



SQL Server 2005 Service Broker

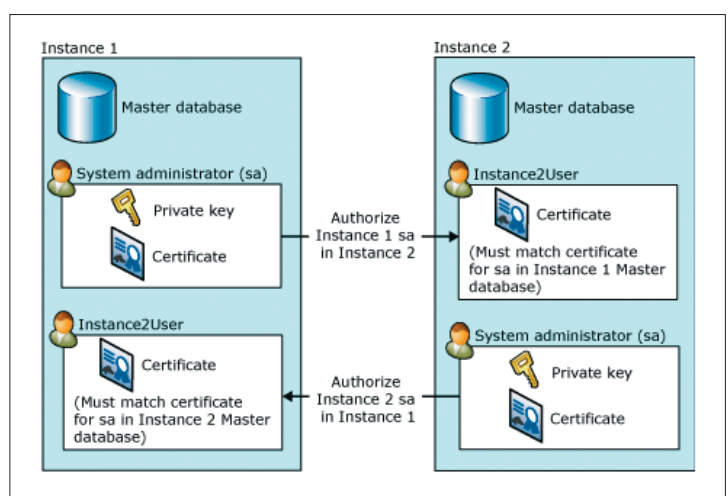
Dialog der Datenbanken

Über den SQL Server Service Broker können mehrere SQL Server 2005 untereinander Nachrichten austauschen. Der erste Teil der Serie hat das Prinzip vorgestellt. Der vorliegende zweite Teil realisiert ein Beispiel. Das ist aber gar nicht so einfach. Denn diesmal hat Microsoft das Thema Sicherheit richtig ernst genommen.

Im ersten Teil der Serie habe ich dargestellt, wie der SQL Server 2005 Nachrichten mit anderen SQL Servern austauschen kann [1]. Die Realisierung eines entsprechenden Beispiels ist aber nicht so einfach. Nach meinen ersten Gehversuchen ging ich voller Enthusiasmus ans Werk und wollte ein verteiltes Szenario mit dem SQL Server Service Broker (SBR) durchspielen, bei dem Nachrichten zwischen zwei Servern versendet werden. Damit bin ich aber kläglich gescheitert.

Im Internet sind zwar etliche Beispiele zu finden, aber alle bezogen sich immer nur auf den einfachsten Fall, den Nachrichtenaustausch innerhalb eines Servers.

Abbildung 1
Zusammenhänge der Transport Security.



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
A0507ServiceBroker



Sprachen

TSQL

Technik SQL Server 2005 Service Broker

Voraussetzungen SQL Server 2005

Serie
Teil 1: Einführung in den Service Broker
Teil 2: Der Service Broker im Detail

Heute weiß ich auch warum. Erst mithilfe der Service Broker Newsgroup [2] und intensivem RTFM der SQL Server Books Online (BOL) gelang es mir, das gewünschte Beispiel zum Laufen zu bringen.

Da hatte doch Microsoft das Thema Sicherheit wirklich einmal so ernst genommen, dass ich erst das Sicherheitskonzept verstehen musste, bevor die Kommunikation funktionierte. Das ist natürlich gut für den laufenden Serverbetrieb. Es wird nicht mehr Tausende unsicherer SQL-Server-Installationen geben, aber dafür wird das Entwicklerleben nicht gerade leichter. Beim Erlernen der vielen neuen Technologien erhöhen Fragen der Sicherheit zusätzlich die Komplexität. Entwickler werden daher in Zukunft mehr Zeit benötigen um Neues zu lernen.

Service Broker Security

Für eine verteilte Service-Broker-Kommunikation sind zwei Sicherheitsmechanismen wichtig, die erst einmal korrekt konfiguriert werden müssen, bevor Nachrichten ausgetauscht werden können. Diese Mechanismen können Windows-Authentifizierung (SSPI) verwenden, de-

ren Einsatz aber nicht überall möglich ist. In vielen Firmen befinden sich SQL Server nicht in der gleichen Domäne oder im gleichen Active Directory und es existieren keine Vertrauensstellungen.

