigene Erweiterungen. Wenn Sie beispielsweise Nachrichten in Form von PDF-Dokumenten versenden möchten, dann müssen Sie sich beispielsweise damit beschäftigen, wie aus Informationen, die in einem XML-Format vorliegen, PDF-Dokumente generiert werden können. Falls Sie ein XSLT-Zauberer sind, stellt das keine größere Hürde dar. Wenn Sie beispielsweise auf Ereignisse reagieren möchten, die Ihnen von Ihrem Host-System über eine Message Queue (Nachrichtenwarteschlange) zur Verfügung gestellt werden, müssen Sie einen Adapter entwickeln, der diese Ereignisse als Nachrichten in Ihr SQLNS-System einträgt. Auch das ist nicht übermäßig kompliziert – aber es muss eben getan werden. Stellen Sie sich die Notification Services am besten als den *Kern* einer Maschine vor, die Nachrichten empfängt, verarbeitet und neue Nachrichten generiert. Dann haben Sie's so ungefähr.

Die Arbeitsweise der SQLNS wird inzwischen gerne als »Publish-Subscribe-Modell« bezeichnet. Das beschreibt recht gut, welche Klassen von Anwendungen besonders von diesem Framework profitieren können. In einem »Pull-Modell« holt der Empfänger die benötigten Informationen – am wahrscheinlichsten über eine Website – aktiv ab. Die meisten der heutzutage verwendeten Paketverfolgungssysteme arbeiten noch so. In einem »Push-Modell« dagegen werden Informationen an Empfänger verschickt. Häufig ist der Empfängerkreis unspezifisch (und fast noch häufiger sind die »Informationen« nicht gewünscht: Spam). Im »Publish-Subscribe-Modell« werden vom Anbieter so genannte

Abonnements zur Verfügung gestellt, für die sich ein Abonnent eintragen kann. Er kann sich dabei aussuchen, *was* ihm zugeschickt werden soll (über Filter), *wann* das passieren soll (zeitgesteuert oder aufgrund eines Ereignisses) und *wie* die Zustellung erfolgt (Kanal und Format). Typische Beispiele sind Nachrichten, die von Internet-Auktionshäusern oder Autobörsen verschickt werden.

Als gestandener Entwickler fragen Sie sich jetzt möglicherweise: Warum programmiere ich das Ganze nicht einfach selbst, anstelle das Notification Services-Framework zu benutzen? Schließlich gibt es tausend und eine Methode aus einer SQL Server-Applikation heraus zumindest E-Mails zu versenden. *SQL Server Database Mail* erlaubt das Verschicken von E-Mails über SMTP. Das geht sehr einfach, ist effektiv (weil asynchron) und sogar clusterfähig. Es gibt aber keine Formatierungsmöglichkeiten – einen Newsletter werden Sie damit nicht verschicken wollen. Die Reporting Services in Verbindung mit ihren Data Driven-Abonnements könnte eine Variante sein. Berichte können als Mailbody verschickt werden. Damit stehen Ihnen natürlich reichhaltige Formatierungsmöglichkeiten zur Verfügung. Über SMTP ist eine Push-Verteilung von Berichten machbar, die ebenfalls sehr effektiv arbeitet. Selbstverständlich könnten Sie über .NET-basierte gespeicherte Prozeduren die ganze Bandbreite des .NET-Frameworks nutzen und Ihre eigene Lösung entwickeln.

Die Beantwortung der Frage, welche Methode Sie einsetzen sollten, ist vom Umfang der Lösung und der Komplexität der Anforderungen abhängig. Neben dem – zugegeben etwas zu allgemeinen – Argument, dass Sie mit den Notification Services eine Technologie einsetzen, die von Microsoft für Sie entwickelt, getestet und auch in Zukunft gewartet wird, gibt es durchaus eine Reihe von interessanten Funktionen in SQLNS, die eventuell erst einmal entwickelt werden müssten.

Die Notification Services machen Ihnen die folgenden Angebote, über die es sich nachzudenken lohnt:

- **Funktionen:** Die SQLNS stellen clevere Funktionen wie ereignisgetriebene Abonnements, gesicherter Versand und Sprachunterstützung zur Verfügung.
- **Administrierbarkeit:** Die elementaren Operationen für die Einrichtung und Überwachung einer so genannten SQLNS-Instanz lassen sich mit dem Kommandozeilen-Utility *NSControl* oder bequem aus dem Objekt-Explorer des Management-Studios heraus erledigen. Über die mitgebrachten Performance-Counter kann die Leistung einer SQLNS-Anwendung überwacht werden und Fehlerzustände tauchen im Ereignisprotokoll des Servers auf.
- **Performance und Ausfallsicherheit:** Die Kernbestandteile der SQLNS arbeiten im Multithreading-Modus. Sie können, je nach Bedarf, die Anzahl der Threads erhöhen und an anderen Performance-Schrauben drehen. Scale-Out-Möglichkeiten sind (zumindest in der Enterprise-Edition) gleich mit eingebaut. Mehrere SQL Server-Instanzen lassen sich zu einer SQLNS-Farm zusammenschließen. Das ist sowohl für die Geschwindigkeit der Nachrichtengenerierung und -verteilung ein Booster, wie auch ein Mittel zur Erhöhung der Verfügbarkeit. Darüber hinaus lassen sich die Notification Services auch in einem Cluster betreiben.
- **Protokollierung:** Die Ereignisverarbeitung lässt sich bei Bedarf gut nachvollziehen. Da alle Informationen in normalen SQL Server-Tabellen gehalten werden, darf nach Herzenslust analysiert und ausgewertet werden.
- **Schnelles Prototyping:** Hält man sich an ein paar Grundregeln (wie zum Beispiel das in diesem Buch erläuterte »Kochrezept«), lässt sich ein erstes System in einem Funktionsprototyp erstaun-

lich schnell umsetzen. Auch wenn das Erstellen eines Prototyps an einem Tag, wie der große Bob[1] es verspricht, ganz klar nur dann machbar ist, wenn Sie sich schon gut auskennen.

- **Flexibilität:** Jede Medaille hat zwei Seiten. Es ist möglich, sich als Entwickler an vielen Stellen in die SQLNS einzuklinken, um vor allen Dingen mithilfe von .NET-Komponenten die Funktionalität zu erweitern. Die SQLNS sind gut darauf vorbereitet ein Host für solche Erweiterungen zu sein. Die Kehrseite: Viele Dinge *müssen* Sie eben auch selbst entwickeln. Wenn Sie zum Beispiel Nachrichten wahlweise in Formaten für SMS, WAP oder PDF anbieten möchten, dann benötigen Sie dazu die entsprechenden Stylesheets oder einen selbst geschriebenen Inhaltsformatierer.
- **Höhere Produktivität:** Wenn Sie ein paar der gerade vorgestellten Features nutzen möchten, dann wird der Einsatz von SQLNS Ihre Entwicklungszeit, die Sie für den *Kern* eines Benachrichtigungssystems aufwenden müssen, dramatisch senken.

Die Notification Services stellen für ausgeklügelte nachrichtenbasierte Anwendungen ein tolle Plattform dar. Sie sind vielleicht nicht gerade das Richtige für Entwickler mit einer ausgeprägten XML-Phobie (die kann man überwinden – glauben Sie mir: es lohnt sich). Was sich allerdings nicht so einfach wegdiskutieren lässt: Die Lernkurve bis zu Ihrer ersten SQLNS-Anwendung ist nicht eben flach. Die SQLNS stellen eine eigene Welt dar, mit einer Menge neuer Begriffe, Konzepten und Features. Leider auch mit einer Menge Begriffe, die »überlagert« sind und an anderen Stellen von SQL Server »so ähnlich« benutzt werden. Vor allen Dingen bei der Replikation. Da müssen Sie aufpassen. Die Architektur ist komplex und für das Definieren einer SQLNS-Applikation sind eine ganze Menge Optionen vorzubereiten und zu konfigurieren. Das Beste ist, Sie arbeiten sich da einmal Schritt für Schritt durch. Anschließend werden Sie dann auch besser verstehen was die SQLNS *nicht* leisten.

Notification Services-Grundlagen

Damit Sie im praktischen Teil dieses Kapitels wissen wovon die Rede ist, gibt es an dieser Stelle eine kurz gefasste Übersicht über die wichtigsten Bestandteile einer SQLNS-Lösung und die wesentlichen Konzepte, die beim Entwickeln eine Rolle spielen.

Übersicht über die Notification Services-Architektur

Die ersten beiden Grundbegriffe im Zusammenhang mit den SQLNS sind diejenigen einer Notification Services-*Instanz* und einer -*Anwendung*. Der Begriff einer Instanz wird ähnlich wie bei SQL Server selbst verwendet. Eine SQLNS-Instanz besteht – physikalisch betrachtet – aus einem Windows-Dienst und einer dazu gehörenden Datenbank, welche Konfigurationsdaten enthält, aber auch Informationen zur Verwaltung von Abonnenten. Dieser Dienst hat nichts mit den sonstigen SQL Server-Standarddiensten zu tun, sondern läuft vollkommen getrennt und existiert nur dann, wenn er konfiguriert und registriert wurde. So gut wie alle Arbeiten, die in einer SQLNS-Lösung anfallen, werden von dem zentralen Dienst erledigt.

---

[1] Bob Beauchemin – Autor und Blogger zum SQL Server 2005.

Ein SQLNS-Dienst kann eine oder mehrere SQLNS-Anwendungen bewirten. Über eine Anwendung wird die Art und Weise beschrieben, wie Benachrichtigungen entstehen. Die SQLNS sind eine sehr allgemein gehaltene Plattform für die Nachrichtenverarbeitung (Abbildung 45.1). Das übergeordnete Verarbeitungsmodell sieht folgendermaßen aus: Aus beliebigen Informationsquellen können *Ereignisse* an eine Notification Services-Anwendung übermittelt werden. Ereignisse sind Informationseinheiten, auf die eine SQLNS-Anwendung reagieren kann. Für die Übermittlung sind so genannte *Ereignisanbieter* verantwortlich. Die SQLNS stellen Standardanbieter zur Verfügung, mit denen XML-Dateien aus Verzeichnissen gelesen oder relationale Daten aus Datenbanken abgeholt werden können. Gehen die Anforderungen darüber hinaus, lassen sich eigene Ereignisanbieter implementieren.

Das Format, in dem die Ereignisse in einer SQLNS-Anwendung eintreffen und abgespeichert werden, wird über *Ereignisklassen* definiert. Als Entwickler gibt man einfach die Struktur einer Ereignisklasse vor (das ähnelt dem Anlegen einer Tabelle in einer Datenbank). Beim Konfigurieren der Anwendungen generiert SQL Server automatisch passende Tabellen, Sichten und weitere Objekte, die notwendig sind, um Ereignisse optimal zu verwalten.

Die *Abonnentenverwaltung* ist eine vom SQLNS-Dienst getrennte Anwendung, die Sie als Entwickler vollständig selbst realisieren müssen. In vielen Fällen wird die Abonnentenverwaltung aus einer ASP.NET-Oberfläche bestehen, in der die Benutzer ihre Abonnentendaten eintragen können. Zur Kommunikation mit den SQLNS wird eine einfache Abonnementverwaltungs-API zur Verfügung gestellt. Abonnenten können aus vorkonfigurierten Abonnements wählen (jedenfalls, wenn Ihre Anwendung das erlaubt), und es können individuelle Einstellungen für den Bezug von Abonnenments gemacht werden. Mit anderen Worten: Es kann pro Abonnent und Abonnement festgelegt werden, für welches *Ausgabegerät* die Nachrichten formatiert werden sollen, und es ist möglich, einem Abonnement Parameterwerte mitzugeben, über die sich die Auslieferung steuern lässt. So kann die Generierung von Nachrichten beispielsweise auf bestimmte Artikel eines Shops eingeschränkt werden.

