Betrouwbare Services bouwen met SQLServer

Frans van der Geer en André van Leeuwen

CF is tegenwoordig de standaardkeuze op het Microsoft-platform voor het maken van services. Maar Service Broker is een serieus alternatief voor niet publieke services. Dit artikel beschrijft de gebruikte techniek: SQL Service Broker, XML-datamanipulatie in T-SQL en een C# stored procedures in combinatie met Linq ter implementatie van de service. Het artikel is van toepassing op SQLServer 2005 en 2008.

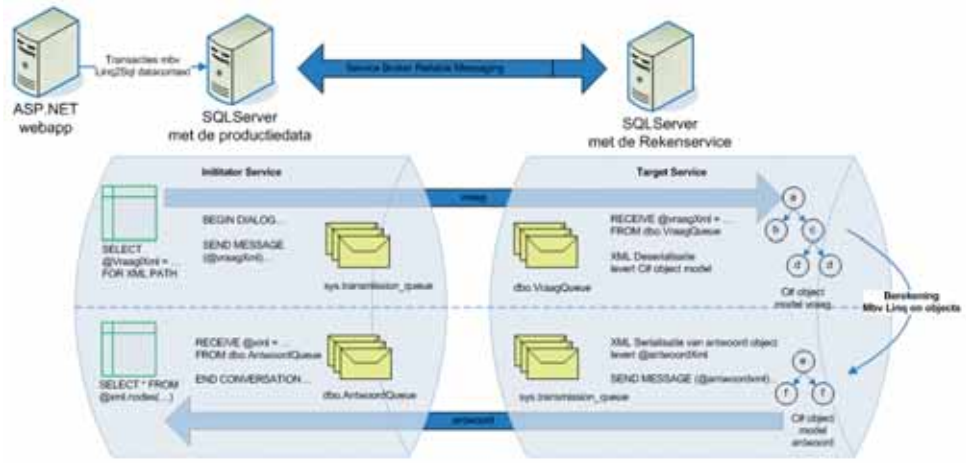
We bekijken de praktijksituatie van de salarisverwerker SDB Groep in de zorgsector. Voor deze dienstverlener is een rekenservice gerealiseerd waarmee alle salariscomponenten opnieuw worden uitgerekend zodra het werkrooster van een zorgverlener wordt gewijzigd. De overheid streeft naar zoveel mogelijk handen aan het bed tegen zo laag mogelijke kosten, dus inzicht in de loonkosten helpt een zorginstelling om de zorgverlening optimaal te plannen. Aan het eind van elke salarisperiode worden de rekenresultaten als invoer gebruikt voor de salarisverwerking.

Figuur 1 geeft de oplossing schematisch weer. Een gebruiker bewerkt een werkrooster via de webapplicatie. Dat resulteert in een databasetransactie om het gewijzigde rooster weg te schrijven. Als gevolg daarvan moeten er salariscomponenten opnieuw berekend worden. Dat wordt gedaan door asynchroon een Rekenservice aan te roepen die is geïmplementeerd als een C# stored procedure in een andere database op een andere server om de online database zo min mogelijk te belasten. Verderop in dit artikel zal de oplossing stap voor stap worden toegelicht. Maar nu volgt eerst enige uitleg van Service Broker.

Service Broker is een wat minder bekend onderdeel van SQLServer dat sinds versie 2005 aan het product is toegevoegd. Menig lezer zal ooit een tabel met een statusveld hebben geïntroduceerd om dit via een SQLServer Agent Job te laten pollen om daarmee een stored procedure te starten zodra gedetecteerd wordt dat er records zijn die aan bepaalde condities voldoen. Dat is een scenario dat nu beter met Service Broker kan worden opgelost. Service Broker biedt de mogelijkheid om asynchroon services te laten uitvoeren via message queueing. Een service wordt geïmplementeerd met een stored procedure die gekoppeld is aan een queue. Een queue is in feite een speciaal soort tabel met een vooraf gedefinieerd schema, waarbij de belangrijkste velden het berichttype en het bericht zelf zijn. Een service kan geïnitieerd worden door een bericht naar een service (lees: queue) te sturen vanuit een databasetransactie. Het verwerken van het bericht uit de queue wordt vervolgens in een nieuwe transactie gedaan door een via Service Broker geïnitieerde stored procedure. Met andere woorden: Service Broker biedt de mogelijkheid tot het asynchroon uitvoeren van databasetransacties.

