

Skalierbare nachrichtenbasierte Systeme mit SQL Server 2005 entwickeln

Der Logikmagnet

Im Herbst 2005 wird SQL Server 2005 auf den Markt kommen und mit ihm der so genannte Service Broker. Die neue Technologie ermöglicht es, zwischen SQL-Server-Datenbanken Nachrichten und Daten asynchron auszutauschen und in Warteschlangen zu speichern. Dadurch wird es möglich, mehr Geschäftslogik in den SQL Server zu verlagern.

Das Ganze erinnert stark an Microsoft Message Queuing (MSMQ), und sogleich stellt sich die erste Frage: Wozu noch ein Messaging-System, und dann auch noch in einer Datenbank? Die Antwort gibt eine kurze Betrachtung zum Unterschied von synchronen und asynchronen Verfahren und ihren Konsequenzen.

Synchron oder asynchron

In den meisten Fällen erfolgt der Aufruf einer Funktion synchron, das bedeutet, der Aufrufer muss so lange warten, bis die Funktion ihre Arbeit beendet hat und ein Ergebnis zurückliefert. Aber was, wenn diese Arbeit sehr lange dauert? In dieser Zeit ist der Aufrufer blockiert und kann nichts anders tun. Deshalb werden in Windows-Anwendungen lange dauernde Operationen oft asynchron ausgeführt. Asynchron heißt, der Aufrufer muss nicht auf das Ende der aufgerufenen Funktion warten und kann in der Zwischenzeit weiterarbeiten. Ist die aufgerufene Funktion mit ihrer Arbeit fertig und kann das Ergebnis zurückliefern, so wird der Anwender informiert.

Bei einer Desktop-Anwendung mag sich der scheinbare Stillstand während des synchronen Funktionsaufrufs kaum dramatisch auswirken – vom Warten einmal abgesehen. In einer Server-Anwendung jedoch können die Konsequenzen verheerend sein. Nämlich dann, wenn viele Clients gleichzeitig Anfragen stellen, die der Server beantworten muss. Dafür stehen einer Server-Anwendung meist mehrere Threads zur Verfügung, um mehrere Client-Anfragen parallel bearbeiten zu können. Die Anzahl dieser Threads ist jedoch begrenzt, und gerade in den Zeiten, in denen Lastspitzen auftreten, müssen Clients warten und sind bei synchronen Funktionsaufrufen an den Server für längere Zeit blockiert.

Briefe mit der Post

Das Versenden eines Briefes mit der Post ist ein asynchroner Vorgang. Viele tausend Menschen können gleichzeitig Briefe an einen Empfänger, eine Behörde zum Beispiel, senden. Die eingegangenen Briefe werden dann nach und nach bearbeitet, und an die Absender werden Antwortbriefe verschickt. Hat die Behörde sehr viele Schreiben erhalten, dann kommen diese auf den Posteingangsstapel (Queue) und die Antwort dauert länger. Der große Vorteil für alle Beteiligten ist, das niemand untätig warten muss. Ein sehr fleißiger Briefeschreiber kann viele weitere Briefe an andere Behörden schreiben oder einfach mal ein Buch lesen, während er auf die Antwort auf den ersten Brief wartet. Die Behörde kann die Flut von Briefen kontinuierlich die ganze Woche über verarbeiten und so in Zeiten mit wenig Posteingang die Briefe aus Spitzenzeiten bearbeiten. Es wäre auch denkbar, dass die Behörde kurzfristig zusätzliche Kräfte einstellt, um die Anfragen zügig abzuarbeiten.

Nachrichtenbasierte Systeme

Der große Vorteil ist, dass die Beteiligten voneinander entkoppelt sind und sich nicht gegenseitig blockieren. Ein solches System wird als nachrichtenbasiert bezeichnet. Die Entwicklung ist ein wenig aufwändiger als die eines synchronen Systems, da hier einige Sonderfälle zu berücksichtigen sind. Denn es kann vorkommen, dass keine Antwort eintrifft. Der Client kann nur begrenzt weiterarbeiten, solange er auf bestimmte Informationen warten muss. Antworten können entweder in einer ganz anderen Reihenfolge eintreffen, als die Anfragen losgeschickt wurden, oder ein inkorrektes Format haben.