In einer SQLNS-Anwendung prüft nun der *Generator* in konfigurierbaren Intervallen ob neue Ereignisse vorliegen, ob es Abonnements für diese Ereignisse gibt und ob die definierten Regeln für das Generieren von Benachrichtigungen greifen. Ist das der Fall, dann übergibt der Generator die Nachrichten in einem internen XML-Format an den *Verteiler.* Dieser rendert die Nachrichten in das gewünschte Zielformat und versendet sie über den vordefinierten Auslieferungskanal. Die Arbeit der Aufbereitung wird von einem *Inhaltsformatierer* geleistet. Als eingebauten Formatierer stellen die SQLNS den XSLT-Formatierer zur Verfügung. Gelingt es Ihnen, den gewünschten Output damit zu generieren, ist alles gut. Falls nicht, müssen Sie – Sie ahnen es – einen eigenen Inhaltsformatierer schreiben.

Sämtliche Bestandteile einer SQLNS-Anwendung werden in einer Applikationsdatenbank gespeichert. Genau wie bei der Instanz gilt: Ein Entwickler deklariert die verschiedenen Komponenten, und bei der Konfiguration werden die notwendigen Datenbankobjekte von SQL Server erzeugt.

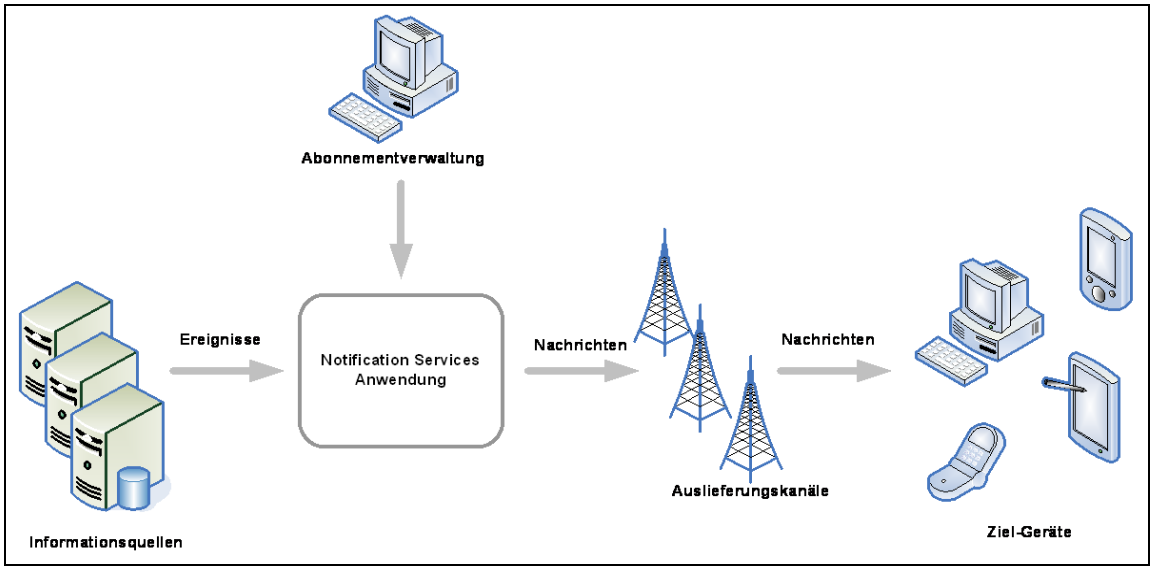

Abbildung 45.1 Prinzip einer Benachrichtigungsanwendung

Entwicklungswerkzeuge

Der vielleicht ungewöhnlichste Aspekt bei der Entwicklung einer SQLNS-Lösung ist die Art und Weise, wie man zu den Komponenten kommt. Neue SQLNS-Instanzen und -Anwendungen können über XML-Konfigurationsdateien definiert werden. Diese enthalten Beschreibungen der zu verwendeten Bestandteile. Die fertigen Konfigurationsdateien werden syntaktisch geprüft, kompiliert und – wenn alles in Ordnung
ist – der Windows-Dienst angelegt sowie die Datenbanken konfiguriert. Alternativ ist es möglich die Konfiguration mit den Notification Management-Objects (NMO) zu betreiben. Insgesamt werden Ihnen die folgenden Werkzeuge bei der Arbeit mit den Notification Services helfen:

- **nscontrol.exe:** Mein persönlicher »Favorit« unter den SQL Server-Werkzeugen ist die *Notification Services-Eingabeaufforderung*. Sie erreichen diese über *Start / Programme / SQL Server 2005 / Konfigurationstools*. Nach dem Anklicken dieses Befehls öffnet sich – ja, tatsächlich – eine Eingabeaufforderung und Sie befinden sich in dem Verzeichnis des Dienstprogramms *nscontrol.exe* (Abbildung 45.2). Dieses Kommandozeilen-Utility ermöglicht das Konfigurieren einer Instanz beziehungsweise einer Anwendung vom DOS-Prompt aus. *Nscontrols.exe* ist für das automatische Erstellen einer Lösung und für die Auslieferung interessant.

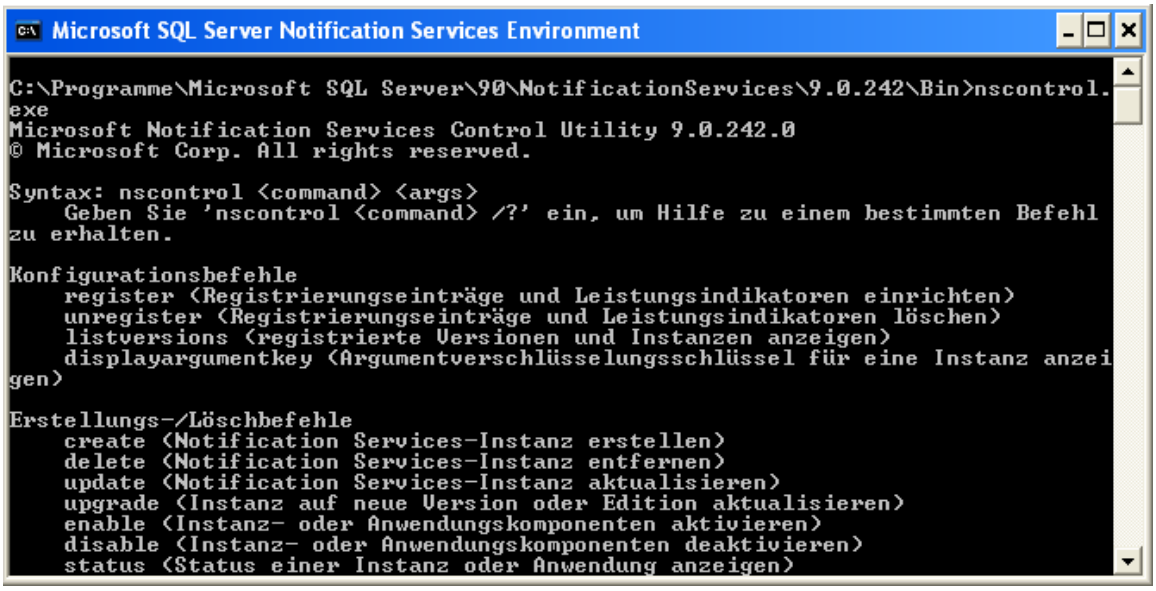

Abbildung 45.2 NSControl zeigt seine Kommandos

- **Management Studio:** Etwas bequemer arbeitet es sich im Management Studio. Unter dem Ordner *Notification Services* im Objekt-Explorer finden Sie die Funktionen von *nscontrol.exe* und noch ein paar mehr. Es sind nicht gerade üppig viele Möglichkeiten, aber man »kommt zurecht«. Falls Sie keinen speziellen XML-Editor verwenden, können Sie den eingebauten rudimentären Editor aus dem Management Studio einsetzen, um die Konfigurationsdateien einzurichten. Da *sämtliche* Informationen der SQLNS in den Tabellen der Instanz- und Anwendungsdatenbanken liegen macht es sich gut, dass Sie diese direkt im Objekt-Explorer öffnen können. An einigen wenigen Stellen werden Sie T-SQL-Abfragen formulieren. Insgesamt ist in der Entwicklung aber recht wenig mit T-SQL machbar.
- **Visual Studio 2005:** Eine Entwicklungsumgebung, wie Visual Studio 2005, benötigen Sie auf jeden Fall, um eine Verwaltungsanwendung für Abonnements zu schreiben. Da Sie aber auch COM-Komponenten dazu einsetzen können, tut es im Prinzip jede Entwicklungsumgebung für Windows- oder Web-Applikationen. Wenn Sie aber vorhaben einen eigenen Ereignisanbieter oder Inhaltsformatierer zu entwickeln, *dann* benötigen Sie auf jeden Fall ein Entwicklungswerkzeug für das .NET 2.0-Framework, da diese Erweiterungen darauf basieren.
- **XML-Stylesheetdesigner:** Haben Sie vor, Benachrichtigungen mithilfe des XSLT-Formatierers in ein Ausgabeformat wie PDF, RTF oder HTML zu überführen, dann werden Sie um ein entsprechendes Werkzeug kaum herumkommen. Gute WYSIWYG-Fähigkeiten bieten nur kommerzielle Produkte. Populär ist Stylevision von Altova, aber es gibt auch andere. Übrigens: Falls Sie nur einen brauchbaren XML-Editor suchen, der die Validierung über Schemata erlaubt und der zusätzlich die Möglichkeit der Vorschau von XSL Transformationen bietet, dann reicht auch Microsofts kleines Freeware-Tool XML Notepad 2.0.
- **Windows Ereignisanzeige:** Manchmal hat es ganz triviale Gründe, wenn Sie eine SQLNS-Lösung nicht zum Laufen bringen können. Es fehlt eventuell an ausreichenden Berechtigungen in einer

Datenbank oder dem Dateisystem oder ein Verzeichnisname ist falsch geschrieben. Auf die Spur kommen Sie solchen Problemen häufig erst, wenn Sie im Anwendungsprotokoll der Ereignisanzeige nach Einträgen des Notification Services-Dienstes Ihrer Test-Instanz gesucht haben.

- **Windows Performance-Monitor:** Eine feine Sache ist die Möglichkeit, die Leistung von SQLNS-Anwendungen anhand von Leistungsindikatoren überwachen zu können. Das werden Sie sicher noch nicht während der Entwicklung der Logik benötigen. Die Leitungsmessungen können Ihnen aber Aufschluss darüber geben, wann es Zeit wird ein Scale-Out vorzubreiten.

Mit den Notification Services entwickeln

Es ist wie so oft. Wen Sie das »erste Mal« hinter sich haben, dann erscheint alles ganz einfach und logisch. Am Anfang kann die Arbeit mit SQLNS aber ein bisschen auf die Nerven gehen. Das liegt nicht zuletzt an den etwas kargen Werkzeugen und der nicht gerade überwältigend klaren Dokumentation in den Books Online. Für die Erstellung Ihrer ersten Prototypen empfehle ich daher die Verwendung des folgenden Kochrezepts, dessen Anwendung anhand eines kleinen Szenarios auch in der Praxis vorgestellt wird. Das Beispielprojekt dazu finden Sie wieder im entsprechenden *Entwicklerbuch*-Verzeichnis. Damit Sie sich beim ersten Anlauf nicht verfransen, gilt am besten die Devise: »Keep it simple, Stupid« (KISS – he, ein neues »Four letter Word« für SQL Server!).

Kochrezept für eine SQLNS-Entwicklung

Die folgende Schritte müssen durchlaufen werden, um zu einer arbeitsfähigen Notification Services Lösung zu kommen. Wenn Sie anhand dieses Kochrezepts einen Prototypen vorbereitet haben, können Sie anschließend die Anwendung beliebig verfeinern.