Het SELECT-statement kan worden gebruikt om in een queue te kijken. Maar T-SQL heeft wel uitbreidingen ondergaan. Bijvoorbeeld de statements CREATE QUEUE, SEND en RECEIVE om respectievelijk een queue aan te maken en berichten op een queue te plaatsen dan wel er vanaf te halen. Enkele voorbeelden volgen later in dit artikel.

Er is locking van toepassing op berichten in de queue, zodat een bericht pas zichtbaar wordt nadat de versturende transactie gecommit is. Service Broker kan ervoor zorgen dat een stored procedure automatisch gestart wordt zodra een bericht op de queue wordt vrijgegeven. Dat doet Service Broker heel efficiënt. Er wordt niet steeds voor elk nieuw bericht een nieuwe stored procedure instantie gestart. Er wordt pas een extra instantie gestart als blijkt dat de reeds actieve instanties het tempo waar-



FIGUUR 1: SCHEMATISCHE WEERGAVE VAN DE OPLOSSING

mee berichten arriveren niet kunnen bijhouden. De database administrator kan het maximum aantal instanties per queue configureren, zodat er niet ongewenst veel resources gebruikt worden.

De verzendende en ontvangende service kunnen in dezelfde database geïmplementeerd zijn, maar ook in verschillende databases en zelfs op een andere SQLServer, zoals in Figuur 1. Een service kan bovendien worden geïmplementeerd op verschillende SQLServers tegelijk. Een bericht zal dan round-robin worden verzonden naar één van de servers om daar afgehandeld te worden. Een service kan hoog beschikbaar worden gemaakt door simpelweg de database hoog beschikbaar te maken met behulp van server clustering of database mirroring.

Waarom Service Broker?

De business casus vereist dat de rekenservice asynchroon, maar gegarandeerd wordt uitgevoerd. Asynchroon, omdat het werk van de planner niet gehinderd mag worden door het feit dat gekozen is om bij het opslaan van een rooster alles opnieuw te laten berekenen. En gegarandeerde verwerking, omdat het om echt geld en om echte verlofdagen gaat.

Eén van de eigenschappen van Service Broker is dat het lezen van de queue of het schrijven naar de queue in dezelfde databasetransactie kan plaatsvinden als het manipuleren van de data in de tabellen van de database. Vergelijk dat eens met een WCF-service die via het MSMQ-transport ook gegarandeerde message verwerking kan bieden, maar alleen in combinatie met de Distributed Transaction Coordinator. Dat geeft een veel minder goede performance dan Service Broker. Vanwege de extra benodigde componenten buiten SQLServer is deze oplossing ook moeilijker te configureren en te beheren.

Het doorslaggevende argument om voor Service Broker te kiezen boven WCF met MSMQ is dat de berekeningen gestuurd worden door verschillende databasetabellen. Als er iets wijzigt in één van die tabellen, moeten berekeningen opnieuw worden gedaan. Met Service Broker is dat eenvoudig te realiseren door een bericht naar een service te sturen vanuit een database trigger.

Maar Service Broker heeft nog meer voordelen ten opzichte van WCF met MSMQ. Wat Service Broker extra heeft ten opzichte van andere messaging systemen is het concept van een dialoog. Als twee services met elkaar moeten gaan communiceren, moet vooraf eerst een contract worden gedefinieerd. Dat beschrijft welke berichttypen er tussen twee services uitgewisseld kunnen worden en met welke berichttypen een dialoog kan beginnen. Service Broker dwingt af dat een dialoog tussen twee services altijd voldoet aan het contract. De ontvangende service kan er dus op rekenen dat ze altijd berichttypen ontvangt die ze kan verwerken. Service Broker garandeert bovendien dat er maar één serviceinstantie tegelijk actief kan zijn met verwerking van berichten van dezelfde dialoog. Daarmee worden problemen met volgordelijkheid van verwerking voorkomen. Een dialoog zou bijvoorbeeld kunnen bestaan uit het eerst versturen van een orderheader, gevolgd door het versturen van één of meer orderregels. Zonder de garantie die Service Broker biedt zou het kunnen zijn dat de eerste orderregel al wordt verwerkt door een tweede serviceinstantie, terwijl de eerste serviceinstantie nog bezig is met de verwerking van de header. Berichten van verschillende dialogen kunnen wel parallel door verschillende serviceinstanties worden verwerkt.