Die Vorteile sind aber auch beachtlich. Ein solches System blockiert nicht und

kann besser skalieren. Auch Erweiterungen lassen sich in ein solches System einfacher integrieren und neue Systemkomponenten in den Nachrichtenfluss einbinden, egal wo sie arbeiten. Eine ausführlichere Beschreibung der Vorteile von nachrichtenbasierten Systemen und MSMQ ist unter [1], [2] und [3] zu finden.

In der Microsoft-Welt existiert bereits seit vielen Jahren der Dienst MS Message Queuing (MSMQ) als asynchrone Nachrichteninfrastruktur. MSMQ muss auf allen beteiligten Computern laufen und wird bei der Windows-Installation leider nicht als Standard eingerichtet. Er ist von COM- und .NET-Komponenten gut anzu-

Auf einen Blick

Autor



Marcel Gnoth ist Leiter der Entwicklungsabteilung bei der Berliner NTeam GmbH. Er gibt Trainings für Entwickler und hält Vorträge auf der Entwicklerkonferenz Basta! Seine Schwerpunkte liegen auf COM, Visual Basic 6, Informationssystemen und .NET. Sie erreichen ihn unter mgnoth@nteam.de oder über www.gnoth.net.

dotnetpro.code
A0506ServiceBroker



Sprachen TSQL

Technik SQL Server Service Broker

Voraussetzungen SQL Server 2005, alle Versionen

Serie

Teil 1: Einführung in den Service Broker

Teil 2: Der Service Broker im Detail

Zum Beispielcode der Heft-CD

Auf der Heft CD befindet sich eine SQL-Datei, die ein durchgehendes Beispiel enthält. Der Code sollte allerdings immer Schritt für Schritt ausgeführt werden. Das Beispiel basiert auf der Betaversion der SQL Server Standard Edition vom Februar 2005. Es sollte aber auch genauso auf einer Express Edition funktionieren.

Allerdings fehlt bei der Expressversion das SQL Server Management Studio und der darin enthaltene Nachfolger des Query Analyzers, über den die SQL-Anweisungen an den Server geschickt werden. Das ist aber auch über ein Datenbankprojekt von Whidbey oder den alten 2000er Query Analyzer möglich.

Der Service Broker wird normalerweise mitinstalliert und ist in neuen Datenbanken aktiv. Mit den folgenden TSQL-Anweisungen können Sie den Status des Service Brokers für eine Datenbank prüfen und setzen:

```
SELECT name, database_id, is_broker_enabled from sys.databases
ALTER DATABASE AdventureWorks SET ENABLE_BROKER
```

sprechen und es ist sogar möglich, eine MSMQ-Transaktion und eine SQL-Server-Transaktion mit dem so genannten Distributed Transaction Coordinator (DTC) zu koordinieren. Aber da zeigen sich auch schon die Nachteile. Wie schon angedeutet, ist MSMQ nicht auf jedem System installiert.

In einigen großen Firmen stellt ein solches Setup immer ein großes Problem dar. Zum anderen sind Transaktionen, bei denen DTC im Spiel ist, immer langsa-

mer, als wenn nur ein Ressourcen-Manager beteiligt ist.

Einsatzszenarien des Service Brokers

Der neue Service Broker des Datenbanksystems ist Bestandteil aller Versionen des SQL Server 2005 einschließlich der Expressversion. Einzige Bedingung: Werden Nachrichten zwischen zwei SQL-Server-Instanzen übertragen, muss eine davon ein

ne Vollversion sein. Ein Szenario könnte sein, auf den Clients eine Expressversion des SQL Servers zu installieren und auf einem Server eine Standardversion. Dann könnten die Clients mit Offline-Funktionalität ausgestattet werden. Außendienstmitarbeiter können neue Bestellungen und Kontaktdaten aufnehmen, die als Nachrichten an den Firmenserver zu übermitteln sind. Sind sie mit dem Firmennetz verbunden, lassen sich die Nachrichten direkt zustellen. Besteht keine Verbindung, so werden die Nachrichten in den lokalen Expressversionen zwischengespeichert und später an das Firmennetz übermittelt.