1. **Das System planen:** Vor Beginn der Implementierung sollten Sie übersichtlich zusammen stellen, welche Instanzen und Anwendungen eingerichtet werden sollen, wie die Ereignisse und Nachrichten aussehen, zu welchen Zeitpunkten der Versand erfolgen soll, welche Regeln für die Generierung von Nachrichten gelten und welche Übertragungskanäle und Zielformate unterstützt werden sollen.
2. **Notification Services-Datenbanken vorbereiten:** Wie fast immer ist es besser, die benötigten Datenbank selbst zu erstellen und das nicht SQL Server zu überlassen.
3. **Eine erste Instanz und Applikation erstellen:** Sie sollten mit einem möglichst stark abgespeckten Prototypen beginnen, um den Rumpf *einer* Instanz und *einer* Applikation zu erstellten. Wenn dies fehlerfrei gelingt, dann fügen Sie nach und nach die weiteren Komponenten hinzu. Als ersten Auslieferungskanal legen Sie hier für das Testen am besten eine Datei fest. Wenn das System problemlos läuft, dann können Sie weitere Kanäle hinzufügen.
4. **Ereignisklassen definieren:** Über Ereignisklassen definieren Sie die Datenstrukturen der eintreffenden Nachrichten.
5. **Anlegen von Benachrichtigungsklassen:** In den Benachrichtigungsklassen legen Sie fest, welche Informationen verteilt werden sollen und welche Inhaltsformatierer die Nachrichten aufbereiten werden.

6. **Die Inhalts-Formatierung vorbereiten:** Wenn Sie mit XSLT-Formatierung arbeiten, dann sollten Sie jetzt ein erstes Stylesheet entwickeln. Falls Sie eigene Formatierer einsetzen wollen, dann ist es an der Zeit, diese anhand von Nachrichten im temporären XML-Zwischenformat zu programmieren.
7. **Anlegen von Abonnementsklassen:** Einerseits wird in einer Abonnementsklasse definiert, welche Informationen beim Registrieren für ein Abonnement angegeben werden müssen und andererseits können Sie hier die Regeln für den Bezug von Benachrichtigung definieren. Diese werden in Form von speziellen T-SQL-Befehlen definiert.
8. **Anlegen von Ereignisanbietern:** Für das Testen einer neuen Anwendung bietet sich die Verwendung des manuellen Generierens von Ereignissen an. Später sollten diese durch die »echten« Ereignisanbieter ersetzt werden.
9. **Generator-, Verteiler- und Applikationseinstellungen:** Hier werden Sie während der Entwicklung die Verarbeitungsintervalle heruntersetzen wollen, um Ihre Anwendung zügiger testen zu können.
10. **Den Windows-Dienst erzeugen:** Vor dem Einrichten des Windows Dienstes müssen Sie passende Konten in SQL Server und in den Datenbanken vorbereiten, über die sich der Dienst verbinden kann, um mit den Objekten der Instanz und der Applikation zu arbeiten. Gelingt das Erstellen und das Starten des Dienstes, dann sollten Sie dessen Zustand und den Zustand der Instanz und der Applikation überprüfen. Liegt alles im »grünen Bereich«, dann sind Sie nicht mehr weit vom Ziel entfernt.
11. **Abonnenten hinzufügen:** Um eine Anwendung testen zu können müssen Sie ein paar Abonnenten registrieren. Sie können dazu das Visual Studio einsetzen, um ein Testprogramm für die Registrierung zu schreiben oder – »quick and dirty« – einfache Skriptdateien verwenden.
12. **Testen der Anwendung:** Mithilfe von gespeicherten Prozeduren ist es möglich Ereignisdaten an eine Anwendung zu übertragen. Jetzt sollte eine entsprechende Ergebnisdatei generiert werden. Ist das nicht der Fall, dann gilt es die Fehlerprotokolle und relevanten Tabellen der SQLNS zu überprüfen.
13. **Abrunden der Anwendung:** Wenn Ihr Prototyp einwandfrei läuft, dann sollten Sie ihn Schritt für Schritt zu Ihrer gewünschten Anwendung ausbauen. Dafür gibt es vielfältige Möglichkeiten. Gehostete Ereignisanbieter können integriert und getestet werden. Zeitgesteuerte Abonnements können aktiviert werden. Fügen Sie den verschiedenen »Klassen« weitere Felder hinzu, erweitern Sie die Stylesheets, fügen Sie neue Kanäle hinzu, verwenden Sie realistischere Zeitpläne für die SQLNS-Vorgänge und so weiter und so weiter.

Implementierung einer einfachen Notification Service-Anwendung

Um zu zeigen, wie das SQLNS-Kochrezept in der Praxis funktioniert, soll jetzt eine kleine Lösung eingerichtet werden. Außer den vom SQL Server-Setup installierten Notification Services-Komponenten und dem Management Studio benötigen Sie weiter keine Hilfsmittel. Es werden XML-Konfigurationsdateien verwendet, weil sich mit diesen auch ohne Visual Studio arbeiten lässt und sich die Konzepte in den Dokumenten sehr klar widerspiegeln. Der Umstieg auf die Arbeit mit dem Objektmodel der NMO ist ein Klacks, da deren Objektmodell eng an die XML-Schemata der Konfigurationsdateien angelehnt ist.

Szenario: Ein Mailing-System für die netShop-Datenbank

Die Notification Services sollen an einem einfachen, überschaubaren Beispiel erläutert werden. Es geht darum, die bestehende *netShop*-Datenbank um elementare Benachrichtigungsfunktionen zu ergänzen. In der *netShop*-Datenbank selbst werden dazu nur die notwendigsten Daten verwaltet, den Rest erledigen die Notification Services.

Die Kunden der *netShop*-Anwendung sollen über interessante Produkte informiert werden. Die Mailings sollen per E-Mail oder auch in gedruckter Form abonniert werden können. Für die Verwaltung der Informationen die in den Mailings enthalten sein werden, reichen zwei neue Tabellen aus, die mit den Befehlen aus Listing 45.1 angelegt werden können. Die Tabelle *InfoMailíngs* enthält die Kerndaten zu einem Mailing: eine Bezeichnung (*Name*), das Datum für den Versand (*StartDate*), die Überschrift des Mailings (*Headline*) und den eigentlichen Text des Mailings (*MailingText*). In der zweiten Tabelle *InfoMailsToArticles* können die Schlüssel von Artikeln eingetragen werden, die mit dem Mailing besonders beworben werden sollen. Durch diese lassen sich die Informationen zu den Artikeln aus der Datenbank fischen. Außerdem finden Sie in dem Listing eine weitere kleine, aber ungemein wichtige Änderung der vorhandenen *netShop*-Datenstruk-tur – die es in jeder Benachrichtigungsanwendung geben sollte: Ein *NoMail*-Flag für Kunden, die nicht durch die Notification Services informiert werden möchten!

Die Mailverwaltung lässt sich natürlich leicht noch komfortabler gestalten. Für ein erstes Beispiel soll dieser Aufbau aber genügen. Es befinden sich keine Informationen zu den Abonnenten in der *netShop*-Datenbank. Diese sollen direkt von den Notification Services verwaltet werden.

```
ALTER TABLE SalesDepartment.Customers ADD  NoMail bit
GO

CREATE TABLE InfoMailings
(
   ID int IDENTITY PRIMARY KEY,
   Name varchar(100)
   MailingDate smalldatetime,
   Headline varchar(100),
   MailingText varchar(max)
)
GO
CREATE TABLE InfoMailsToArticles
(
   InfoMailID int,
   ProductID int,
   CONSTRAINT PK_InfoMailsToProducts PRIMARY KEY (InfoMailID, ProductId)
)
```

Listing 45.1 Die neuen Tabellen für die eMailings

Für das Ausprobieren finden Sie im Skript zu Listing 45.2 ein paar Beispieldatensätze. Der folgende Ausschnitt zeigt die Kommandos, mit denen das erste Mailing angelegt wird.

```
INSERT dbo.InfoMails ( Date, Headline, MailingText )
VALUES ( '01.08.2006', 'Sonderaktion gesunde Sommergetränke!', 'Liebe Kunden…)
INSERT dbo.InfoMailsToArticles VALUES ( 1, 39)
```

```
INSERT dbo.InfoMailsToArticles VALUES ( 1, 40)
INSERT dbo.InfoMailsToArticles VALUES ( 1, 41)
-- und so weiter…
```

Listing 45.2 Beispieldaten für das Erzeugen neuer Newsletter

In der *netShop*-Datenbank sind das schon alle notwendigen Änderungen. Es geht weiter mit dem Konfigurieren der Notification Services-Komponenten.

Das System planen

Für das *netShop*-System wird eine Instanz mit einer einzelnen Applikation geplant. Kommen später noch weitere Anwendungen hinzu, dann können auch diese innerhalb der Instanz laufen. Das ist eine übliche Konfiguration. Alles läuft auf einem einzigen Rechner und die Datenbanken befinden sich in der Standardinstanz von SQL Server. Für den Prototypen wird eine einfache Ausgabedatei als Kanal vorgesehen, später soll dann SMTP-Mail hinzukommen. Die Klassen der Anwendung werden in der ersten Version sehr simpel realisiert, sie entsprechen nahezu der Infomailings-Tabelle aus Listing 45.1.

Notification Services-Datenbanken vorbereiten

Obwohl die für die Benachrichtigungslösung benötigten Datenbanken auch durch die Notification Services selbst angelegt werden könnten, ist es doch etwas unkomplizierter sie auf dem gewohnten Weg einzurichten. Wo und wie die Datenbanken angelegt werden ist davon abhängig, welche Datenvolumen erzeugt werden. Die allgemeinen Kriterien sind in Kapitel 6 beschrieben worden. Für erste Experimente tun es zwei Mini-Datenbanken, je eine für eine neue Notification Services-Instanz und eine für die erste Notification Services-Anwendung. Wenn Sie nicht gerade vorhaben eine neue Lösung für eBay zu implementieren, dann ist das eine typische Blaupause für eigene Notification Services-Einrichtungen. In einer gemeinsamen Instanz werden verschiedene Applikationen verwaltet und diese teilen sich eine gemeinsame Datenbank. Ein Scale Out können Sie später immer noch ohne großen Aufwand durchführen.

Die Datenbanken werden mit den Befehlen aus Listing 45.3 angelegt, und dabei gibt es keine Besonderheiten:

```
CREATE DATABASE netShop_NS_Instance ON  PRIMARY
( NAME = N'netShopNSInstance', FILENAME = N'D:\Daten\netShopNSInstance.mdf',
  SIZE = 3072KB )
LOG ON
( NAME = N'netShopNSInstanceLog', FILENAME = N'D:\Daten\netShopNSInstanceLog.ldf',
  SIZE = 1024KB)

CREATE DATABASE netShop_NS_Applications ON  PRIMARY
( NAME = N'netShopNSApplications', FILENAME = N'D:\Daten\netShopNSApplications.mdf',
  SIZE = 3072KB )
LOG ON
( NAME = N'netShopNSApplicationsLog', FILENAME = 'D:\Daten\netShopNSApplicationsLog.ldf',
 SIZE = 1024KB )
```

Listing 45.3 Datenbanken für die Instanz und die Applikation

Eine erste Instanz und Applikation erstellen

Um eine Instanz vorzubereiten, benötigen Sie zwei XML-Dokumente. Mit dem ersten – dem *Instance Configuration File (*ICF*)* – werden die Grundeinstellung für eine Instanz gemacht. Da eine Instanz mindestens eine Applikation enthalten muss ist es erforderlich, auch gleich das entsprechende *Application Definition File (ADF)* anzulegen. Ein guter Weg ist der Start mit zwei minimalen Versionen der beiden Konfigurationsdateien, in denen nur diejenigen Elemente vorhanden sind, die laut Schema wirklich benötigt werden. Einige Elemente dürfen zwar leer sein, müssen aber dennoch in der richtigen Reihenfolge vorkommen, damit die Konfigurationsdateien kompiliert werden. Listing 45.4 zeigt, wie eine erste Konfigurationsdatei für die Instanz aussehen könnte.