In sommige situaties is de garantie van volgordelijke verwerking binnen een dialoog nog niet voldoende. Bijvoorbeeld in de praktijk van de salarisverwerker, wanneer een planner twee keer achter elkaar een berekening voor dezelfde zorgverlener in dezelfde salarisperiode kan starten door twee wijzigingen achter elkaar uit te voeren op hetzelfde werkrooster. Elke berekening start een nieuwe dialoog met de rekenservice. De salarisverwerker moet er zeker van kunnen zijn dat alleen het rekenresultaat van de laatste situatie wordt bewaard. Dat is mogelijk met het Service Broker concept Conversation Group. Berichten van verschillende dialogen die onderdeel worden gemaakt van dezelfde Conversation Group, zullen gegarandeerd door één serviceinstantie tegelijk worden afgehandeld. Dat werkt op basis van een Conversation Group lock, die een Service Broker service automatisch vraagt zodra ze het eerste nieuwe bericht van een Conversation Group leest. Er is maar één serviceinstantie die deze exclusieve lock kan krijgen. Andere berichten binnen dezelfde Conversation Group blijven verborgen voor andere serviceinstanties, totdat de ene serviceinstantie geen berichten voor die Conversation Group meer in de queue vindt en de lock loslaat. Er kunnen daarna nog meer berichten arriveren binnen dezelfde Conversation Group. De eerste de beste serviceinstantie die daarvan het eerste nieuwe bericht te pakken krijgt, plaatst opnieuw een lock en gaat er mee aan de slag. Een Conversation Group bestaat uit niets anders dan een unique identifier. De oplossing voor de salarisverwerker is dus om een Conversation Group ID per combinatie van medewerker en salarisperiode in de database te administreren en te zorgen dat elke dialoog met de juiste Conversation Group ID wordt geïnitieerd.

Is SQLServer wel geschikt om complexe berekeningen te doen?

SQLServer is erg goed in setsgewijze datamanipulaties met behulp van T-SQL, maar berekeningen die meer vergen dan een paar aggregatiefuncties kunnen beter in een procedurele taal worden geschreven. De SQLCLR komt daarbij goed van pas. Stored procedures kunnen in C# worden geschreven en de .NET-assembly kan in de database worden ondergebracht. De SQLCLR kan de code uitvoeren alsof het een T-SQL procedure is. Procedurele code is dus krachtig. Maar aan de andere kant: berekeningen op basis van een verzameling ingeplande diensten zijn toch ook weer setsgewijs. Het zou mooi zijn als Linq in de database gebruikt zou kunnen worden, zodat setsgewijze expressies gecombineerd kunnen worden met ingewikkelde rekenkundige expressies.

SQLServer 2005 weet niets van het bestaan van Linq. Maar Linq is niets anders dan een compiler truck die gewone CLR 2.0 compatible code oplevert met referenties naar een assembly uit het .Net 3.5 Framework. En de SQLServer CLR is compatible met de CLR 2.0. Dus het enige dat moet gebeuren om Linq te ondersteunen vanuit de SQLServer CLR is het registreren van de betreffende .Net 3.5 Framework assembly in SQLServer. Codevoorbeeld 1 laat zien dat dit eenvoudig is.

```
use master
go

-- CLR ondersteuning aanzetten
EXEC sp_configure 'clr enabled', 1
RECONFIGURE
go

IF @@VERSION NOT LIKE 'Microsoft SQL Server 2008%'
BEGIN
  -- Trustworthy setting is nodig om
  UNSAFE assemblies te mogen gebruiken.
```

```
  -- Een assembly is UNSAFE als hij
  afhankelijk is van een assembly die
  niet op de trusted list staat.
  ALTER DATABASE [$(TargetDB)] SET
  TRUSTWORTHY ON
END
go

USE [$(TargetDB)]
go

IF @@VERSION NOT LIKE 'Microsoft SQL
Server 2008%'
BEGIN
  -- Om Linq in de database te kunnen
  gebruiken is onderstaande .NET 3.5
  assembly nodig.
  -- Omdat deze assembly nog niet op de
  trusted assembly list staat van SQL Ser-
  ver 2005,
  -- is de enige optie om hem in UNSAFE
  mode te gebruiken.
  -- N.B. de $ parameter vereist dat SSMS
  in SQLCMD mode staat (zie de button
  -- met het rode uitroepteken op de tool-
  bar)
  IF NOT EXISTS(
    SELECT 1
    FROM sys.assemblies
    WHERE name = 'System.Core')
  BEGIN
    CREATE ASSEMBLY [System.Core]
    FROM '$(ProgramFiles)\Reference Assem-
    blies\Microsoft\Framework\v3.5\System.
    Core.dll'
    WITH PERMISSION_SET = UNSAFE
  END
END
go
```