Clients für den Service Broker

Auf den Clients muss aber kein lokaler SQL Server laufen. Jede Client-Software, die in der Lage ist, TSQL-Befehle an einen SQL Server 2005 zu senden, kann mitspielen. Eine ASP.NET-Anwendung könnte so Bestellungen über eine gespeicherte Prozedur in eine Orders-Datenbank eintragen. Die gespeicherte Prozedur würde gleichzeitig Nachrichten an die Zulieferer- und an die Buchhaltungsdatenbank senden, die aus Performance- oder Sicherheitsgründen auf separaten SQL Servern laufen. So lassen sich Geschäftsprozesse voneinander entkoppeln. Die ASP.NET-Anwendung muss nicht auf die Arbeit der anderen Systeme warten, sie schickt ihnen nur Nachrichten und kann dann weiterarbeiten.

Interne Anwendungen

Der Service Broker wird auch für einige SQL-Server-Funktionen verwendet. Die beiden Mechanismen heißen Event Notification und Query Notification. Zu Event Notification gehören Nachrichten, die als Reaktion auf DDL-Anweisungen oder Trace-Ereignisse an einen Service Broker gesendet werden.

Darüber hinaus ist der neue SQL Server in der Lage, SQL-Ergebnismengen, die er an Clients geschickt hat, auf Änderungen hin zu überwachen. Dafür verwendet er den gleichen Mechanismus wie bei indextierten Views. Ändert sich die Ergebnismenge, wird eine Nachricht erzeugt und an eine Service-Broker-Queue geschickt. Diese Fähigkeit heißt Query Notification und wird durch zwei neue Klassen in ADO.NET 2.0 unterstützt. Sie lässt sich aber auch ohne ADO.NET 2.0 nutzen.

Listing 1

TSQL-Code zum Anlegen der Service-Broker-Objekte in der Datenbank.

```
- 1. Create the database
Create Database HalloBerlin
Use HalloBerlin
- 2. Create MessageTypes, Request need to be XML, Response not
CREATE MESSAGE TYPE [mtBerlinRequest] VALIDATION = well_formed_xml
CREATE MESSAGE TYPE [mtBerlinResponse] VALIDATION = NONE
- 3. Create a Contract that defines which MessageTypes are allowed in the
- Conversation and send by Initiator and Target
CREATE CONTRACT [BerlinContract] (
    [mtBerlinRequest] SENT BY INITIATOR,
    [mtBerlinResponse] SENT BY TARGET
)
- 4. Create the Queues for the Conversation
CREATE QUEUE [quBerlinInitiator]
CREATE QUEUE [quBerlinTarget]
- 5. A Service is the Name of the Conversation Endpoints Messages
- are exchanged between Endpoints
CREATE SERVICE [svcBerlinRequest] ON QUEUE [quBerlinTarget] (
    [BerlinContract]
)
CREATE SERVICE [svcBerlinResponse] ON QUEUE [quBerlinInitiator] (
    [BerlinContract]
)
```

Asynchrone Trigger

Trigger werden innerhalb der Transaktion ausgeführt, die den Trigger ausgelöst hat. Bei vielen Triggern ist das kein Problem. Unschön ist dies jedoch, wenn so ein Trigger eine Aufgabe ausführt, die länger dauert und die auch später hätte ausgeführt werden können. In diesem Fall könnte der Trigger einfach eine kurze Nachricht mit der Aufgabenstellung versenden und die auslösende Transaktion könnte schnell beendet sein. Das Versenden der Nachricht erfolgt ebenfalls transaktional und ist daher zuverlässig.

Beim Eintreffen der Nachricht in der Queue kann automatisch eine gespeicherte Prozedur aktiviert werden, die innerhalb einer eigenen Transaktion die Arbeit des Triggers übernimmt. Es sind auch andere Aktivierungen beim Eintreffen einer Nachricht möglich.

Durch eine solche Entkopplung der Arbeit auf dem Server lässt sich auch die Anzahl der Sperren verringern, da Aktionen nun in unabhängigen Transaktionen nacheinander erfolgen. Auch für die Batch-Verarbeitung oder für die Lastverteilung bietet der Service Broker eine Menge Möglichkeiten.

Integration des Service Brokers in die Datenbank

Die Objekte, mit denen der Service Broker programmiert wird, sind vollständig in das Datenbanksystem eingebunden. Queues, Services, Message-Typen und so weiter haben den gleichen Stellenwert wie andere Objekte in der Datenbank. Im neuen Enterprise Manager, dem Management Studio, sind die wichtigsten Objekte einsehbar. Jede Datenbank verfügt nun auch über einen Knoten *Service Broker*, siehe Abbildung 1. Service-Broker-Objekte werden auch beim Backup und Restore der Datenbank gesichert und wiederhergestellt. Auf diese und weitere Objekte kann auch über SQL-Anweisungen wie SELECT, CREATE, DROP oder ALTER zugegriffen werden.