Neben dem Namen für die neue Instanz muss vor allen Dingen angegeben werden, auf welchem SQL Server diese installiert werden soll und welche Datenbank für das Speichern der Instanzinformationen Verwendung findet. Im Element *<Applications>* werden der Instanz die Applikationen zugeordnet. Hier trägt man im Unterelement *<Application>* den Namen einer Applikation ein, den Pfad zu den Applikationsdateien und den Namen der Applikations-Definitionsdatei. Zu einer Instanz muss auf jeden Fall mindestens eine Anwendung definiert werden.

Den Abschluss bildet die Deklaration eines ersten Ausgabekanals. Für Testzwecke wird man in der Regel zunächst Ausgabedateien verwenden. In diesem Fall eine Datei, welche die gerenderten Daten im HTML-Format enthalten soll. Damit ist die erste Version des Konfigurationsdokuments bereits vollständig.

```
<NotificationServicesInstance xmlns:xsd=… xmlns="…/ConfigurationFileSchema">

  <InstanceName>netShop</InstanceName>       <!-- Bezeichung für die neue Instanz -->
  <SqlServerSystem>Vishnu</SqlServerSystem>  <!-- wo läuft die Instanz? -->
  <Database>
     <DatabaseName>netShop_NS_Instance</DatabaseName>  <!—Datenbankname -->
  </Database>

  <!-- Anwendung(en), die von dieser Instanz ausgeführt werden sollen -->
  <Applications>
    <Application>
      <ApplicationName>netShopInfoMailings</ApplicationName>
      <BaseDirectoryPath>C:\SQLEntwicklerbuch\Projekte\Kapitel … </BaseDirectoryPath>
      <ApplicationDefinitionFilePath>45.05.xml</ApplicationDefinitionFilePath>
     </Application>
  </Applications>

  <!-- Ausagabekanäle für diese Instanz -->
  <DeliveryChannels>
    <DeliveryChannel>
      <DeliveryChannelName>FileChannel</DeliveryChannelName>
      <ProtocolName>File</ProtocolName>
      <Arguments>
        <Argument>
          <Name>FileName</Name>
          <Value>D:\Mailings\Demo.htm</Value>
        </Argument>
      </Arguments>
    </DeliveryChannel>
```

```
  </DeliveryChannels>
</NotificationServicesInstance>
```

Listing 45.4 Instanz-Konfigurationsdokument für die netShop-Instanz

Das allererste Applikations-Definitionsdokument fällt sogar noch etwas karger aus (Listing 45.5). Zunächst sind nur die Elemente *<Generator>* und *<Distributors>* mit den Namen der lokalen SQL Server-Instanz gefüllt.

```
<Application xmlns:xsd=… xmlns="…/ApplicationDefinitionFileSchema">

   <!-- Abonnementsklassen -->
   <SubscriptionClasses></SubscriptionClasses>

   <!-- Benachrichtigungsklassen -->
   <NotificationClasses></NotificationClasses>

   <!-- Generator Einstellungen -->
   <Generator>
      <SystemName>Shiva</SystemName>
   </Generator>

   <!-- Verteiler Einstellungen -->
   <Distributors>
      <Distributor>
         <SystemName>Shiva</SystemName>
      </Distributor>
   </Distributors>

</Application>
```

Listing 45.5 Applikations-Definitionsdokument für die netShopInfoMails-Applikation

Der erste Testlauf für die ICF- und *ADF*-Dokumente kann nun mit den Funktionen im Management Studio durchgeführt werden. Eine Neue Instanz *registrieren* Sie im Object Explorer über das Kontextmenü des Ordners *Notification Services* und dem Befehl *Neue Notification Services Instanz ...* Dadurch werden vor allem die Datenbanken, die Tabellen für die Verwaltung von Instanz und Applikation vorbereitet. Kann der SQL Server die Informationen aus den Konfigurationsdateien ohne Fehler verarbeiten, dann bekommen Sie das so angezeigt, wie in Abbildung 45.3 zu sehen ist. Anschließend wird die neue Instanz im Ordner *Notification Services* angezeigt und Sie haben über deren Kontextmenü Zugriff auf die Verwaltungsbefehle.

Entgegen der sonstigen Gepflogenheiten der SQL Server-Programmierung gibt es keine Möglichkeit, eine Instanz mittels gespeicherter T-SQL-Prozeduren zu registrieren. Das geht tatsächlich nur über die Oberfläche des Management Studios, das Kommandozeilentool NS*Control* oder das Programmiermodell der Notification Services Management Objects.

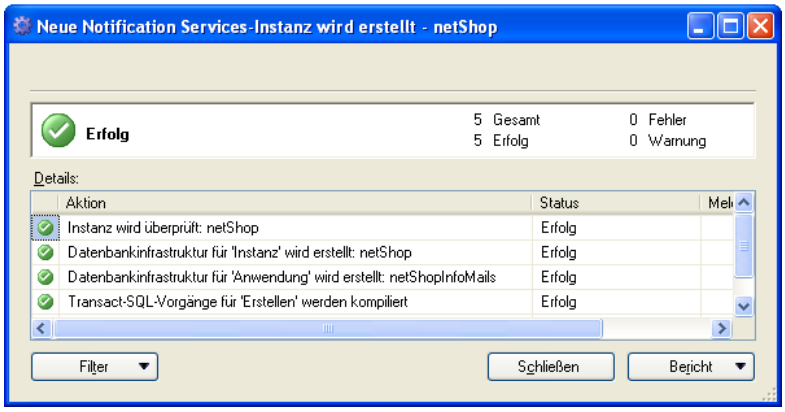

Abbildung 45.3 Feedback einer erfolgreichen Registrierung

TIPP Falls Sie beim Experimentieren mit den Notification Services einmal in die Situation geraten, dass eine »unsichtbare« Instanz entsteht – das ist eine Instanz, die zwar nicht im Management Studio angezeigt wird, aber in den Metadaten eingetragen wurde – dann können Sie keine neue Instanz unter diesem Namen anlegen. Eine lästige Situation! Der offizielle Weg wäre nun das Kommando *delete* des *nscontrol-Utilities* oder des GUIs zu verwenden. Das *kann* funktionieren. Falls nicht, dann verwenden Sie den folgenden Work Around.

Lassen Sie sich in der *msdb*-Datenbank die dort eingetragenen Notification Services-Instanzen anzeigen. Das funktioniert über das Kommando *SELECT * FROM NS90.NSInstanceInfo*. Dort sollten Sie den Namen der »klebrigen« Instanz finden. Gehen Sie jetzt knallhart vor und entfernen Sie die Information zu dieser Instanz mit einem Befehl à la *DELETE FROM NS90.NSInstanceInfo WHERE InstanceName = 'netShop'*. Ab jetzt werden Sie den Instanznamen wieder verwenden können. Dieses Vorgehen wird von Microsoft ausdrücklich nicht empfohlen, ist aber in manchen Situationen leider der einzige Ausweg.

Mit Parametern arbeiten

Eine praktische Angelegenheit, die das Arbeiten mit den Konfigurationsdateien etwas bequemer macht, ist die Möglichkeit, Parameter zu verwenden. Damit können Server- und Datenbanknamen, Dateipfade und alles andere im Kopf der Instanz-Konfigurationsdatei vereinbart werden. Bei einem Deployment auf einen zweiten Rechner müssen dann die Informationen nur an einer Stelle geändert werden. Außerdem können Sie direkt auf Umgebungsvariablen zugreifen und die Parameter lassen sich auch vom Management Studio aus mit aktuellen Werten füllen.

Parameter werden schlicht und ergreifend dadurch gekennzeichnet, dass Sie einen Bezeichner in einer Konfigurationsdatei in Prozentzeichen einschließen. So ist *%myServer%* also ein Parameter, der durch einen Wert vorbelegt werden kann. Das geschieht am Anfang einer Konfigurationsdatei innerhalb des Elements *<ParameterDefaults>*. Ein Parametername, welcher der Bezeichnung einer Umge-

bungsvariablen entspricht, wird direkt aus dieser gefüllt. Dafür ist *%Computername%* ein gutes Beispiel. In der zweiten Version der ICF-Datei nach Listing 45.6 werden Parameter verwendet, um die Konfiguration einfacher an neue Gegebenheiten anpassen zu können (in diesem Listing wurden unveränderte Teile weggelassen).

Interessant ist der Parameter-Block, der innerhalb der Applikations-Deklaration durch das Element *<Parameters>* gebildet wird. An dieser Stelle ist es möglich Parameter an eine Applikation durchzureichen. So muss man Parameterwerte nur einmal – in der Konfigurationdatei nämlich – festlegen. In der Applikations-Definitionsdatei kann auf die Parameter nach Listing 45.6 wiederum über die Prozentschreibweise zugegriffen werden. Der Ausschnitt der Definitionsdatei in Listing 45.7 zeigt, wie das geht.

```
<!-- Standardwerte für die Parameter der Instanz und globale Applikationsparameter -->
   <ParameterDefaults>
      <Parameter>
         <Name>myServer</Name>         <!-- SQL Server Instanz, die als Host dient -->
         <Value>%COMPUTERNAME%</Value> <!-- = Standard SQL-Server-Instanz          -->
    </Parameter>
    <Parameter>
      <Name>myOutputDir</Name>          <!-- Verzeichnis für die Ausgabedateien  -->
      <Value>D:\Mailings</Value>
    </Parameter>
    <Parameter>
      <Name>myInstanceDatabase</Name>   <!-- Datenbank, für die Instanz -->
      <Value>netShop_NS_Instance</Value>
    </Parameter>
    <Parameter>
      <Name>mySystemDir</Name>          <!-- Verzeichnis für *alle* Dateien -->
      <Value>C:\SQLEntwicklerbuch\Projekte\Kapitel 45 - Notification Services</Value>
    </Parameter>
  </ParameterDefaults>
…
 <!-- Anwendung(en), die von dieser Instanz ausgeführt werden sollen -->
  <Applications>
    <Application>
      <ApplicationName>netShopInfoMails</ApplicationName>
      <BaseDirectoryPath>%mySystemDir%</BaseDirectoryPath>
      <ApplicationDefinitionFilePath>45.07.xml</ApplicationDefinitionFilePath>
      <Parameters>
        <Parameter>
          <Name>myApplicationServer</Name>
          <Value>%myServer%</Value>
        </Parameter>
        <Parameter>
          <Name>myApplicationDir</Name>
          <Value>%mySystemDir%</Value>
        </Parameter>
      </Parameters>
    </Application>
  </Applications>
…
```

Listing 45.6 Instanz-Konfigurationsdokument mit Parametern

```
…
  <Generator>
    <SystemName>%myApplicationServer%</SystemName>
  </Generator>
…
```

Listing 45.7 Verwendung eines Parameters in der ADF-Datei

Nachdem Sie Änderungen an den Konfigurationsdateien vorgenommen haben, müssen Sie die vorhandene Instanz aktualisieren, um diese zu testen. Dazu verwenden Sie den Befehl *Tasks / Aktualisieren …* aus dem Kontextmenü einer Instanz im Management Studio. Befinden sich Parameter in der ICF-Datei, dann werden Ihnen diese im Dialog *Instanz aktualisieren* angezeigt und Sie könnten die Werte jetzt noch anpassen (Abbildung 45.4). Nach dem Start der Aktualisierung bekommen Sie in einer kurzen Zusammenfassung zu sehen, was SQL Server zu tun gedenkt, und durch einen Klick auf *Aktualisieren* in diesem zweiten Dialog nimmt SQL Server die Änderungen vor.