CODEVOORBEELD 1: REGISTREREN VAN DE ASSEMBLY MET LINQ EXTENSION METHODS IN SQLSERVER 2005

Ten eerste moet de SQLCLR worden aangezet. Vervolgens kan de assembly worden geregistreerd. De registratie van de .Net 3.5 assembly is niet nodig voor SQLServer 2008, omdat deze de benodigde assembly standaard op zijn vertrouwde lijst met assemblies heeft staan. Dit is niet het geval voor SQLServer 2005 en daarom moet de assembly als UNSAFE worden gemarkeerd. Een database administrator die daar huiverig voor is, moet zich realiseren dat deze settings alleen nodig zijn op de SQLServer instantie die de Rekenservice host. En dat is natuurlijk een andere dan waarin alle productiedata zit. Die instantie draait zelfs op een andere server, om de primaire databaseserver niet te belasten met complexe berekeningen. Dus nog een extra SQLServer licentie erbij? Nee hoor, een gratis SQLServer Express Edition is voldoende mits de limiet van 1 CPU geen probleem is. SQLService Broker werkt ook volledig op een SQLServer Express Edition, mits tenminste een Standard Edition in een dialoog betrokken is. En de SQLServer met de productiedata is dat.

Vervolgens is de rekenservice in een aantal stappen gemaakt.

1) Startpunt is een XML-Schema met de definitie van een drietal berichten. Ten eerste het vraagbericht met daarin alle data die nodig is voor een berekening. Ten tweede het antwoordbericht voor de rekenresultaten en ten derde een foutbericht mocht de berekening om een of andere reden niet lukken.
2) Op basis van het XML-Schema is met behulp van het utility XsdObjectGen een C# objectmodel gegenereerd. Alle berekeningen zullen geschreven worden op dit objectmodel.
3) De data wordt aangeleverd als een XML-stream. Om de XML om te zetten in een instantie van een objectmodel en vice versa zijn XML (de)serializers nodig. Het .Net framework kan dynamisch XML (de)serializers genereren, maar de SQLCLR ondersteunt geen dynamisch gegenereerde code, dus in plaats daarvan zijn de serializers van tevoren uitgegenereerd met behulp van het utility XgenPlus.
4) Tenslotte is met behulp van Visual Studio 2008 een C# databaseproject gemaakt. Dit projecttype bevat onder andere een template voor het maken van een CLR stored procedure. Daarbij moet bij de eigenschappen van het project gekozen worden voor .Net 3.5 als doelplatform; Linq behoort dan tot de mogelijkheden.