Deshalb bietet der SQL Server eine Alternative an. Dafür werden Zertifikate mit öffentlichen und privaten Schlüsseln verwendet. Diese Zertifikate werden mit ihren öffentlichen Schlüsseln zwischen den Servern ausgetauscht. Der private Schlüssel ist nur lokal in dem Server bekannt, dem das Zertifikat auch gehört. So können die Server bei Netzwerkkommunikation mithilfe der Zertifikate und der Schlüssel die Identität eines entfernten Servers sicherstellen und die Kommunikation vor Missbrauch schützen.

Die Zertifikate entfernter Server werden lokalen Usern zugeordnet, in deren Sicherheitskontext die Kommunikation stattfindet. Informationen über das Erstellen von Zertifikaten finden Sie im Kasten *Zertifikate erstellen*.

Transport Security

Die Transport Security regelt grundsätzlich, welche Server miteinander reden

Listing 1

Transport-Security in der Tokyo-DB anlegen.

```
use master
go
--create master key in the master database
CREATE master KEY encryption BY password = 'TokyoMasterDB'
--create the certificate for transport security
CREATE certificate ctfTokyoMaster
FROM FILE = 'C:\_SBR\SBR-Verteilt\SBR-Verteilt\Tokyo.cer'
WITH private_key
(FILE='C:\_SBR\SBR-Verteilt\SBR-Verteilt\Tokyo.pvk',
 decryption_password='Hallo')
active FOR begin_dialog = ON

--User, that holds the remote certificate with the public key
CREATE LOGIN remcert WITH password = ''
CREATE user remcert FOR LOGIN remcert
--remcert need connect privileges
GRANT connect TO remcert
--create the remote cert and assign a local user
CREATE certificate ctfKyotoMaster
AUTHORIZATION remcert
FROM FILE = 'C:\_SBR\SBR-Verteilt\SBR-Verteilt\Kyoto.cer'
active FOR begin_dialog = ON
select * from sys.certificates
```

dürfen, sie regelt aber nicht die Inhalte der Kommunikation. So kann der Netzzugriff auf einen Server eingeschränkt werden.

Die Authentifizierung erfolgt über SSPI oder Zertifikate, die in der jeweiligen Master-DB installiert werden. Der lokale Server installiert ein Zertifikat mit dem öffentlichen Schlüssel des entfernten Servers. So kann er eintreffende Nachrichten, die mit dem privaten Schlüssel verschlüsselt wurden, entschlüsseln. Damit ist sicher, dass diese Nachrichten von dem Server stammen, dessen Zertifikat lokal gespeichert ist. Abbildung 1 stellt die Zusammenhänge schematisch dar.

Welche Form der Authentifizierung verwendet wird hängt von den Einstellungen der Endpoints – Näheres dazu folgt weiter unten – und der Anwesenheit von Zertifikaten ab. Wo es möglich ist verwendet der SBR die Authentifizierung mit Zertifikaten.

Listing 1 zeigt das Anlegen der Zertifikate auf einem Server. Zuerst wird das eigene Zertifikat mit Private Key aus einer Datei in der Master-DB installiert. Danach wird aus einer zweiten Datei, die vom entfernten Server stammt, ein Zertifikat ohne Private Key in der Master-DB installiert und einem lokalem User Account mit *Connect*-Privileg zugeordnet. Aktivitäten der entfernten DB, wie das Senden von Nachrichten, werden mit diesem User Account durchgeführt.

In der System-View *sys.certificates* sind dann die beiden Zertifikate zu finden. Eines ist mit dem Masterkey der Datenbank verschlüsselt. Das andere enthält nur den öffentlichen Schlüssel des entfernten SQL Servers, dem die Kommunikation mit dem lokalen SQL Server prinzipiell erlaubt wird.