Wenn Sie das Beispiel mitarbeiten möchten: Jetzt und auch bei allen späteren Aktualisierungen verwenden Sie *immer* die ICF-Datei *45.06.xml*. Diese wird ab jetzt nicht mehr verändert. Es ändert sich allerdings jeweils die referenzierte *ADF*-Datei. In diesem Fall muss die *ADF* 45.07 eingebunden werden. Ändern Sie also den entsprechenden Abschnitt in der ICF-Datei in *<ApplicationDefinitionFilePath>45.07.xml</ApplicationDefinitionFilePath>*. Vergessen Sie nicht vor dem Aktualisieren die beiden Dateien erneut zu speichern.

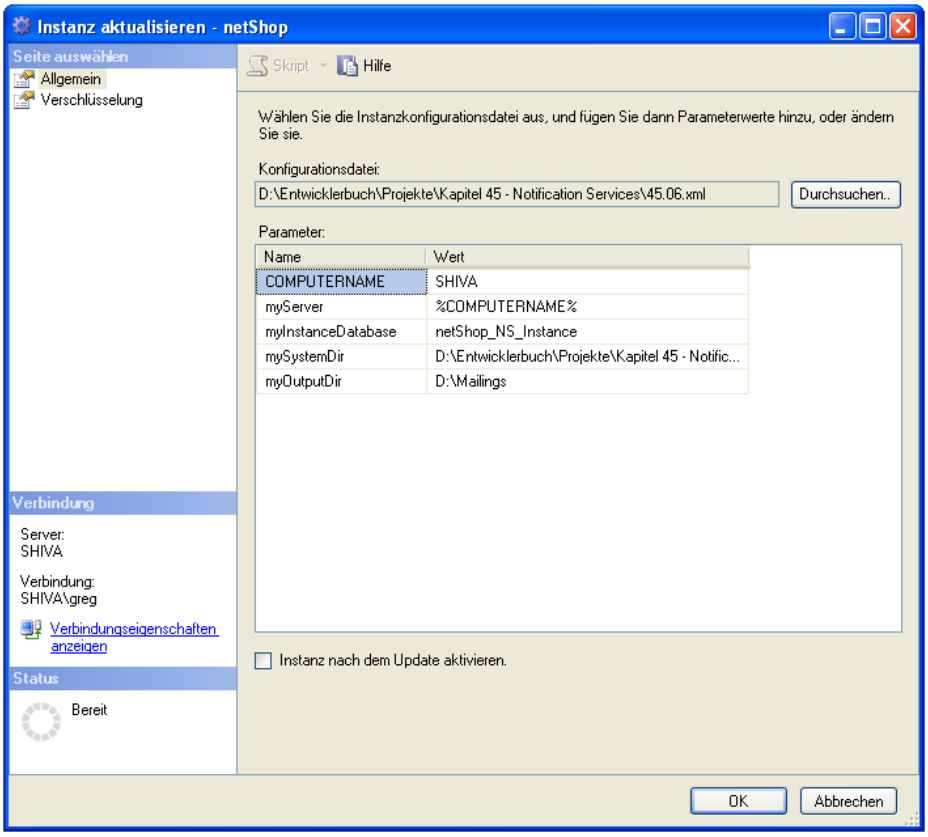

Abbildung 45.4 Dialog Instanz aktualisieren

Anlegen einer Ereignisklasse

Im nächsten Schritt legen Sie jetzt am Besten fest, mit welchen Ereignisdaten Sie in Ihrer Anwendung arbeiten möchten. Die Felder einer Ereignisklasse bestimmen, wie die Informationen beschaffen sind, die an Ihre Applikation übertragen werden können. Es ist ohne weiteres möglich mit mehreren Ereignisklassen zu arbeiten.

Das Anlegen einer Ereignisklasse ähnelt ein wenig dem Anlegen einer SQL Server-Tabelle. Und letzten Endes ist dies auch genau das, was beim Kompilieren der ACF-Datei passiert. Es wird eine neue Tabelle sowie eine Sicht in der Applikationsdatenbank angelegt. Sie können sogar Indizes für häufig genutzte Felder definieren. Welche Attribute Ihre Ereignisklasse enthält ist nicht davon abhängig, wie die Datenquellen aussehen. Die Felder *können* den Spalten von Tabellen entsprechen, aus denen Daten für die Notification Services entnommen werden, müssen es aber nicht. Die Daten der Ereignisklassen werden in der Applikationsdatenbank gesammelt und von dort aus weiter verarbeitet.

In dem einfachen Anwendungsbeispiel der *netShop*-Datenbank ist das Anlegen der Ereignisklasse eine übersichtliche Angelegenheit. Es werden die Ereignisfelder *MailingDate*, *Headline* und *MailingText* für die Gestaltung einer Nachricht benötigt. Letzten Endes sind das natürlich die Felder aus der Tabelle *InfoMailings*. Die Definition der Eventklasse gehört in das Element *<EventClasses>* einer *ADF*-Datei.

Wie beim Anlegen einer SQL Server-Tabelle müssen für die Felder eine Name, ein Datentyp und weitere Eigenschaften angegeben werden. Die Datentypen sind hier die ganz normalen SQL Server-Datentypen und *nicht* etwa XML-Datentypen.

```
<!-- Ereignisklassen -->
  <EventClasses>
    <EventClass>
      <EventClassName>MailingData</EventClassName>
      <Schema>
        <Field>
          <FieldName>MailingDate</FieldName>
          <FieldType>smalldatetime</FieldType>
          <FieldTypeMods>not null</FieldTypeMods>
        </Field>
        <Field>
          <FieldName>Headline</FieldName>
          <FieldType>varchar(100)</FieldType>
          <FieldTypeMods>not null</FieldTypeMods>
        </Field>
        <Field>
          <FieldName>MailingText</FieldName>
          <FieldType>varchar(max)</FieldType>
          <FieldTypeMods>not null</FieldTypeMods>
        </Field>
      </Schema>
    </EventClass>
  </EventClasses>
```

Listing 45.8 Definition einer Ereignisklasse

Vergessen Sie nach der Definition der Ereignisklasse das Aktualisieren der Instanz nicht! Ändern Sie in der ICF-Datei den Bezug auf die *ADF*-Datei *45.08.xml*.

Die Inhaltsformatierung vorbereiten

Der Standard Inhaltsformatierer der Notification Services verwendet benutzerdefinierte XSLT-Stylesheets um die zu versendenden Nachrichten in das gewünschte Ausgabeformat zu bringen. XSLT-Stylesheets bringen die vom Verteiler produzierten XML-Daten in das gewünschte Ausgabeformat wie HTML, PDF, Word und so weiter. XSL-Transformationen sind eine wirkungsvolle Methode, den Output für verschiedene Medien zu produzieren. Allerdings ist nicht jeder Entwickler ein begeisterter XML-Anhänger. Als Alternative bietet sich dann ein eigener Formatierer an – in Form eines gehosteten .NET-Assemblies.

Die Arbeit mit XSLT-Stylesheets ist allerdings *so* kompliziert auch wieder nicht und kann sogar richtig Spaß machen, wenn man als Werkzeug nicht nur einen einfachen ASCII-Editor zur Verfügung hat, sondern ein professionelles Werkzeug für die Stylesheet-Entwicklung. Für das simple *netShop*-Demoszenario reicht allerdings auch das gute alte Notepad aus.

Die Daten, die von einem Inhaltsformatierer verarbeitet werden sollen, erreichen diesen in einem übersichtlich aufgebauten XML-Datenstrom. Damit ein Stylesheet außerhalb der Notification Services-Umgebung entwickelt werden kann ist es sinnvoll, sich eine einfache XML-Datei in dem genutzten Zwischenformat zu Recht zu legen. Ein Beispiel finden Sie in Listing 45.9. Verschachtelt in das äußere

Element *<notifications>* sind die einzelnen Benachrichtigungs-Datensätze in den Elementen *<notification>*. Deren Struktur wiederum wird durch die Definition von so genannten Benachrichtigungsklassen vorgegeben. Das Vorgehen dabei entspricht nahezu demjenigen beim Anlegen einer Ereignisklasse. Keine Geheimnisse an dieser Stelle (die Details zu den Benachrichtigungsklassen folgen im nächsten Abschnitt).

```
<notifications>
    <notification>
      <MailingDate>01.08.2006</MailingDate>
      <Headline>Sonderaktion gesunde Sommergetränke!</Headline>
      <MailingText>Liebe Kunden - es gibt gute Nachrichten!...</MailingText>
    </notification>
    <notification>
      <MailingDate>01.09.2006</MailingDate>
      <Headline>Wir erweitern unser Sortiment</Headline>
      <MailingText>Liebe Kunden - Neuigkeiten von Ihrem netShop!...</MailingText>
    </notification>
</notifications>
```

Listing 45.9 Beispieldaten aus der netShop-Applikation

Müssen Sie das Stylesheet »zu Fuß« entwickeln, dann können Sie als »Testumgebung« einen Browser, wie Firefox oder Internet Explorer (ab Version 6.X) verwenden. Ergänzen Sie dann in der Testdatendatei einfach eine Zeile mit einem Verweis auf die Stylesheet-Datei. So in der Art: *<?xml-stylesheet type="text/xsl" href="45.10.xsl"?>*. Beim Öffnen der Testdatendatei wird dann der Browser angewiesen, die Stylesheet-Transformationen durchzuführen und Sie bekommen die gerenderten Daten zu sehen. Das Ergebnis der serverseitigen Transformationen kann eventuell leicht davon abweichen; das ist vom jeweils verwendeten XML-Parser abhängig.

Ein einfaches Test-Stylesheet finden Sie in Listing 45.10. XSLT-Stylesheets an dieser Stelle zu erklären macht wenig Sinn und kostet zu viel Platz. In den Ressourcen zu diesem Buch finden Sie ein paar schöne Online-Quellen dazu. Es gibt natürlich auch jede Menge guter Bücher zu diesem Thema. Falls Sie noch nicht mit XSLT gearbeitet haben nur so viel: XSLT-Stylesheets bestehen aus einer Anzahl so genannter Templates, die auf die Knoten des XML-Dokumentenbaums angewendet werden. Im Beispiel findet man zwei Templates, das erste, nämlich *<xsl:template match="notifications">* bezieht sich auf das Element *<notifications>*, also der obersten Ebene des XML-Benachrichtigungsdokuments. An dieser Stelle werden allgemeine Informationen ausgegeben, die sich noch nicht auf einzelne <notification>–Elemente beziehen. Außerdem wird über die Anweisung *<xsl:apply-templates/>* die Verarbeitung der Kind-Knoten zu *<notifications>* angeschoben. Das zweite Template verarbeitet diese <notification>-Elemente. Die *value-of select*-Anweisungen dienen dazu, den Inhalt eines Quell-Elements in das Ziel zu übernehmen. So wird beispielsweise die Überschrift aus dem Element *<Headline>* in HTML-h3-Tags verpackt.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="notifications">
      <html>
         <head><title>netShop Infomails</title></head>
         <body>
            <h1>InfoMail-Testdokument</h1>
            <xsl:apply-templates/>
```

```
        </body>
    </html>
  </xsl:template>

   <xsl:template match="notification">
      <h2>netShop-InfoMail vom <xsl:value-of select="MailingDate"/></h2>
      <h3><xsl:value-of select="Headline"/></h3>
      <xsl:value-of select="MailingText"/>
   </xsl:template>

</xsl:stylesheet>
```

Listing 45.10 Test-Stylesheet für das netShop-InfoMailystem

Anlegen einer Benachrichtigungsklasse

Über eine Benachrichtigungsklasse wird sozusagen das andere Ende einer Notification-Applikation definiert – das, was »hinten dabei rauskommt«: Die Struktur der Benachrichtungsinformationen nämlich. Genau, wie beim Anlegen der Ereignisklassen werden durch die Definition von Benachrichtigungsklassen interne Tabellen, Sichten und gespeicherte Prozeduren vorbereitet, die dem Sammeln und in diesem Fall dem Verschicken von Nachrichten dienen.

Zusammen mit der Datenstruktur müssen auf jeden Fall der zu verwendende Inhaltsformatierer und ein Auslieferungsprotokoll angegeben werden. Im einfachsten Fall – und dieser soll jetzt implementiert werden – stellen die Felder der Benachrichtungsklasse eine Teilmenge einer Ereignisklasse dar oder sind sogar mit dieser identisch, wie im Listing 45.11. Im Beispiel-Listing können Sie erkennen, dass der Standard-Formatierer (»XsltFormatter«) eingesetzt wird und das gerade vorgestellte Stylesheet Verwendung finden soll.