Codevoorbeeld 2 toont de C# stored procedure met de code om de XML-stream naar het objectmodel te converteren en vice versa. Daarbij wordt gebruik gemaakt van de gegenereerde XML-serializer classes in de namespace SDB.Mplus.Rekenserver.ObjectModel.Serializers. Het T-SQL type *xml* mapt op het .Net type *System.Data.SqlTypes.SqlXml*. Nadat deze 'plumbing' code is geschreven kan de daadwerkelijke implementatie van de rekenlogica getypeerd worden uitgeschreven tegen het objectmodel dat op basis van het XML-schema gemaakt is.

```
using System;
using System.Data.SqlTypes;
using System.Xml;
using System.IO;
using System.Linq;
using SDB.MPlus.RekenServer.ObjectModel;
using Helper = SDB.MPlus.RekenServer.
ObjectModel.Serializers;

public partial class StoredProcedures
{
  [Microsoft.SqlServer.Server.
  SqlProcedure]
  public static void BerekenLoon
  factoren(SqlXml vraag, out SqlXml
  antwoord)
  {
    // Creer het objectmodel uit de xml
    van de vraag
    XmlReader reader = vraag.Create-
    Reader();
    Helper.LoonfactorBerekeningVraag.
    LoonfactorBerekeningVraagSerializer
    vraagSerializer = new Helper.
    LoonfactorBerekeningVraag.Loonfactor
    BerekeningVraagSerializer();
    LoonfactorBerekeningVraag vraagObject
    = vraagSerializer.Deserialize(reader)
    as LoonfactorBerekeningVraag;

    // Bereken het antwoord
    LoonfactorBerekeningAntwoord
    antwoordObject = BerekenLoonfactoren-
    Implementatie(vraagObject);

    // Converteer het antwoord naar xml
    Helper.LoonfactorBerekeningAntwoord.
    LoonfactorBerekeningAntwoord-
    Serializer antwoordSerializer = new
    Helper.LoonfactorBerekeningAntwoord.
    LoonfactorBerekeningAntwoord-
    Serializer();
    MemoryStream antwoordStream = new
    MemoryStream();
    antwoordSerializer.Serialize
    (antwoordStream, antwoordObject);
    antwoordStream.Flush();
    antwoordStream.Position = 0;
    antwoord = new SqlXml
    (antwoordStream);
  }


  /// <summary>
  /// De type-safe implementatie van de
  LoonfactorBerekening
  /// </summary>
  private static LoonfactorBerekening-
  Antwoord BerekenLoonfactoren-
  Implementatie(LoonfactorBerekeningVraag
  vraag)
  {
    // Implementatie code weggelaten
    return new LoonfactorBerekening-
    Antwoord();
  }
}
```

CODEVOORBEELD 2: C# STORED PROCEDURE MET XML-SERIALIZATIE

Om een idee te krijgen hoe vervolgens Linq op het objectmodel leidt tot leesbare en dus onderhoudbare code is in Codevoorbeeld 3 een codefragment weergegeven. Deze expressie had nog prima in T-SQL uitgeschreven kunnen worden, maar het gaat om het idee.

```
int aantalDagenGewerkt =
    (from d in vraag.DienstCollection.
    Cast<Dienst>()
    from tb in d.TijdblokCollection.
    Cast<Tijdblok>()
    where (tb.DienstType.TeltMeeIn-
    GewerkteUren && !tb.IsZiek)
    select d.Datum.Date).Distinct().
    Count();
```

CODEVOORBEELD 3: EEN BEREKENING OP SETSGEWIJZE DATA MET BEHULP VAN LINQ

Nadat de C# stored procedure is voltooid en tot een library assembly is gecompileerd, moet hij in SQLServer worden gedeployed. Codevoorbeeld 4 toont hoe dat moet. Eerst moet de assembly worden geregistreerd en vervolgens kan een T-SQL stored procedure wrapper om de C# static methode worden gemaakt. Omdat deze assembly afhankelijk is van de .Net 3.5 System.Core assembly en deze door SQLServer 2008 wel standaard wordt vertrouwd, wordt bij het registreren van de assembly onderscheid gemaakt tussen het benodigde permission level in SQLServer 2005 en 2008.

```
IF @@VERSION LIKE 'Microsoft SQL Server
2008%'
BEGIN
  CREATE ASSEMBLY [SDB.MPlus.RekenServer.
  LoonfactorBerekening]
  FROM '$(RekenserverAssemblyDir)\ SDB.
  MPlus.RekenServer.LoonfactorBerekening.
  dll'
  WITH PERMISSION_SET = SAFE
END
ELSE IF @@VERSION LIKE 'Microsoft SQL
  Server 2005%'
BEGIN
  CREATE ASSEMBLY [SDB.MPlus.RekenServer.
  LoonfactorBerekening]
  FROM '$(RekenserverAssemblyDir)\ SDB.
  MPlus.RekenServer.LoonfactorBerekening.
  dll'
  WITH PERMISSION_SET = UNSAFE
END
go

CREATE PROCEDURE dbo.BerekenLoonfactoren
(
  @Vraag xml
, @Antwoord xml OUTPUT
)
-- [assembly name].[fully qualified type
name].[method name]
AS EXTERNAL NAME [SDB.MPlus.RekenServer.
LoonfactorBerekening].[StoredProcedures].
[BerekenLoonfactoren]
go
```

