

Das Nachrichtensystem

Skalierbare, nachrichtenbasierte Anwendungen mit dem Service Broker entwickeln

von Marcel Gnoth **BASTA!-SPEAKER**



Wenn Sie das neue SQL Server Management Studio starten und im Object-Explorer die Unterknoten einer Datenbank aufklappen, finden Sie den Eintrag „Service Broker“. Ein erster Blick in die Hilfe erläutert: Der Service Broker erlaubt es, zwischen SQL-Server-Datenbanken asynchron Nachrichten zu verschicken. „Warum das denn?“ war meine erste Frage ...

Nehmen wir z.B. eine stark belastete Webapplikation, die ihren eigenen SQL Server hat und Bestellungen entgegennimmt. Jeder Auftrag soll eine Reihe weiterer Aktivitäten in anderen Datenbanken wie *Lager*, *Kunde*, *Rechnung* oder anderen Systemen auslösen. Diese Aktionen müssen aber nicht unmittelbar bzw. in der gleichen Transaktion wie das *INSERT* der neuen Bestellung ausgeführt werden, sondern können unabhängig davon später erfolgen.

Wenn die Datenbank für das Frontend stark belastet ist, wäre es doch schön, wenn sie nach dem Erfassen der Bestellung (*INSERT*) eine kurze Nachricht mit den Bestelldaten an einen anderen SQL Server sendete, der dann seinerseits die Verarbeitung übernimmt und Daten in die anderen Systeme oder Datenbanken einträgt. Dadurch würde der Frontendserver entlastet.

Ein weiterer Vorteil wäre, dass, falls das Backendsystem nicht zur Verfügung stünde, das Frontendsystem trotzdem weiter Nachrichten in seinen Postausgang legen könnte, die bei Gelegenheit an das Backendsystem zugestellt würden.

Ein weiteres Szenario wäre eine Batch-Verarbeitung. Gerade hier treten zum Teil hohe Lasten auf, wenn sehr viele Daten auf einmal eintreffen. Dann ist keine Zeit zur Verarbeitung, das System ist mit dem Eingang der Nachrichten mehr als ausgelastet. Die Daten in der Warteschlange werden sukzessive je nach Auslastung abgearbeitet. Über asynchrone Nachrichten können so Prozesse voneinander entkoppelt werden. Das alles bietet der SQL Server Service Broker.

Integrierte Datenbankobjekte

Die Objekte, die der Service Broker verwendet, sind keine Fremdkörper, sondern integraler Bestandteil einer Datenbank des Servers. Dafür gibt es eine Reihe versteckter Systemtabellen in jeder Datenbank, in denen Objekte, aber auch Nachrichten, gespeichert sind. Dadurch profitieren die Service-Broker-Objekte von allen Datenbankfunktionen wie Mirroring, Clustering, Backup und Restore. Der Zugriff auf die verschiedenen Objekte erfolgt meist über System-Views, wie zum Beispiel *Select * from sys.routes* oder über neue T-SQL-Anweisungen wie *Receive* oder *Send*.

In einer komplexen IT-Welt, in der verschiedene Systeme zusammenarbeiten müssen, ist eine lose und einfache Kopplung der Systeme mit klarer Aufgabenstellung gewünscht. Deshalb ist der Begriff SOA neuerdings so beliebt.

In jeder Datenbank können Queues angelegt werden, die Nachrichten speichern. Nachrichten werden jedoch nicht direkt zwischen Queues ausgetauscht. Zwischen Services (SOA) und einem Service ist immer eine Queue zugeordnet,

welche die Nachrichten auch physisch enthält. Durch diesen Abstraktionsschritt verringern sich die Abhängigkeiten der Kommunikationspartner untereinander. Es muss nur der richtige Servicename des Partners und bei entfernten Partnern auch eine Route zu ihm bekannt sein.

Der Nachrichtenaustausch zwischen zwei Services erfolgt transaktional und zuverlässig. Eine Nachricht wird erst aus der Postausgangs-Queue des Senders entfernt, wenn diese Nachricht vom Empfänger akzeptiert und bestätigt wurde. Kann eine Nachricht nicht zugestellt werden, dann wird ein Fehler ausgelöst.

Nachrichtenaustausch

Die Kommunikation zwischen zwei Services wird als *Conversation* bezeichnet. Man unterscheidet zwischen den beiden Unterarten *Monologe* und *Dialoge*. Ein Dialog besteht aus genau zwei Partnern, die sich Nachrichten senden. Der Partner, der den Dialog eröffnet, wird als *Initiator* bezeichnet, der andere als *Target*. Beide können sich so oft Nachrichten senden, bis der Dialog von einem der beiden beendet wird. Der andere erhält dann eine Nachricht, dass der Partner den Dialog beendet hat und muss seinerseits den Dialog ebenfalls beenden.