```
<!-- Benachrichtigungsklassen -->
<NotificationClasses>
  <NotificationClass>
    <NotificationClassName>InfoMailings</NotificationClassName>
      <Schema>
        <Fields>
          <Field>
            <FieldName>MailingDate</FieldName>
            <FieldType>smalldatetime</FieldType>
            <FieldTypeMods>not null</FieldTypeMods>
          </Field>
          <Field>
            <FieldName>Headline</FieldName>
            <FieldType>varchar(100)</FieldType>
            <FieldTypeMods>not null</FieldTypeMods>
          </Field>
          <Field>
            <FieldName>MailingText</FieldName>
            <FieldType>varchar(max)</FieldType>
            <FieldTypeMods>not null</FieldTypeMods>
          </Field>
        </Fields>
      </Schema>
```

```
      <ContentFormatter>
        <ClassName>XsltFormatter</ClassName>
        <Arguments>
          <Argument>
            <Name>XsltBaseDirectoryPath</Name>
            <Value>%myDir%</Value>
          </Argument>
          <Argument>
            <Name>XsltFileName</Name>
            <Value>45.10.xsl</Value>
          </Argument>
        </Arguments>
      </ContentFormatter>
    <Protocols>
      <Protocol>
        <ProtocolName>File</ProtocolName>
      </Protocol>
     </Protocols>
  </NotificationClass>
</NotificationClasses>
```

Listing 45.11 Definition einer Benachrichtungsklasse

Wenn Sie hier noch »mitspielen« möchten (was ich sehr hoffe), dann denken Sie bitte daran, in der ICF-Datei den Bezug auf die *ADF*-Datei *45.11.xml* zu setzen, und aktualisieren Sie das Ganze ein weiteres Mal.

Anlegen einer Abonnementsklasse

Die Verarbeitung von Abonnements kann auf zwei unterschiedlichen Wegen eingeleitet werden: Entweder direkt ausgelöst durch das Eintreffen neuer Ereignisdaten oder zeitgesteuert in vorgegebenen Intervallen. Für das Testen besser geeignet ist natürlich das ereignisgesteuerte Verfahren. Durch das Hineinpumpen entsprechender Daten kann das erfolgreiche Generieren der erwarteten Nachrichten sofort kontrolliert werden. Aus diesem Grund findet man im XML-Fragment in Listing 45.12 kein *<ScheduledRules>*-Element, welches für das Festlegen von zeitgesteuerten Regeln angewendet wird.

Damit überhaupt Benachrichtigungen generiert werden, wenn neue Ereignisse eintreffen, müssen allerdings auf jeden Fall Regeln definiert werden (mindestens eine). In unserem Beispiel befindet sich genau eine Regel im Element *<EventRules>*. Hier werden die *ereignisgesteuerten* Regeln angelegt. Regeln sind nichts anderes als ein oder mehrere T-SQL-Befehle, die Daten in die Datenstruktur einer Benachrichtigunsklasse einfügen. Datenquellen und Datenziele sind nichts anderes als Tabellen und Sichten, die von SQL Server beim Konfigurieren einer Anwendung automatisch anhand der Beschreibungen der Ereignis-, Benachrichtigungs- und Abonnementsklassen definiert werden. Zu jeder Regel muss mindestens ein Name vergeben, sowie der Bezug auf eine Ereignisklasse hergestellt werden, auf welche sich die neue Regel bezieht. Das passiert in der Zeile *<EventClassName>InfoMailingsEvent</EventClassName>*.

Die eigentlichen Anweisungen der Regel befinden sich unter *<Action>*. Aktionen *können*, müssen aber nicht die Abonnententabelle und die Ereignissicht abfragen. Sie *können* Benachrichtigungen generieren, indem Sie neue Datensätze in eine Benachrichtigungstabelle einfügen. Im Prinzip sind Sie völlig frei in der Gestaltung der T-SQL-Befehle. Wenn Benachrichtigungen erzeugt werden sollen, muss

aber mindestens ein Kommando neue Datensätze in die Benachrichtigungstabelle einfügen. Deren Name entspricht dem Namen einer der verwendeten Benachrichtigungsklassen. Das Einfügen geschieht dann, wie in Listing 45.12, einfach durch ein passendes *INSERT INTO*. Da bei der Verarbeitung einer Aktion sowohl die Daten der neuen Ereignisse, wie auch die Daten des Abonnenments zur Verfügung stehen, lassen sich die einzuhaltenden Regeln leicht definieren. Die Regel im Beispiel verhindert, dass Empfänger veraltete eMailings erhalten. Das ist für eine ereignisgesteuerte Regel nicht so interessant, da sowieso nur Benachrichtigungen auf der Grundlage von eintreffenden Ereignissen generiert werden können. Später könnte aber eine zeitgesteuerte Regel angewendet werden, die potenziell auch auf alte Mailings zugreifen könnte.

Beachten Sie bei der Gestaltung von Aktionen, dass es Pflichtfelder für das *INSERT INTO* gibt, die Sie auf jeden Fall angeben müssen, damit eine Verarbeitung der Regel stattfindet. Die ersten drei Spalten müssen – in dieser Reihenfolge – *SubscriberID, DeviceName* und *SubscriberLocale* sein. Wenn in Ihrer Applikation ein Gerätename oder eine Gebietsdefinition keine Rolle spielt, dann setzen Sie hier einfach Standardwerte ein, wie es das Beispiel zeigt.

```
<!-- Abonnementsklassen -->
  <SubscriptionClasses>
    <SubscriptionClass>
      <SubscriptionClassName>SubMailings</SubscriptionClassName>
      <Schema>
        <Field>
          <FieldName>StartDate</FieldName>
          <FieldType>smalldatetime</FieldType>
          <FieldTypeMods>not null default getdate()</FieldTypeMods>
        </Field>
      </Schema>
      <EventRules>
        <EventRule>
          <RuleName>InfoMailingsRule</RuleName>
          <EventClassName>NewMailings</EventClassName>
          <Action>
            INSERT INTO MailingsToSend
            ( SubscriberID, DeviceName, SubscriberLocale, MailingDate, Headline,
              MailingText )
            SELECT
              ims.SubscriberID, 'MyDevice', 'de-de', md.MailingDate, md.Headline,
              md.MailingText
            FROM
              NewMailings md, SubMailings ims
          </Action>
        </EventRule>
      </EventRules>
    </SubscriptionClass>
  </SubscriptionClasses>
```

Listing 45.12 Definition einer Abonnementsklasse

Haben Sie es »erraten«? Sie müssen die Instanz mit der ICF-Datei *45.06.xml* aktualisieren, in der Bezug auf die *ADF*-Datei *46.12.xml* genommen wird. Passen Sie die Datei entsprechend an.

Anlegen eines Ereignisanbieters

Die Notification Services bieten drei Standardereignisanbieter:

- **File System Watcher-Ereignisanbieter:** Überwacht ein Dateisystemverzeichnis. Liest XML-Dateien ein, die dort abgelegt werden.
- **SQL Server-Ereignisanbieter:** Verwendet T-SQL-Befehle um in regelmäßigen Intervallen Informationen aus beliebigen Datenbanken abzuholen. Sie sind als Entwickler vollständig dafür verantwortlich Abfragen zu entwerfen, welche die Daten filtern und vermeiden, zu viele Ereignisdaten aus den Basistabellen abzuholen. Große Datenmengen abzuholen, die erst durch Abonnementregeln gefiltert werden, ist natürlich schlecht für die Leistung des Systems, und doppelt abgeholte Ereignisse werden auch doppelt verarbeitet.
- **Analysis Services-Ereignisanbieter:** Dieser überwacht die Daten in einem Analysis Services Cube und generiert Ereignisse auf der Grundlage eines MDX-Ausdrucks (MDX = *Multidimensionale Ausdrücke*, eine der Abfragesprache der Analysis Services).

Da man es sich während der Entwicklung auch beim Vorbereiten des Event Providers des ersten Applikationsprototypen möglichst einfach machen sollte, wird hier ein eigener Provider eingesetzt. Und zwar ein nicht gehosteter Ereignisanbieter (also kein in .NET geschriebener Anbieter). Dadurch ist es möglich, Ereignisse zu einem beliebigen Zeitpunkt mit T-SQL-Mitteln zur Verfügung zu stellen. Die Definition eines externen Providers fällt kurz und knapp aus, wie in Listing 45.13 zu sehen. Es muss nur ein Name vergeben werden, auf den später verwiesen werden kann. Das ist dann schon alles. Die Daten werden später mit der Hilfe von vorbereiteten gespeicherten Prozeduren der Anwendung eingefügt.

```
<!-- Event Providers-->
<Providers>
   <NonHostedProvider>
      <ProviderName>InfoMailsEventProviderSP</ProviderName>
   </NonHostedProvider>
</Providers>
```

Listing 45.13 Externer Event Provider

Und jetzt bitte das Aktualisieren mit dem neuen *ADF*-Dateinamen nicht vergessen, um die Einstellungen prüfen zu lassen.

Generator-, Verteiler- und Applikationseinstellungen

So, jetzt sind wir schon fast fertig! Im Grunde sind die Instanz und die Anwendung jetzt schon vollständig konfiguriert. Damit das Testen leichter fällt, sollen jetzt noch die Intervalle für die Aktivierung von Generator und Verteiler angepasst werden. So hat der Generator als Grundeinstellung ein Arbeitsintervall von 1 Minute. Das kommt jedem Entwickler bei einem Test wie eine Ewigkeit vor. Das Intervall für den Generator, *Quantumdauer* genannt, legt fest, wie häufig Abonnementsregeln überprüft und Benachrichtigungen generiert werden. Die Quantumdauer für den Verteiler legt fest, wie häufig dieser aufgerufen wird, um Nachrichten zu rendern und auszuliefern. Die kurzen Vorgaben in Listing 45.14 sind für das Testen gut, können aber im Echtbetrieb zu Problemen führen, wenn die Zeitspannen nicht

mehr ausreichen, um die Nachrichten zu verarbeiten. Die Zeiten sollten so großzügig wie möglich bemessen sein.

Weiter Einstellungen für Generator und Verteiler betreffen das Performance-Tuning. Sie können die Zahl der gleichzeitig verfügbaren Arbeitsthreads erhöhen oder die Komponenten auf andere Rechner verlegen, um die Arbeitslast zu verteilen.

Zu den Ausführungseinstellungen der Applikation gehört das Element *<Vacuum>*. Diese »Staubsaugereinstellung« *muss* irgendwann einmal gemacht werden. Ist Sie nicht vorhanden, dann werden die veralteten Daten einer Anwendung *nie* gelöscht. Im Testbetrieb ist es sinnvoll, die Daten ein paar Tage aufzuheben, da die Informationen im Klartext lesbar sind und für die Fehlersuche genutzt werden können. Die Einstellungen im XML-Ausschnitt legen eine Bewahrungsdauer von 14 Tagen fest. Das Aufräumen veralteter Daten findet jeden Morgen um 01:00 h statt.