Das gilt für alle Datenbanken und alle Service-Broker-Instanzen des lokalen Servers. Soll die SSPI-Authentifizierung

verwendet werden, so muss in beiden Master-DBs ein Login für das Dienststartkonto des jeweils anderen SQL Servers angelegt werden. Dann müssen keine Zertifikate installiert sein.

Dialog Security

Ist die Netzwerkkommunikation zwischen zwei DB-Servern erst einmal erlaubt, dann ist die Dialog Security für den Ablauf des Nachrichtenaustausches verantwortlich. Sie prüft die Integrität der Nachrichten, verschlüsselt Nachrichten und erlaubt nur autorisierten Services den Nachrichtenversand. Es gibt drei verschiedene Modi der Dialog Security:

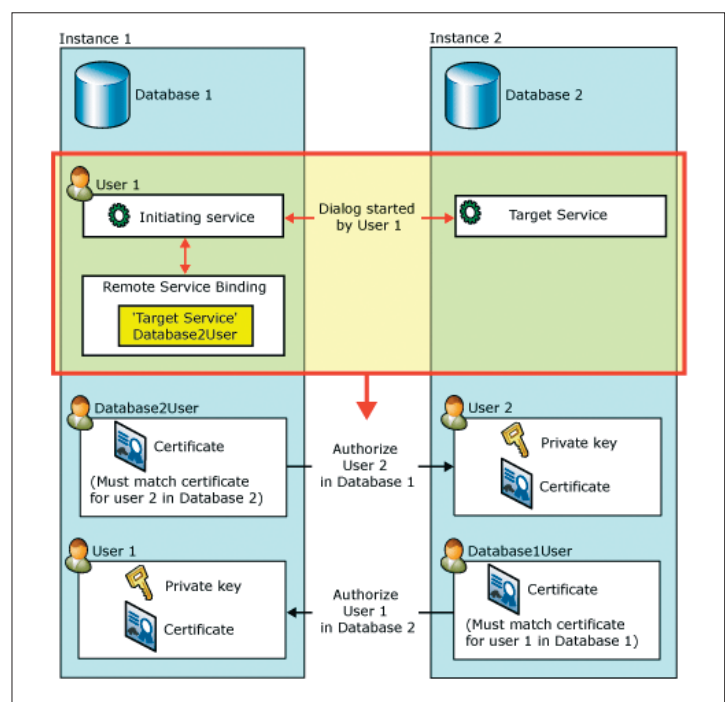
- Full Security,
- Anonymous Security,
- No Dialog Security.

Ohne Dialog Security muss auf alle Vorteile verzichtet werden. Anonyme Sicherheit bietet Verschlüsselung und Integritätsprüfung. Allerdings kann der aufgerufene Service nicht die Identität des aufrufenden Services prüfen. Deshalb läuft die Verbindung unter dem Gastkonto, das dann über entsprechende Rechte verfügen muss.

Der beste Weg ist sicher die volle Sicherheit. Dazu werden nicht in der Master-DB, sondern in der Datenbank, in der sich die SBR-Instanz befindet, analog zu Listing 1 auch zwei Zertifikate eingetragen. Abbildung 2 zeigt die Zusammenhänge.

Das Zertifikat des entfernten Servers wird wieder einem lokalen Anwender zugeordnet. Nach diesem Schritt existieren auf jedem SQL Server vier Zertifikate, und zwar je zwei in den Master-Datenbanken

Abbildung 2
Ablauf der Dialog Security.



und in der Tokyo/Kyoto-DB. Ein Zertifikat enthält den eigenen privaten Schlüssel und das zweite nur den öffentlichen Schlüssel des entfernten Servers.

Remote Service Binding

Für eine volle Dialog Security ist ein Remote Service Binding erforderlich. Näheres dazu finden Sie bei den SQL Server Books Online (BOL). Hier wird der Name eines entfernten Services mit einem lokalen Benutzer verknüpft.