Kommuniziert ein Service Broker für einen Geschäftsvorgang mit mehreren Partnern, dann muss er mit jedem einen eigenen Dialog beginnen. Mehrere Dialoge können in einer *Conversation Group* zusammengefasst werden. Innerhalb einer Conversation werden alle Nachrichten in der Reihenfolge des Absendens zugestellt. Der Monolog wurde im SQL Server 2005

kurz & bündig

Inhalt

Entwicklung skalierbarer, nachrichtenbasierter Anwendungen mit dem Service Broker des SQL Server 2005

Zusammenfassung

Der Service Broker ist eine interessante neue Möglichkeit, SQL-Server-Applikationen zu entwickeln. Er ist sehr performant und zuverlässig, kann aber nur zwischen SQL Servern kommunizieren



Quellcode auf CD

noch nicht implementiert. Er ist dafür gedacht, dass ein Service viele andere Services informieren kann, eine Art Broadcast-Mechanismus. Es ist geplant, Monologe in der nächsten SQL-Server-Version zur Verfügung zu stellen.

Nachrichtentypen

Für eine Conversation werden Nachrichtentypen definiert, um die eintreffenden Nachrichten besser unterscheiden und verarbeiten zu können. Dabei wird ein Name als Bezeichner und die Art der Validierung festgelegt. Eintreffende Nachrichten können überprüft werden, ob sie aus wohlgeformtem XML bestehen oder sogar, ob ihr Inhalt einem vorher definierten XML Schema entspricht. Schlägt die Validierung fehl, wird die Nachricht abgewiesen:

Listing 1

Öffnen einer Conversation und Senden einer Nachricht

```
Begin Transaction
DECLARE @conversationHandle uniqueidentifier

BEGIN DIALOG @conversationHandle
FROM SERVICE [svcCoffeeResponse]
TO SERVICE 'svcCoffeeRequest'
ON CONTRACT [CoffeeContract]
WITH ENCRYPTION = OFF, LIFETIME = 600;

-- Send message
SEND ON CONVERSATION @conversationHandle
MESSAGE TYPE [mtCoffeeRequest]
(CAST(N'<Request>Coffee 500 t</Request>' AS XML))
Commit
```

Listing 2

Empfangen und Beantworten einer Nachricht

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

Begin Transaction;
RECEIVE top(1)
@message_type_name = message_type_name,
@conversationHandle = conversation_handle,
@message_body = message_body
FROM [quBogota]

Print @message_body

if @message_type_name = 'mtCoffeeRequest'
Begin
SEND ON CONVERSATION @conversationHandle
MESSAGE TYPE [mtCoffeeResponse] N('Nix mehr da!')

END CONVERSATION @conversationHandle -
End
Commit
```

```
Create Database dbColombia
go
Use dbColombia
-- Create MessageTypes
CREATE MESSAGE TYPE [mtCoffeeRequest]
VALIDATION = well_formed_xml
CREATE MESSAGE TYPE [mtCoffeeResponse]
VALIDATION = NONE
CREATE MESSAGE TYPE [mtInform] VALIDATION = NONE
```

Darüber hinaus gibt es noch Systemnachrichtentypen, die innerhalb der Conversation versendet werden können, z.B.: `http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog` und `http://schemas.microsoft.com/SQL/ServiceBroker/Error`. Der Name eines Nachrichtentyps ist nicht zwingend eine URL, sollte aber im Gesamtsystem eindeutig sein, da an einer Brokerkommunikation auch mehrere Server beteiligt sein können. `EndDialog`-Nachrichten werden versendet, wenn einer der Partner die Conversation beendet und Error-Nachrichten, wenn ein Fehler auftritt.

Contract

Über einen *Contract* wird definiert, wer welche Nachrichtentypen verwenden darf. Dadurch wird sichergestellt, dass nur bestimmte Arten von Nachrichten in einer Queue ankommen dürfen und das Programmieren der Nachrichtenverarbeitung wird einfacher:

```
CREATE CONTRACT [CoffeeContract] (
[mtCoffeeRequest] SENT BY INITIATOR,
[mtCoffeeResponse] SENT BY TARGET,
[mtInform] SENT BY ANY
)
```

Queues

Endlich werden die *Queues* angelegt, welche die empfangenen Nachrichten speichern:

```
CREATE QUEUE [quBogota]
CREATE QUEUE [quCali]
```

Queues werden intern über versteckte Tabellen realisiert, die nicht direkt manipuliert werden können. Mit `Select * From quBogota` kann aber eine View der versteckten Tabelle abgerufen werden.

Services

Die Nachrichten werden nicht direkt zwischen zwei Queues ausgetauscht, sondern immer zwischen zwei Services. Die Details eines Service bleiben dem Benutzer verborgen.

So kann zur Laufzeit die Queue eines Services verändert oder auf einen anderen Rechner verschoben werden. Dafür sind nur Änderungen an der Infrastruktur notwendig, aber nicht an der Anwendung selbst:

```
CREATE SERVICE [svcCoffeeRequest] ON QUEUE [quBogota]
(
[CoffeeContract]
)
CREATE SERVICE [svcCoffeeResponse] ON QUEUE [quCali] (
[CoffeeContract]
)
```

Für einen Service werden Queue und Contract festgelegt und mit dem Contract auch die *MessageTypes*, die an der Kommunikation teilnehmen können (Abb. 1).

Senden einer Nachricht

Nach all diesen Vorbereitungen können jetzt Nachrichten versendet werden. In Listing 1 wird als Erstes eine neue Conversation vom Typ *Dialog* eröffnet. Dabei wird ein Handle für die Conversation vom Service Broker zurückgegeben, das auch für das Senden von Nachrichten verwendet wird.

Der *From Service* ist der Initiator des Dialoges und muss als Service in der Datenbank angelegt worden sein. Der Name des *To Service* wird als String übergeben. Er muss kein lokaler Service sein, deshalb kann der Service Broker nicht gleich feststellen, ob es den Service gibt und er verfügbar ist. Wird der Name nicht lokal gefunden, sucht der Service Broker in der System-View `Select * from sys.routes` nach einer Route zu diesem Service. Kann der SSB den Servicennamen in der Routing-Tabelle nicht finden oder ist der Service nicht erreichbar, wird die Übertragung mit Fehler abgebrochen.

Die Systemwarteschlange `sys.transaction_queue` ist mit dem Postausgang von Outlook vergleichbar. Solange keine Bestätigung vom Empfänger eingetroffen ist, bleibt die zu sendende Nachricht in dieser Queue. Tritt ein Fehler auf, kann er hier gefunden werden.

Beim Versenden kann auch ein *Timeout* angegeben werden. Wird die Nachricht nicht innerhalb der Zeitspanne vom Empfänger verarbeitet, dann erhält der sendende Service eine Fehlernachricht über den Timeout in seine Queue. Das ist insbesondere deshalb wichtig, da Service-Broker-Kommunikationen über eine sehr

lange Zeit laufen können. Zum Beispiel kann der Zielrechner nicht erreichbar sein. Dann liegt es in der Logik der Anwendung, wie lange sie auf eine Antwort warten möchte und was sie macht, wenn ein Time-Out auftritt. Das gehört zur Komplexität asynchroner, nachrichtenbasierter Anwendungen.

Empfangen und Beantworten einer Nachricht

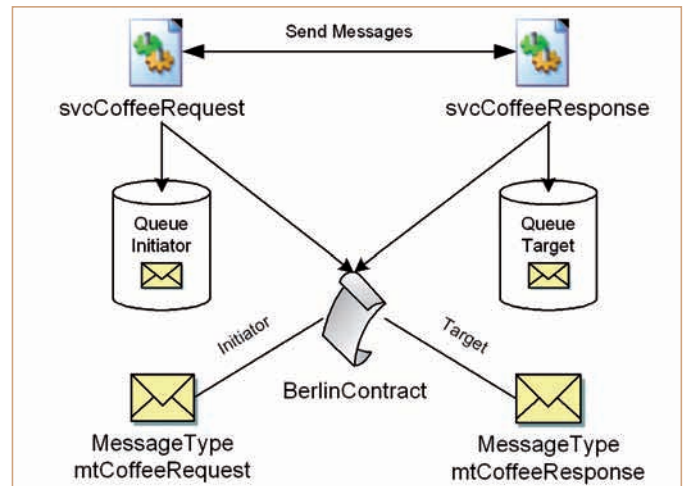
Auf der anderen Seite können nun die Nachrichten über *Select* gelesen oder über *Receive* entnommen werden. Dabei sollte das Conversation-Handle für das Senden der Antwort gespeichert werden. Zu einer Conversation können mehrere Nachrichten gehören. Receive entnimmt immer alle zu einer Conversation gehörenden Nachrichten. Soll nur eine Nachricht entnommen werden, kann das über die *Top*-Klausel geschehen. Befindet sich keine Nachricht in der Queue, dann blockiert diese Anweisung.

Jetzt muss geprüft werden, was für eine Nachricht sich in der Queue befindet. Es ist auch möglich, dass sich mehrere Services eine Queue teilen und jeder dieser Services kann einen anderen Contract vereinbart haben. Auch eine Reihe von Systemnachrichten ist möglich, wie *EndDialog* oder *Error*. Deshalb wird in Listing 2 geprüft, ob es sich um einen *CoffeeRequest* handelt, bevor mit einem *CoffeeResponse* geantwortet wird. Dabei muss das Conversation-Handle der empfangenen Nachricht verwendet werden. Da jetzt keine weitere Kommunikation mehr erforderlich ist, wird die Conversation beendet. Das alles erfolgt transaktional. Vor dem Senden der Antwort können noch eine Reihe von DML-Anweisungen ausgeführt werden, die innerhalb der gleichen Transaktion laufen.