```
<!-- Generator Einstellungen -->
  <Generator>
    <SystemName>%myApplicationServer%</SystemName>
  </Generator>

  <!-- Distributor Einstellungen -->
  <Distributors>
    <Distributor>
      <SystemName>%myApplicationServer%</SystemName>
      <QuantumDuration>PT10S</QuantumDuration>
    </Distributor>
  </Distributors>

  <!-- Ausführungseinstellungen -->
  <ApplicationExecutionSettings>
    <QuantumDuration>PT15S</QuantumDuration>
    <Vacuum>
      <RetentionAge>P14D</RetentionAge>
      <VacuumSchedule>
        <Schedule>
          <StartTime>01:00</StartTime>
        </Schedule>
      </VacuumSchedule>
    </Vacuum>
  </ApplicationExecutionSettings>
```

Listing 45.14 Generator-, Verteiler- und Ausführungseinstellungen

So – es muss zum letzten Mal aktualisiert werden. Mit dem Eintrag *45.14.xml* für die *ADF*-Datei.

Den Windows-Dienst erzeugen

Hat der ganze Ablauf bis jetzt ohne Fehlermeldungen funktioniert, sind nun die Applikations-Tabellen und sonstigen Datenbankobjekte angelegt und mit den notwendigen Anwendungsdaten gefüllt. Bevor jetzt die neue SQLNS-Engine durch das Anlegen einer Dienstinstanz aus der Taufe gehoben wird, müssen noch kleinere Vorbereitungen getroffen werden. Zunächst einmal ist ein Login vorzubereiten, welches der Dienst später benutzen soll um auf die SQLNS-Objekte zuzugreifen. Im Echtbetrieb ist es

wahrscheinlich, dass Sie ein Dienstkonto für den Dienst einrichten, da im Normalfall auf verschiedene Ressourcen des Servers oder des Netzwerks zugegriffen werden muss.

Zum Testen reicht auch ein SQL Server-Login. Das folgende T-SQL-Skript bereitet das vor und legt auch gleich die Benutzer in den Datenbanken an. Über die Rollenzuweisungen werden die notwendigen Rechte zugewiesen.

```
CREATE LOGIN netShop_NS_User
WITH
   Password = 'TotalGeheim!',
   CHECK_EXPIRATION =  OFF,
   CHECK_POLICY = OFF

USE netShop_NS_Instance
CREATE USER netShop_NS_User FOR LOGIN netShop_NS_User
EXEC sp_addrolemember 'NSRunService', 'netShop_NS_User'
GO

USE netShop_NS_InfoMailings_Application
CREATE USER netShop_NS_User FOR LOGIN netShop_NS_User
EXEC sp_addrolemember 'NSRunService', 'netShop_NS_User'
GO
```

Listing 45.15 Anlegen eines Logins und der Datenbankbenutzer

Als nächstes geht es um einen Vorgang, der als Registrieren einer SQLNS-Instanz bezeichnet wird. Dabei wird dem Betriebssystem der neue Dienst bekannt gemacht, der sich um das Ausführen der Instanz und der darin enthaltenen Anwendungen kümmern soll. Die Informationen über den neuen Dienst werden in die Windows-Registrierungsdatenbank geschrieben und es werden Leistungsindikatoren für die Überwachung der Aktivitäten angelegt. Tatsächlich basiert der neue Dienst auf der ausführbaren Datei NS*Service.exe* der als Startparameter der Name der Instanz übergeben wird.

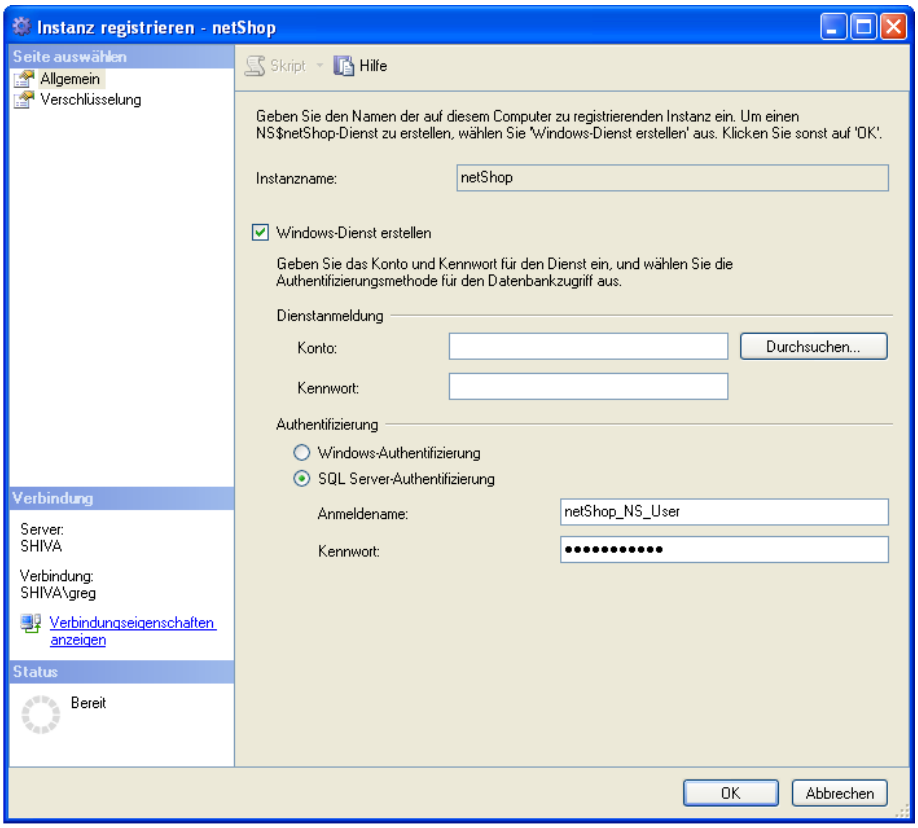

Abbildung 45.5 Registrieren einer SQLNS-Instanz

Das Registrieren wird über das gleichnamige Kontextkommando gestartet. Achten Sie darauf, die Verbindungsinformationen sorgfältig anzugeben, damit der Start des Dienstes reibungslos funktioniert (Abbildung 45.5).

Nach Abschluss der Registrierung finden Sie im SQL Server-Konfigurations-Manager den neuen Dienst unter dem Namen der Instanz mit dem Präfix »NS$« (für das Beispiel also »NS$netShop). Sie können den Dienst hier wie gewohnt verwalten.

Und noch immer läuft das neue Benachrichtigungssystem nicht. Jetzt fehlen aber nur noch zwei kurze Handgriffe. Der erste besteht in der Aktivierung der Anwendung(en). Über den Kontextmenübefehl *Aktivieren* werden die Komponenten aller installierten Anwendungen »scharf geschaltet«. Anders ausgedrückt: Sobald der Dienst gestartet ist, läuft die Maschinerie dann auch tatsächlich los. Und das *Starten* ist nun wirklich der allerletzte Schritt. Uff – die Anwendung läuft und wartet auf Input in Form von Ereignissen!

Werfen Sie jetzt sicherheitshalber noch einen Blick auf die Eigenschaften der Instanz. Wenn die so aussehen, wie in Abbildung 45.6 dargestellt, dann hat die Erzeugung und Initialisierung des Dienstes funktioniert.

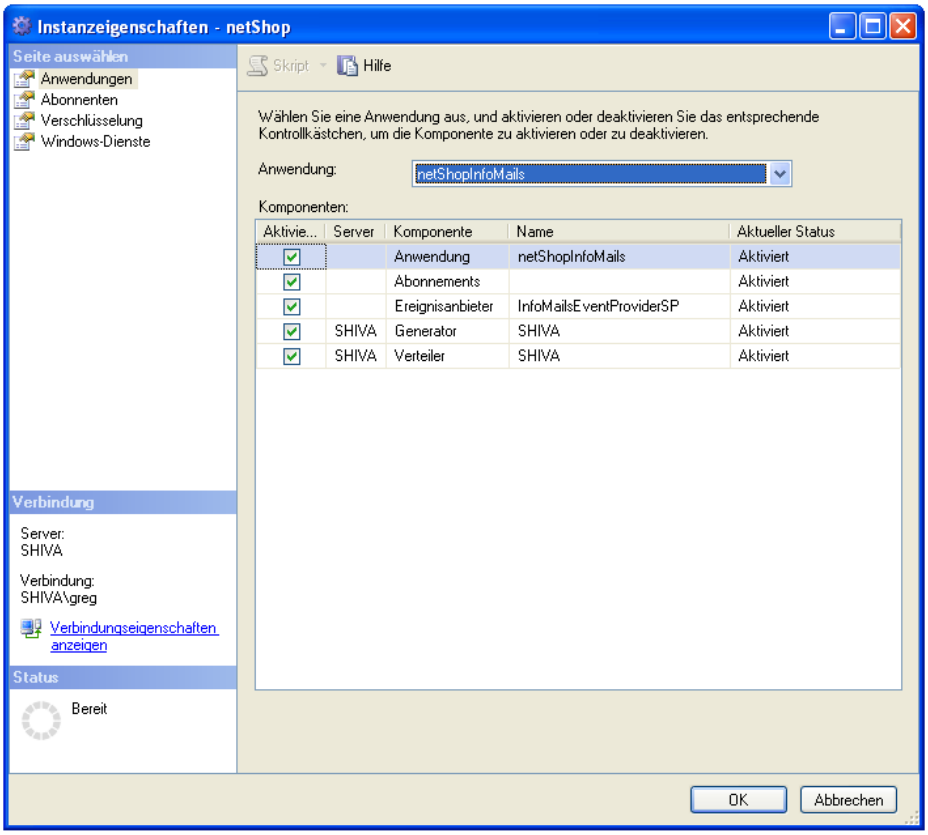

Abbildung 45.6 Eigenschaften einer aktivierten und gestarteten Instanz

Abonnenten hinzufügen

Das Schreiben einer vollständigen Verwaltungslösung wäre zu diesem Zeitpunkt ein kleiner Overkill. Um eine paar Abonnenten hinzuzufügen tun es auch zwei kleine Visual Basic Skripte, die an der Kommandozeile aufgerufen werden. Sie können diese Skripte leicht anpassen, um die Daten für Ihre eigene Anwendung hinzuzufügen. Das Skript nach Listing 45.16 liest sich nicht allzu schwer, denke ich. Es fügt der *netShop*-Instanz drei neue Abonnenten hinzu. Ja genau: Abonnenten werden auf der Instanzebene verwaltet. Das hat den Vorteil, dass die Abonnentendaten für jede Anwendung der Instanz verwendet werden können.

```
Dim nsInstance, nsSubscriber

' Es wird ein NSInstance-Objekt benötigt

Set nsInstance = WScript.CreateObject("Microsoft.SqlServer.NotificationServices.NSInstance")

nsInstance.Initialize "netShop"
```

```
' Und dann noch ein NSSubscriber-Objekt

Set nsSubscriber = WScript.CreateObject("Microsoft.SqlServer.NotificationServices.Subscriber")
nsSubscriber.Initialize (nsInstance)

' Es kann losgehen

nsSubscriber.SubscriberId = "Peter"
nsSubscriber.Add
nsSubscriber.SubscriberId = "Paul"
nsSubscriber.Add
nsSubscriber.SubscriberId = "Marie"
nsSubscriber.Add

wscript.echo "Neue Abonnenten wurden erfolgreich hinzugefügt!"
```

Listing 45.16 Anlegen neuer Abonnenten

Das Registrieren für Abonnements geschieht auf der Anwendungsebene. Auch Listing 45.17 weist keine besonderen Hürden auf. Nach dem Erzeugen des *NSInstance*-Objekts auf oberster Ebene geht es weiter mit dem Instanziieren der passenden Abwendungs- und Abonnements-Objekte (die jeweils mit den korrekten Bezeichnern initialisiert werden müssen). Anschließend wird das *nsSubscription*-Objekt jeweils mit einer der *SubscriberID* aus Listing 45.16 versehen und mit einem Startdatum für den Beginn des Abo-Bezuges als Parameter versorgt. Das Feld *StartDate* ist ja Bestandteil der Abonnements-Klasse *netShopInfoMails*.