Diesem Benutzer muss auch das Zertifikat der entfernten Datenbank zugeordnet sein, wie es Listing 2 demonstriert. Der Parameter *Anonymous* steuert dabei, welche Art von Dialog Security eingesetzt wird.

Ein weiterer wichtiger Parameter für die Dialog Security ist der *Encryption*-Parameter der *Begin Dialog*-Anweisung, der die Verschlüsselung steuert. Eine tabellarische Übersicht der Optionen finden Sie in den Books Online (BOL).

Listing 2

Ein Remote Service Binding erstellen.

```
-create a remote service binding
CREATE REMOTE SERVICE BINDING [Japan_Binding]
  TO SERVICE 'svcKyotoTanukiRequest'
  WITH USER = remcert,
  ANONYMOUS=off
-Create Endpoint for TCP Communication
create endpoint epTokyo
state = started
as TCP (LISTENER_PORT = 4022)
FOR SERVICE_BROKER (AUTHENTICATION = REQUIRED)
-Pause and start the Endpoint
ALTER ENDPOINT BrokerEndpoint
STATE = STOPPED ;
ALTER ENDPOINT BrokerEndpoint
STATE = STARTED ;
```

Nachrichten, die zwischen zwei SQL-Server-Instanzen versendet werden, werden über so genannte Endpoints mit TCP

verschickt. Damit der SQL Server auf Service-Broker-Nachrichten reagieren kann, muss ein Endpoint erstellt werden, wie es Listing 2 zeigt. Dabei wird auch der Port für die Kommunikation angegeben.

Befindet sich eine Firewall zwischen den Servern, muss der Port freigeschaltet sein. Mit *netstat -a* können Sie prüfen, ob der Port aktiv ist und lauscht. Die Kommunikation zu einem SBR können Sie mit *Alter Endpoint* unterbrechen.

Nachrichteninfrastruktur

Nachdem die Security nun vorbereitet ist, werden wie im ersten Teil des Artikels die Objekte für das Versenden der Nachrichten erstellt. Neu ist die Route in Listing 3. Als Zielangabe wird ja immer nur der Name des Ziel-Services angegeben. Um diesen Ziel-Service zu finden, sucht der SBR in der Systemtabelle *sys.routes* nach passenden Routen zu diesem Service.

Es gibt immer eine lokale Standardroute, deshalb funktionierte auch das Beispiel aus dem ersten Teil des Artikels. In diesem Fall möchte ich aber eine Nachricht an einen anderen Computer senden und deshalb gebe ich hier IP-Adresse und Port an. Der andere Computer muss natürlich auch einen Endpoint für diesen Port erstellt haben.

Nach all diesen vorbereitenden Schritten, die Sie auf beiden Servern durchführen müssen – siehe dazu auch den Kasten *Die Codebeispiele auf der Heft-CD ausführen* – können Sie nun endlich auch

Zertifikate erstellen

Für die Sicherheitseinstellungen benötigen Sie vier Zertifikate: TokyoTransport, TokyoDialog, KyotoTransport und KyotoDialog.

Für die Erstellung von Testzertifikaten können Sie den SQL Server oder das Tool *makecert* aus dem SDK verwenden. Vom .NET Command Prompt aus können Sie *makecert* aufrufen:

```
makecert -n "CN=Tokyo" -sv Tokyo.pvk Tokyo.cer
```

Damit erzeugen Sie neben der Zertifikatsdatei, die den öffentlichen Schlüssel enthält, auch noch eine Datei mit dem privaten Schlüssel. Der private Schlüssel wird nur lokal in die Datenbanken eingetragen. Die Zertifikatsdatei wird auch auf dem entfernten Server verwendet.

Wenn Sie die Zertifikatsdateien wie in Listing 1 gezeigt einbinden wollen, dann beachten Sie, dass das Dienststartkonto des SQL Servers die Berechtigungen haben muss, auf diese Dateien zuzugreifen. Deshalb legen Sie die Dateien am besten in einem lokalen Verzeichnis und nicht auf einem Netzlaufwerk ab. Nach dem Import in den SQL Server können Sie die Dateien löschen.