Das HalloBerlin-Beispiel im Service Broker

Listing 1 enthält den TSQL-Code zum Anlegen der Objekte. Der Kasten „Zum Beispielcode der Heft-CD“ bietet dazu weitere Hinweise. Als Erstes wird eine neue, leere Datenbank mit dem Namen *Hallo-Berlin* erzeugt. Das Beispiel lässt sich auch mit jeder anderen Testdatenbank

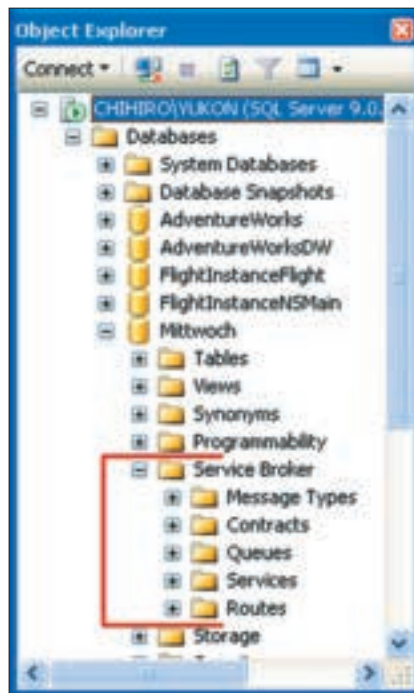


Abbildung 1 Im Object Explorer des SQL Server Management Studios sind die Service-Broker-Objekte zu finden.

ausführen. Im Beispiel werden Nachrichten zwischen einer Request- und einer Response-Queue ausgetauscht. Beide Queues befinden sich in derselben Datenbank auf demselben Server.

Ein einfaches *HalloWelt* zwischen zwei SQL-Server-Instanzen ist komplexer und folgt im zweiten Artikel zu diesem Thema in der nächsten dotnetpro. Abbildung 2 zeigt einen Überblick über die beteiligten Objekte.

Der Nachrichtentyp

Für Anfragen und Antworten sind jeweils in dem TSQL-Statement die entsprechen-

den Nachrichtentypen zu definieren. Dabei lässt sich gleichzeitig auch eine Validierung festlegen. In Listing 1 muss die Anfrage aus gültigem XML bestehen, die Antwort kann beliebig sein. Für die Validierung ist es auch möglich, ein XML-Schema zu verwenden. Schlägt die Validierung fehl, so verweigert die empfangende Queue die Annahme, worauf die sendende Queue eine Fehlernachricht erzeugt. Die Bezeichnung des *MessageTypes* ist dabei frei wählbar. Sie dient den Empfängern auch zur inhaltlichen Unterscheidung der eintreffenden Nachrichten. Deshalb kann es sinnvoll sein, für strukturell gleiche Nachrichten unterschiedliche Typen zu definieren.

Contract

Ein so genannter *Contract* legt fest, mit welchen Nachrichtentypen die Anfrage und die Antwort zu erfolgen hat. Dabei lassen sich auch mehrere Nachrichtentypen für die Anfrage und die Antwort festlegen. Ein Contract wird auch für Services benötigt.

Queue

Eine *Queue* enthält empfangene Nachrichten sowie Meldungen über Kommunikationsfehler. Deshalb müssen entnommene Nachrichten auf ihren Typ geprüft werden. Beim Erzeugen einer Queue lassen sich noch eine Reihe weiterer Parameter angeben, doch dazu mehr im zweiten Teil der Serie.