CODEVOORBEELD 4: REGISTRATIE VAN DE ASSEMBLY EN DE C# STORED PROCEDURE IN SQLSERVER

Om deze stored procedure asynchroon via Service Broker te kunnen aansturen, moeten een aantal Service Broker artifacts worden gemaakt, zoals aan het begin van dit artikel is beschreven. Codevoorbeeld 5 toont een gedeelte van de code aan de serverzijde; het aanmaken van een aantal berichttypen, een contract, een queue en een service. Er is geen ruimte om alles te laten zien. Wat bijvoorbeeld ontbreekt, is het maken van een Service Broker endpoint waarmee SQLServer berichten zal ontvangen, alsmede de routes waarmee een bericht naar de juiste service wordt geleid. Dit is wel beschreven in de SQLServer Books Online.

```
IF NOT EXISTS(SELECT 1 FROM sys.service_
message_types WHERE name = '/Rekenserver/
LoonfactorBerekening/Vraag')
  CREATE MESSAGE TYPE [/Rekenserver/Loon-
  factorBerekening/Vraag] VALIDATION =
  WELL_FORMED_XML

IF NOT EXISTS(SELECT 1 FROM sys.service_
message_types WHERE name = '/Rekenserver/
LoonfactorBerekening/Antwoord')
  CREATE MESSAGE TYPE [/Rekenserver/Loon-
  factorBerekening/Antwoord] VALIDATION =
  WELL_FORMED_XML

IF NOT EXISTS(SELECT 1 FROM sys.service_
message_types WHERE name = '/Rekenserver/
Fout')
  CREATE MESSAGE TYPE [/Rekenserver/Fout]
  VALIDATION = WELL_FORMED_XML

IF NOT EXISTS(SELECT 1 FROM sys.service_
contracts WHERE name = '/Rekenserver/Loon-
factorBerekening/DataContract')
  CREATE CONTRACT [/Rekenserver/Loon-
  factorBerekening/DataContract]
  (
    [/Rekenserver/LoonfactorBerekening/
    Vraag] SENT BY INITIATOR
  , [/Rekenserver/LoonfactorBerekening/
  Antwoord] SENT BY TARGET
  , [/Rekenserver/Fout] SENT BY TARGET
  )

IF NOT EXISTS(SELECT 1 FROM sys.service_
  queues WHERE name = 'Loonfactor-
  BerekeningRekenQueue')
  CREATE QUEUE dbo.LoonfactorBerekening-
  RekenQueue

-- Rekenserver moet berichten kunnen
ontvangen
GRANT RECEIVE ON dbo.LoonfactorBerekening-
RekenQueue TO RekenserverUser

-- lokaal service endpoint aanmaken
IF NOT EXISTS(SELECT 1 FROM sys.services
WHERE name = '/Rekenserver/Loonfactor-
Berekening/RekenService')
BEGIN
  CREATE SERVICE [/Rekenserver/Loon-
  factorBerekening/RekenService]
  AUTHORIZATION RekenserverUser
  ON QUEUE dbo.LoonfactorBerekeningReken-
  Queue
  (
    [/Rekenserver/LoonfactorBerekening/
    DataContract]
  )
END

-- MPlus moet messages kunnen versturen
naar de reken service
GRANT SEND ON SERVICE::[/Rekenserver/Loon-
factorBerekening/RekenService] TO
MPlusUser
```