Alternativ dazu können Sie die Zertifikate auch über TSQL-Anweisungen erzeugen:

```
CREATE CERTIFICATE ctfTokyoMaster
  AUTHORIZATION [dbo]
  WITH SUBJECT='Instance certificate for transport security',
  ENCRYPTION_PASSWORD = 'Freude$2005';
```

Jetzt existiert das Zertifikat nur im lokalen Server, es muss aber auch auf dem entfernten Server eingetragen werden. Dazu können Sie das Zertifikat in eine Datei schreiben, diese können Sie dann auf den anderen Server kopieren und dann dort importieren. Die Private-Key-Datei muss dafür nicht angelegt werden.

```
DUMP CERTIFICATE ctfTokyoMaster TO FILE='c:\Tokyo.cer'
PRIVATE_KEY TO FILE = 'C:\Tokyo.prv'
USING PASSWORD = 'Freude$2005'
```

Listing 3

Objekte für den Nachrichtensend anlegen.

```
create message type [mtAskKyoto]
  validation = well_formed_xml
create message type [mtResponseTokyo]
  validation = well_formed_xml
create contract [ctrOrderTanuki](
  [mtAskKyoto] sent by initiator,
  [mtResponseTokyo] sent by target
)
create queue [quTokyo] with status = on
create service [svcTokyoTanukiResponse]
  on queue [quTokyo]( [ctrOrderTanuki] )
grant send on service::
[svcTokyoTanukiResponse] to remcert
create route [RouteTokyo]
with service_name = 'svcKyotoTanukiRequest',
address = 'TCP://192.168.0.3:4022'
select * from sys.routes
```

Nachrichten versenden. Das ist ähnlich einfach wie im ersten Teil des Artikels.

Der Unterschied ist, dass im Hintergrund die Route aktiv ist und Nachrichten an den Service *svcKyotoTanukiRequest* über die Route zu einem andern SQL Server gesendet werden. Den eigentlichen *Send*-Befehl müssen Sie dabei nicht anpassen. So lassen sich Routen leicht im laufenden Betrieb anpassen.

Als Erstes wechseln Sie in die Tokyo-Instanz und führen den Teil zum Senden einer Nachricht aus. Dazu markieren Sie den kompletten Teil aus dem Abschnitt *Sending the Message* und führen ihn mit [F5] aus. In der System-View *sys.conversation_endpoints* sollte ein neuer Endpoint mit dem Status *Conversing* und in *sys.dm_broker_transmission_status* eine neue Nachricht erscheinen.

Jetzt wechseln Sie in die Kyoto-Instanz auf dem anderen SQL Server und prüfen als Erstes die *quKyoto* in der *dbKyoto*. Im Abschnitt *Send a Response* finden Sie eine vorbereitete Anweisung. Kopieren Sie aus der Ausgabe der Anweisung das Conversa-

tion Handle, ersetzen Sie es in der *Send*-Methode und führen Sie diese dann aus.

Jetzt sollte in der Tokyo-Instanz die Antwort angekommen sein. Schließen Sie jetzt in der Kyoto-Instanz die Conversation mit dem aktuellen Conversation Handle. Der *conversation_endpoint* sollte jetzt den Status *Closed* haben.

In der Tokyo-Instanz ist jetzt eine weitere Nachricht vom Typ *EndDialog* angekommen. Mit dem Schließen der Conversation gehen auch die Nachrichten der Conversation verloren, aber nur auf der Seite, die die Conversation schließt. Deshalb sollten Sie die Nachrichten mit *Receive* vorher aus der Queue entnehmen. In der Tokyo-Instanz existieren die beiden Nachrichten noch, bis auch diese Instanz mit ihrem aktuellen Conversation Handle geschlossen wird.