Service

Ein *Service* ist die Verbindung zwischen einer Queue und einem Contract. Außerdem werden Nachrichten nicht direkt an

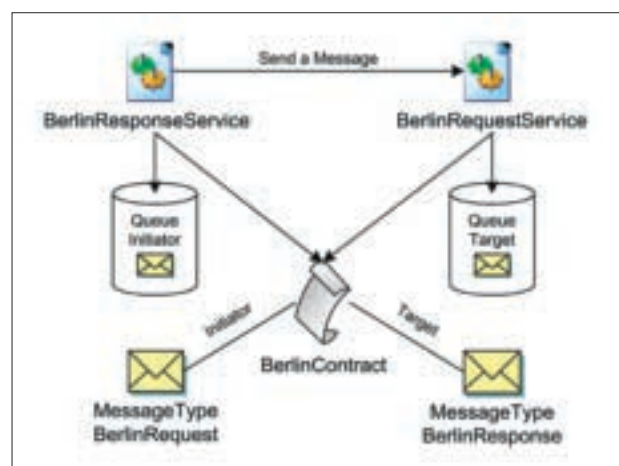


Abbildung 2 Die Service-Broker-Infrastruktur für das Senden und Empfangen von Nachrichten zwischen zwei Queues in einer Datenbank.

eine Queue gesendet, sondern immer an einen Service, hinter dem sich dann eine Queue verbirgt. Ein Service akzeptiert nur Nachrichten, die dem Contract entsprechen. Wird beim Erzeugen des Services kein Contract angegeben, dann kann dieser Service nur senden, aber nicht als Empfänger angegeben werden. Auch die Definition von *Routes* zwischen SQL-Servern und *EndPoints* erfolgt mithilfe von Services, doch dazu später mehr. Ein Service ist mit einem abstrakten Dienstleister vergleichbar, dessen Ressourcen in Anspruch genommen werden.

Dialog und Conversation

Der *Dialog* ist eine Spezialform der *Conversation* zwischen zwei Services. Die physische Kommunikation findet zwischen zwei Kommunikations-*Endpoints* statt, aber auch dazu mehr im nächsten Teil. Die Nachrichten innerhalb eines Dialoges werden in der gleichen Reihenfolge empfangen, in der sie gesendet wurden, auch über Transaktionsgrenzen hinweg. Jede Nachricht verfügt über ein *ConversationHandle*, über das Nachrichten einer *Conversation* zugeordnet werden können. Ein Dialog basiert immer auf einem Contract, der die zu verwendenden Nachrichtentypen definiert.

In künftigen Versionen ist auch eine andere Form der *Conversation*, der so genannte *Monolog* geplant. Dabei handelt es sich um eine Publisher/Subscriber-Kommunikation.

Conversation Groups

Wenn Geschäftsprozesse mehrere zusammenhängende Dialoge wie Request/Response erfordern, so lassen sich Dialoge in *Conversation Groups* zusammenfassen. Das ist bei der Verarbei-

Listing 2

Einen Request senden.

```
Begin Transaction
SET NOCOUNT ON
DECLARE @conversationHandle uniqueidentifier
-- Begin a dialog to the HalloBerlin Service
BEGIN DIALOG @conversationHandle
FROM SERVICE [svcBerlinResponse]
TO SERVICE 'svcBerlinRequest'
ON CONTRACT [BerlinContract]
WITH ENCRYPTION = OFF, LIFETIME = 600;

-- Send message
SEND ON CONVERSATION @conversationHandle
MESSAGE TYPE [mtBerlinRequest]
(CAST(N'<Request>I want a piece of the
Berlin wall</Request>' AS XML))
Commit
```

tung der Nachrichten wichtig. Serviceprogramme, von denen mehrere parallel laufen können, verarbeiten die eintreffenden Nachrichten einer Queue. Gehören mehrere Nachrichten zu einer Conversation Group, so wird sichergestellt, dass immer dieselbe Instanz des Serviceprogramms diese Nachrichten bearbeitet.

Eine Anfrage senden

Damit steht die Infrastruktur für ein einfaches Request/Response-Szenario. Listing 2 zeigt den Beispielcode für das Senden eines Requests innerhalb einer Transaktion. Um Nachrichten später einander zuordnen zu können, wird als Erstes eine Art Aktenzeichen, das *conversationHandle* definiert.

Anschließend wird der Dialog eröffnet. Dabei werden Quelle und Ziel sowie der Contract, der diesen Dialog definiert, angegeben. Darüber hinaus noch Verschlüsselung und Timeout.

Dabei ist *FROM SERVICE* ein lokales Objekt, dessen Existenz geprüft wird. Ob und wo *TO SERVICE* existiert, ist beim Versenden noch unbekannt, deshalb wird der Name auch als String übergeben. Das Versenden wird erst einmal erfolgreich abgeschlossen.