Queue-Aktivierung

Nun ist es nicht so schön, wenn die Queue ständig daraufhin überprüft werden muss, ob neue Nachrichten eingegangen sind, besser wäre es, wenn automatisch eine Verarbeitungsroutine gestartet wird, sobald neue Nachrichten eintreffen – dieser Mechanismus heißt *Queue Activation*. Auf der Heft-CD finden Sie ein Beispiel für eine Stored Procedure, die in der Testdatenbank angelegt wird und beim Eintreffen von Nachrichten diese in eine Log-Tabelle schreibt.

Abb. 1: Ein Service definiert, welche Queue mit welchem Contract verwendet wird



Dieser Queue wird dann die Prozedur zur Aktivierung zugeordnet und sie wird dann immer aufgerufen, wenn neue Nachrichten eintreffen. Für den Fall, dass die Verarbeitung einzelner Nachrichten lange dauert, können auch mehrere Instanzen der Stored Procedure aktiviert werden. Über den *Max_Queue_Reader*-Parameter wird die Anzahl der Instanzen gesteuert, damit nicht einzelne Nachrichten andere blockieren:

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

Kommunikation zwischen zwei Service Brokern

Bis jetzt haben wir nur den einfachen Fall einer lokalen SSB-Kommunikation betrachtet, findet aber die Kommunikation zwischen zwei SQL-Server-Instanzen statt, dann muss die Sicherheit konfiguriert werden. Es werden für die Dialog- und die Transport-Security-Zertifikate mit öffentlichem und privatem Schlüssel verwendet. Es muss eine Route zwischen den Instanzen definiert und über die *SQL Server Surface Area Configuration* muss der Service Broker und Netzwerkzugriff aktiviert werden. Ein Beispiel für eine verteilte Kommunikation finden Sie unter [1].

Wie geht es weiter?

Der Service Broker ist eine interessante neue Möglichkeit, SQL-Server-Applikationen zu entwickeln. Mit ihm ist es möglich, lange dauernde Datenbankprozesse

zu entkoppeln oder eine Art Batchverarbeitung zu implementieren. Sein Vorteil ist gleichzeitig auch sein Nachteil. Er ist sehr performant und zuverlässig, kann aber nur zwischen SQL Servern kommunizieren. Sollen COM+-Komponenten in die Transaktion mit eingebunden werden, dann geht das nur im Rahmen von ADO-Befehlen und der DTC kommt ins Spiel, um die verschiedenen Ressourcen-Manager zu koordinieren.

Der Service Broker kann von jedem Datenbank-Client über ADO oder ADO .NET verwendet werden. Er ist dabei kein Ersatz für MSMQ, die andere Nachrichtentechnologie aus dem Hause Microsoft. MSMQ verfügt über einige Fähigkeiten, die der Service Broker nicht hat, z.B. Broadcasts. Aber wenn ein System SQL-Server-Datenbanken verwendet und dann keine COM+-Komponenten im Spiel sind, ist der Service Broker die bessere Wahl. Insgesamt ist der SSB ein weiterer Baustein, um aus dem SQL Server einen Applikationsserver zu machen. Die CLR-Integration, native Web Services und Service Broker stellen eine neue, stark in die Datenbank integrierte Laufzeitplattform dar, die viele Möglichkeiten bietet.

.....
Marcel Gnoth ist seit 2006 als Senior Consultant bei Avanade Deutschland im Bereich Office Business Application tätig. Sie erreichen ihn unter MarcelG@avanade.com oder über seine Homepage www.gnoth.net.

● Links & Literatur

- [1] Demos, Vorträge und Artikel zum Service Broker: www.gnoth.net/ServiceBroker/
- [2] Roger Walter: The Rational Guide to SQL Server 2005 Service Broker. Rational Press 2005