Jeder neue Dialog erzeugt eine Conversation ID, die für alle Nachrichten innerhalb des Dialogs gleich ist. So können Nachrichten einem Dialog zugeordnet werden. Zusätzlich existiert auf jeder Seite des Dialogs ein unterschiedliches, lokales

Conversation Handle. Dieses Handle müssen Sie bei vielen Anweisungen angeben.

Den Status aller Conversations können Sie wie folgt abfragen:

```
select * from sys.conversation_endpoints
```

Trouble Shooting

Wenn die Nachricht nicht wie erhofft auf dem Zielsystem eintrifft, dann geht die Sucherei los. Es gibt einige Stellen, die Sie überprüfen sollten.

In der System Queue *sys.transmission_queue* sind ausgehende Nachrichten zu finden, bis diese versendet werden können. Gibt es Probleme mit dem Versand, dann lohnt ein Blick in die Queue. Vor allem das Feld *last_transmission_error* kann sehr hilfreich sein. Ist der Ziel-Service nicht erreichbar, zum Beispiel weil die IP-Adresse in der Route falsch ist, dann enthält das Feld folgende Meldung:

Connection attempt failed with error: '10060(Ein Verbindungsversuch ist fehlgeschlagen, da die Gegenstelle nach einer be-

Die Codebeispiele auf der Heft-CD ausführen.

Die Kommunikation wird zwischen zwei Virtual PCs durchgeführt, auf denen die Beta 2 (9.00.852) des SQL Server 2005 installiert ist. Beide Instanzen befinden sich in einer Workgroup. Stellen Sie sicher, dass die beiden Computer miteinander kommunizieren können, und notieren Sie sich deren IP-Adressen.

Auf der Heft-CD befinden sich zwei Ordner mit SQL-Dateien, die die TSQL-Anweisungen für ein verteiltes *HalloWelt* zwischen zwei Servern enthalten. Außerdem finden Sie Zertifikat- und Schlüsseldateien, die von den TSQL-Anweisungen benötigt werden.

Kopieren Sie jeweils einen der Ordner auf das Laufwerk C:\ einer Instanz und starten Sie dann das SQL Server Management Studio mit einem Doppelklick auf die Datei *TokyoDB.sql* beziehungsweise *KyotoDB.sql*. Führen Sie die Anweisungen der Dateien Schritt für Schritt aus, nicht alle auf einmal.

Der Code dieses Artikels basiert auf der originalen SQL Server Beta 2. Mit der Beta 2 von Visual Studio wird eine neuere Version, eine Community Technical Preview (CTP) ausgeliefert, in dieser Version ändern sich ein paar kleine Syntaxdetails. Dazu zählen die Anweisungen *Create Certificate* und *Create Endpoint*.

Create Certificate

- Private_Key -> Private Key
- decryption_password -> decryption by password

Create Endpoint

- Authentication Required -> AUTHENTICATION = CERTIFICATE ctfKyotoMaster

Aktualisierungen finden Sie unter [3].

stimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung war fehlerhaft, da der verbundene Host nicht reagiert hat.)'.

Kann der Zielservicename keiner Route zugeordnet werden und existiert auch nicht lokal dann erscheint: *The target service broker is unreachable.*

Wird die Route korrigiert, dann wird die Nachricht nach einiger Zeit erfolgreich versendet und danach sollte diese Queue wieder leer sein.

Auch ein Blick in die View *sys.service_broker_endpoints* lohnt sich. Hier können Sie kontrollieren, ob der Endpoint aktiv ist.

Sind Sie auf diesem Wege nicht fündig geworden, dann sollten Sie nun das Eventlog auf beiden SQL Servern zu Rate

ziehen. Dort erscheinen fehlerhafte Login-Versuche, nicht erreichbare Endpoints und andere Fehlerursachen. Am besten, Sie filtern die Log-Datei, da sonst zu viele Einträge erscheinen. Über den Performance Monitor oder auch eine .NET-Anwendung können Sie einige SBR-Counter, wie die Anzahl der versendeten Nachrichten, überwachen.