Wird *TO SERVICE* nicht gefunden, dann erzeugt der Service Broker eine Fehlermeldung, die mit

```
select * from sys.transmission_queue
```

abzufragen ist. Zum Testen kann einfach der Name von *TO SERVICE* verändert werden, dann wird die Übermittlung fehlschlagen, da der Name nicht lokal existiert und keine Route definiert wurde.

Andere Fehlerursachen

Für den Nachrichtentyp *mtBerlinRequest* wird über *well_formed_xml* gültiges XML vereinbart. Diese Prüfung findet nicht beim Versenden, sondern vor dem Empfangen statt. Dabei kann der Empfänger die Nachricht zurückweisen, wenn ihm der Inhalt der Nachricht nicht gefällt. Ändern Sie das gültige XML in *SEND ON CONVERSATION* in ungültiges, indem Sie *</Request>* durch *<Req>* ersetzen.

Die Nachricht wird zurückgewiesen und landet in der *quBerlinInitiator*-Queue, die zum Initiatorservice gehört. Fehlermeldungen haben als Schema schemas.microsoft.com/SQL/ServiceBroker/Error hinterlegt.

Eine weitere Fehlerursache kann der Timeout sein, der bei *BEGIN DIALOG* festgelegt wurde. Wenn der Empfänger die Nachricht nicht innerhalb des Timeouts aus seiner Queue nimmt und beantwortet, dann wird in der Queue des Empfängers eine Fehlermeldung abgelegt. Diese Fehlermeldungen, aber auch gültige, erfolgreich übermittelte Nachrichten

Listing 3

Nachricht empfangen und beantworten.

```
DECLARE @conversationHandle uniqueidentifier
DECLARE @message_body nvarchar(MAX)
DECLARE @message_type_name sysname;
Begin Transaction;

RECEIVE top(1) - Takes only one message from the queue
@message_type_name = message_type_name,
@conversationHandle = conversation_handle,
@message_body = message_body
FROM [quBerlinTarget]

-- check the type of the Message, there can also be other types
if @message_type_name = 'mtBerlinRequest'
Begin
SEND ON CONVERSATION @conversationHandle
MESSAGE TYPE [mtBerlinResponse]
N('Nix mehr da! Nachricht von Server: '+@@servername )
-- With this answer the conversation is over
-- and we can end the dialog now.
END CONVERSATION @conversationHandle
End
Commit
```

Listing 4

Die gespeicherte Prozedur für die interne Aktivierung.

```
CREATE PROCEDURE [dbo].[LogMessage]
WITH EXECUTE AS CALLER
AS
DECLARE @dh uniqueidentifier
DECLARE @msg varbinary(max)
    - retrieve the message from the queue
    WAITFOR(RECEIVE TOP(1) @dh = conversation_handle, @msg = message_body FROM [quBerlinTarget]
    ), TIMEOUT 15000

INSERT INTO dbo.MessageLogTable
(Message) VALUES (cast(@msg as nvarchar(MAX)))
```

können mit folgenden Anweisungen eingesehen werden:

```
select cast(message_body as nvarchar(MAX)),
    * from [quBerlinTarget]
select cast(message_body as nvarchar(MAX)),
    * from [quBerlinInitiator]
```

Der Body einer Nachricht wird binär übertragen und muss in Text umgewandelt werden. Je nachdem, ob Unicode verwendet wird oder nicht, muss nach *varchar* oder nach *nvarchar* gecastet werden.

Die Nachricht empfangen und beantworten

Listing 3 zeigt, wie sich eine Nachricht manuell empfangen und beantworten lässt. Beim Eintreffen einer Nachricht kann auch automatisch eine gespeicherte Prozedur gestartet werden, welche die Nachricht verarbeitet. Es können sogar mehrere gespeicherte Prozeduren parallel eintreffende Nachrichten verarbeiten. Eine derartige gespeicherte Prozedur wird auch als Serviceprogramm bezeichnet.

Mit der RECEIVE-Anweisung wird eine Nachricht ausgelesen – und damit auch aus der Queue entfernt. Alternativ dazu kann auch SELECT verwendet werden, das die Nachrichten in der Warteschlange lässt. Dann wird der Typ der Nachricht geprüft. In einer Queue können Nachrichten verschiedener *Services* landen, dazu gesellen sich noch Fehlermeldungen.