```
Dim nsInstance, nsApplication, nsSubscription

' Wie immer: ein NSInstance-Objekt wird benötigt
Set nsInstance = WScript.CreateObject("Microsoft.SqlServer.NotificationServices.NSInstance")
nsInstance.Initialize "netShop"

' Und ein NSApplication-Objekt
Set nsApplication = WScript.CreateObject("Microsoft.SqlServer.NotificationServices.NSApplication")
nsApplication.Initialize (nsInstance), "netShopInfoMails"

' Und ein NSSubscription-Objekt
Set nsSubscription = WScript.CreateObject("Microsoft.SqlServer.NotificationServices.Subscription")
nsSubscription.Initialize (nsApplication), "InfoMailingsSubscription"

' Die Abonnements registrieren
nsSubscription.SubscriberId = "Peter"
nsSubscription.SetFieldValue "StartDate", "01.06.2006"
nsSubscription.Add
nsSubscription.SubscriberId = "Paul"
nsSubscription.SetFieldValue "StartDate", "01.06.2006"
nsSubscription.Add
nsSubscription.SubscriberId = "Marie"
nsSubscription.SetFieldValue "StartDate", "01.06.2006"
nsSubscription.Add

wscript.echo "Abonnements erfolgreich hinzugefügt."
```

Listing 45.17 Registrieren der Abonnements

Führen Sie beide Skripte nacheinander durch Doppelklicks aus. Bekommen Sie ein positives Feedback (»...erfolgreich...«), dann wurden die neuen Abonnenten angelegt und die Abonnements registriert.

Die Konfiguration prüfen

Vor dem ersten Testlauf können Sie das System noch durch das Abfragen der internen Sichten und Tabellen überprüfen. Das nächste Skript gibt einen kleinen Eindruck davon. Sie sollten die eingetragenen Abonnentendaten wiederfinden und auch einen Ereignis-Provider. Ereignisdaten gibt es natürlich (noch) nicht.

```
USE netShop_NS_Instance

-- Abonennenten mit Geräten eingetragen?
SELECT * FROM NSSubscriberDeviceView

USE netShop_NS_Applications

-- Provider eingetragen?
SELECT * FROM InfoMailings.NSProviders

-- Sind Ereignisse vorhanden?
EXEC InfoMailings.NSEventBatchDetails @EventClassName = NewMailings, @EventBatchId = 1
```

Listing 45.18 Überprüfung der Systemdaten

Die Anwendung testen

Um die Anwendung zu Testen müssen Ihr ein paar Ereignisdaten zugeführt werden. Freundlicher Weise hat der SQL Server beim Konfigurieren der Anwendung ein paar gespeicherte Prozeduren erzeugt, mit denen sich leicht Testdaten erzeugen lassen. NS*EventBeginBatchInfoMailingsEvent* startet einen so genannten Event Batch (also eine Anzahl von Ereigniszeilen). Die Zeilen selbst werden mit NS*EventWriteInfoMailingsEvent* eingefügt. Die Namen der Prozeduren und die Parameter wurden anhand der Definition der Ereignisklasse *InfoMailingsEvent* gebildet. Der komplette Ereignisstapel wird mit dem abschließenden NS*EventFlushBatchInfoMailingsEvent* an die Anwendung übergeben. Listing 45.19 zeigt, wie es geht.

```
USE netShop_NS_InfoMailings_Application

-- Wir brauchen einen EventBatch

DECLARE @BatchID bigint;
EXEC dbo.NSEventBeginBatchInfoMailingsEvent N'InfoMailsEventProviderSP', @BatchID OUTPUT;

EXEC dbo.NSEventWriteInfoMailingsEvent
   @EventBatchId=@BatchID,
   @MailingDate='01.08.2006',
   @Headline='Sonderaktion gesunde Sommergetränke!',
   @MailingText= 'Liebe Kunden - es gibt gute Nachrichten! Unsere Sommeraktion...'
EXEC dbo.NSEventWriteInfoMailingsEvent
   @EventBatchId=@BatchID,
   @MailingDate='01.09.2006',
   @Headline='Wir erweitern unser Sortiment',
```

```
   @MailingText= 'Liebe Kunden - Neuigkeiten von Ihrem netShop! Zum Ende...'
EXEC dbo.NSEventFlushBatchInfoMailingsEvent @BatchID;
```

Listing 45.19 Die Anwendung wird mit Testdaten »gefüttert«

Lassen Sie den Anwendungen jetzt ein paar Sekunden Zeit. Danach sollte im Zielordner *D:\Mailings* das HTML-Dokument *Demo.htm* erscheinen. Das Zieldokument wird im Browser ungefähr so aussehen, wie es Abbildung 45.7 zeigt.

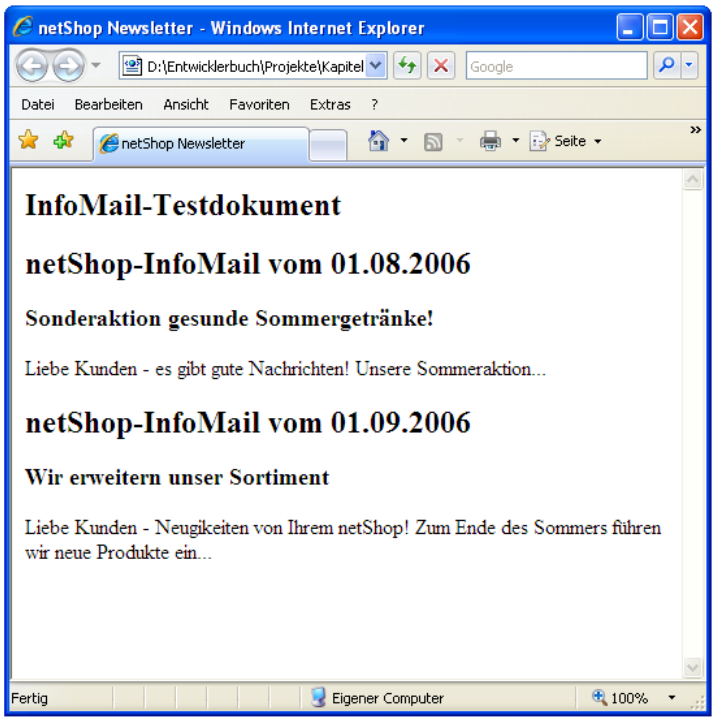

Abbildung 45.7 Das Testdokument mit den generierten Benachrichtigungen

TIPP Dummer Weise bemerken Sie manche Fehler in den Konfigurationsdateien erst, wenn Sie die Anwendung testen. Um den Problemen auf die Schliche zu kommen bietet es sich an, die verschiedenen von den Notification Services verwendeten Tabellen zu untersuchen. Diese sind (leider nur) teilweise in der Online-Dokumentation beschrieben. Unscheinbare Fehler verstecken sich häufig in den verwendeten T-SQL-Befehlen. Kontrollieren Sie vor allen Dingen, ob sich in den internen Tabellen die erwarteten Datensätze befinden und suchen Sie auf diese Art die Stelle, an welcher der Ablauf stockt. Wenn Sie Änderungen in den ICF- und AdW-Dateien vornehmen möchten, dann deaktivieren Sie die SQLNS-Instanz dazu temporär.

Ein richtiges Debugging gestaltet sich etwas schwierig, da es so etwas wie Stopp-Punkte nicht gibt. Die entscheidenden Tabellen und Sichten, aus denen die Regeln ihre Daten beziehen, verlieren zudem nach der Verarbeitung sofort Ihren Wert, sodass ein Test der Aktionen mit Live-Daten nicht ganz einfach ist. Glücklicherweise gibt es ein paar Hilfsprozeduren, mit denen Sie die Verarbeitung in einem angehaltenen System simulieren können: *NSPrepareRuleFiring*, *NSExecuteRuleFiring* und *NSSetQuantumClock* sind die wichtigsten.

Notification Management Objects

Getreu der guten alten Devise »Echte Programmierer füllen keine Formulare« aus, können Sie die notwendigen Konfigurationen der Komponenten einer SQLNS-Lösung auch in einer .NET-Applikation mithilfe der Verwaltungsobjekte der Notification Management Objects durchführen. Um in einem Projekt mit den NMO arbeiten zu können, müssen Sie die folgenden Assemblies referenzieren:

- **Microsoft.SqlServer.Smo.dll:** In dieser DLL sind die Klassen der NMO enthalten. Die NMO bilden eine Teilmenge der SMO (Kapitel 27). Es gibt keine getrennte Assembly.
- **Microsoft.SqlServer.ConnectionInfo.dll:** Diese DLL wird Ihnen aus dem Kapitel 27 bekannt vorkommen. Sie enthält die Hilfsklassen die notwendig sind, um eine Verbindung mit einem SQL Server herzustellen.

Die eigentlichen Objekte der NMO befinden sich im Namensraum *Microsoft.SqlServer.Management.Nmo*. Zusätzlich werden Sie in Verwaltungsanwendungen für die SQLNS mit den allgemeineren Namensräumen *Smo* und *Common* arbeiten.

Wenn Sie programmatisch auf die Notification Services zugreifen möchten um per managed Code Instanzen und alle anderen Objekte einer SQL-NS-Applikation einzurichten, muss zunächst einmal eine administrative Verbindung auf die SQL Server-Instanz geöffnet werden, welche die Instanz beheimaten soll. Das Vorgehen kennen Sie ebenfalls bereits aus Kapitel 27. Es werden die üblichen SMO-Objekte eingesetzt. Zur Konfiguration von Instanzen und Anwendungen instanziieren Sie die entsprechenden Klassen, stellen die Eigenschaften ein und fügen die Objekte an die korrekten Elternobjekte an. Listing 45.20 gibt einen Eindruck. Es wird ein neue Instanz mit einer ersten Anwendung vorbereitet. Da Sie diese Vorgehensweise sicher aus allen möglichen Objektmodellen kennen, möchte ich an dieser Stelle nicht weiter darauf eingehen. Im Projektverzeichnis zu diesem Kapitel finden Sie ein kleines Projekt – *Notification Services NMO. Vbproj* – welches die Konfigurationsschritte, die in den vorherigen Abschnitten vorgestellt wurden, mit den NMO implementiert. Dieses Projekt können Sie als Rumpf für eine eigene Verwaltungsanwendung benutzen. Viel Spaß dabei!

```
Dim myInstance As Instance = _New Instance(objNotificationServices, "netShop")
Dim myfileChannel As DeliveryChannel = New DeliveryChannel(myInstance, "FileChannel")

myfileChannel.ProtocolName = "File"

Dim myfileNameArg As DeliveryChannelArgument = _
   New DeliveryChannelArgument(myfileChannel, "FileName")
myfileNameArg.Value = "D:\Mailings\Demo.htm"
```

```
myfileChannel.DeliveryChannelArguments.Add(myfileNameArg)
myInstance.DeliveryChannels.Add(myfileChannel)
Console.WriteLine("Auslieferungskanal hinzugefügt.")

Dim myApp As Application = ConfigureApplication(myInstance)
myInstance.Applications.Add(myApp)
Console.WriteLine("Applikation hinzugefügt.")
```

Listing 45.20 Einrichten einer SQLNS-Instanz mit den NMO