In Aktivierungsprozeduren mit *MAX_QUEUE_READERS>1* sollten Sie beim Eintreffen neuer Nachrichten das *WaitFor*-Statement vor dem *Receive* verwenden. Sonst kann es vorkommen, dass bei leerer Queue keine Nachricht zurückgeliefert wird, da mehrere QueueReader gleichzeitig aktiv sein können. In diesen Fällen muss die Prozedur darauf reagieren können und sollte sich am besten beenden. Treffen neue Nachrichten ein, dann wird wieder eine Instanz aktiv.

Es ist möglich, die Nachrichten bis zum Ende einer Conversation in der Queue zu belassen (message retention) aber das kostet Ressourcen für den Broker-Mechanismus. Es ist besser, die wichtigen Informationen in einer separaten Tabelle zwischenspeichern und die Nachrichten aus den Queues zu entnehmen.

Beenden Sie Conversations, wenn diese nicht mehr benötigt werden. SBR überwacht alle offenen Conversations

und das kostet auch Ressourcen, die sich in der Masse summieren.

Versuchen Sie innerhalb einer Transaktion nicht zu viele Nachrichten zu versenden, halten Sie die Dialoge und Transaktionen kurz.

Alle diese Tipps werden wichtiger, je mehr Nachrichten Sie innerhalb kurzer Zeit austauschen müssen. Für die ersten Testversuche oder wenig belastete Systeme spielen sie keine bedeutende Rolle. Aufwand und Nutzen sollten sich wie immer die Waage halten.

Ein Hinweis noch zum Forwarding. SBR ist auch in der Lage, Nachrichten über mehrere SQL-Server-Instanzen hinweg zu einem Zielserver zu übertragen. Dieser Mechanismus wird Forwarding genannt. Er dürfte allerdings eher selten zum Einsatz kommen. Ein häufiges Szenario dürften allerdings SQL-Server-Express-Versionen sein, die mit einer oder zwei SQL-Server-Standard-Versionen über SBR Nachrichten austauschen, hier ist kein Forwarding nötig.

Fazit

Der SBR stellt eine interessante Infrastruktur zum Nachrichtenaustausch zur Verfügung, auf die in der Praxis etliche Aufgaben warten. Die Verwendung des SBR von einer .NET-Anwendung aus konnte hier noch nicht gezeigt werden, aber die TSQL-Befehle, die in den Beispielen zum SQL Server gesendet wurden, können genauso auch von einer .NET-Client-Anwendung geschickt werden.

Eine Alternative dazu sind die SQL Server Management Objects (SMO) für den Zugriff auf SBR.

Es existiert noch kein richtiges GUI für den Service Broker, weshalb der Zugriff auf alle wichtigen Elemente händisch erfolgen muss. Aber ich bin mir sicher, dass die Community in den nächsten Monaten an solchen Tools arbeiten wird.

Ich habe ein eigenes kleines Hilfsprojekt gestartet, das mir viel Arbeit abnimmt, bin aber noch nicht ganz fertig. Ich werde unter [3] über den Fortschritt informieren.

||||||

XML und Unicode

Im ersten Teil des Artikels wurde beschrieben, wie Sie XML-Text der Send-Methode übergeben müssen. Viel einfacher als das Casten ist es, eine Variable vom Typ XML zu verwenden. Nähere Infos dazu bietet der Code auf der Heft-CD.

[1] Marcel Gnoth, Der Logikmagnet, Skalierbare nachrichtenbasierte Systeme mit SQL Server 2005 entwickeln, dotnetpro 6/2005, Seite 69 ff.

[2] SQL Server 2005 Beta News Groups: msdn.microsoft.com/SQL/2005

[3] Marcel's SBR Page: www.gnoth.net/ServiceBroker