Ist die Nachricht gefunden, kann ihr Inhalt ausgewertet und eine Antwort an den Absender zurückgeschickt werden. Da in diesem einfachen Request/Response-Dialog keine weiteren Nachrichten in diesem *Gespräch* zu erwarten sind, kann die *Conversation* geschlossen und das *Handle* freigegeben werden.

Interessant ist auch, dass der Initiator nicht nur seine Antwort erhält, sondern auch eine Nachricht vom Typ *http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog*.

Queue-Aktivierung

Wenn eine Nachricht in einer Queue eintrifft, dann kann über die Aktivierung des Service Brokers eine Verarbeitung der Nachricht angestoßen werden. Dabei ist zwischen internen Prozessen, also den gespeicherten Prozeduren, und externen Prozessen außerhalb des SQL Servers zu unterscheiden.

Am einfachsten ist dies über eine gespeicherte Prozedur, siehe Listing 4. Zum Ausführen lässt sich ein eigenes Anwenderkonto angeben. So braucht der Sender nur Senderechte für die Queue, die Arbeit der Prozedur kann unter einem eigenem Sicherheitskontext erfolgen.

Als Nächstes muss die Queue-Aktivierung über *STATUS* eingeschaltet werden (Listing 5). Auch hierbei kann ein eigenes Anwenderkonto für den Start der Prozedur angegeben werden.

Wenn viele Nachrichten in kurzer Zeit eintreffen, kann der Service Broker mehrere Instanzen der Prozedur starten. Deshalb sind die *Conversation Groups* so wichtig, da sie gewährleisten, das zusammengehö-

rige Nachrichten immer von der gleichen Instanz der Prozedur bearbeitet werden. *Max_Queue_Readers* definiert die Zahl der parallel laufenden Instanzen.

Clients

Der Service Broker ist nicht nur von gespeicherten Prozeduren aus zu verwenden. Jeder Client, der in der Lage ist, TSQL-Anweisungen zum Server zu senden, kann Service Broker einsetzen, also auch VBA oder VB6.

Besonders interessant wird der Einsatz des Service Brokers zusammen mit ADO 2.0 in Verbindung mit der Queue-Aktivierung. Mit den Klassen *SqlQLDependency* und *SqlNotificationRequest* lässt sich der .NET-Client aktiv über das Eintreffen neuer Nachrichten informieren.

Fazit und Ausblick

Der Vorteil des Service Brokers ist gleichzeitig sein Nachteil: Die Kommunikation ist nur zwischen SQL Servern möglich – dann aber schnell und leistungsfähig mit wenig Overhead. Der erfahrene Datenbankentwickler kann mit vertrauten Befehlen wie SELECT und CREATE arbeiten. Dies ist eine interessante Alternative zu MSMQ, insbesondere für die Entwickler, die schon immer gern die Geschäftslogik in den SQL Server verlagert haben. Zusammen mit der Integration der Common Language Runtime in Yukon zeigt sich eine Alternative zu einer COM+-Geschäftslogikschicht. Der SQL Server wird so zum Logikmagneten.

Sie kennen nun die grundlegenden Fähigkeiten und Objekte des Service Brokers. Im nächsten Teil geht es dann um die Details. Für die Kommunikation zwischen zwei Servern spielen die Sicherheitseinstellungen ebenso eine große Rolle wie EndPoints, Service Binding und Routing. Außerdem werden Sie ein Whidbey-Projekt kennenlernen, das mit dem Service Broker arbeitet. |||||

Listing 5

Queue-Aktivierung.

```
ALTER QUEUE [quBerlinTarget]
WITH ACTIVATION (
    STATUS = ON,
    PROCEDURE_NAME = [LogMessage],
    EXECUTE AS SELF,
    MAX_QUEUE_READERS = 2)
```

- [1] Marcel Gnoth, Mit MSMQ ein nachrichtenbasiertes Informationssystem entwickeln, *basicpro 5/2001*, Seite 26 ff.
- [2] Marcel Gnoth, Auf die Post ist Verlass, *Microsoft Message Queue Service: Mitteilungen veröffentlichen und lesen*, *dotnetpro 3/2004*, Seite 114 ff.
- [3] Marcel Gnoth, Einschreiben mit Rückschein, *MSMQ: Infrastruktur für den Nachrichtentransport*, *dotnetpro 4/2004*, Seite 124 ff.