CODEVOORBEELD 5: REGISTRATIE VAN SERVICE BROKER ARTIFACTS

Vervolgens moet er een stored procedure worden geschreven die geactiveerd wordt als er een bericht in de queue verschijnt. Codevoorbeeld 6 toont deze stored procedure. De stored procedure is zo geschreven dat hij niet voor elk afzonderlijk bericht hoeft te worden geactiveerd. Als hij eenmaal is opgestart, blijft hij berichten uit de queue lezen zolang die er nog zijn (WHILE @@ERROR = 0). Het lezen van een bericht gebeurd via het RECEIVE statement. Met het nieuwe WAITFOR statement kan de maximale wachttijd worden geregeld. In het voorbeeld wordt de stored procedure beeindigd zodra er gedurende één seconde lang geen nieuwe berichten meer in de queue verschijnen. Elk bericht wordt apart transactioneel afgehandeld. Het daadwerkelijk afhandelen van het bericht wordt overgelaten aan de C# stored procedure waarvoor in Codevoorbeeld 4 een T-SQL wrapper gemaakt is.

```
-- xml type methoden vereisen deze
setting:
SET QUOTED_IDENTIFIER ON
go

-- Deze stored procedure is bedoeld als
"activating" stored procedure
-- voor de queue dbo.LoonfactorBerekening-
Queue.
-- Op deze queue komen alle verzoeken
binnen voor een loonfactorberekening.
CREATE PROCEDURE dbo.LoonfactorBerekening-
RekenQueueReader
AS
BEGIN
   IF @@TRANCOUNT > 0
   BEGIN
     RAISERROR('dbo.Loonfactor-
     BerekeningRekenQueueReader mag
     niet vanuit een transactie gestart
     worden', 16, 1)
     ROLLBACK TRANSACTION
     RETURN 1
   END

   WHILE (@@ERROR = 0)
   BEGIN
     DECLARE @DialogHandle
     UNIQUEIDENTIFIER
     DECLARE @RequestMessage xml
     DECLARE @ResponseMessage xml
     DECLARE @MessageType sysname

     BEGIN TRANSACTION

     BEGIN TRY

       WAITFOR
       (
         RECEIVE TOP(1)
           @DialogHandle =
           conversation_handle,
           @RequestMessage =
           message_body,
           @MessageType = message_
           type_name
         FROM dbo.Loonfactor-
         BerekeningRekenQueue
       ), TIMEOUT 1000

       IF @@ROWCOUNT = 0
       BEGIN
         -- Geen berichten meer in
         de queue
         ROLLBACK TRANSACTION
         BREAK;
       END

       -- Voor testdoeleinden:
       -- SELECT @MessageType as
       RequestMessageType, @Request-
       Message as RequestMessage
```

```
    IF @MessageType = N'/SDB/MPlus/
    Rekenserver/Loonfactor-
    Berekening/Vraag'
    BEGIN
      EXEC BerekenLoonfactoren @
      RequestMessage, @Response-
      Message OUTPUT;

      SEND ON CONVERSATION
      @DialogHandle
      MESSAGE TYPE [/SDB/MPlus/
      Rekenserver/Loonfactor-
      Berekening/Antwoord]
      (@ResponseMessage);
    END
    ELSE IF @MessageType =
    N'http://schemas.microsoft.com/
    SQL/ServiceBroker/EndDialog'
    BEGIN
      -- Service initiator heeft
      de dialoog beeindigd. Dan
      doen wij dat hier ook.
      END CONVERSATION
      @DialogHandle;
    END
    COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    IF XACT_STATE() = -1
    BEGIN
      -- huidige transactie is
      niet committable
      -- Dit zou nooit mogen
      gebeuren. Gebeurt het wel, dan
      -- wordt de foutmelding in
      de SQLServer log geschreven.
      PRINT ERROR_PROCEDURE()
      PRINT ERROR_LINE()
      PRINT ERROR_MESSAGE()
      ROLLBACK TRANSACTION
    END

    BEGIN TRANSACTION

      -- Verwerk de fout,
      desnoods buiten de
      oorspronkelijke transactie om
      DECLARE @FoutBericht xml;

      WITH XMLNAMESPACES (DEFAULT
      'http://www.sdbnieuws.nl/
      MPlus/Rekenserver/2008')
      SELECT @FoutBericht =
      (
        SELECT
          ERROR_NUMBER()
          'ErrorNumber'
        ,  ERROR_SEVERITY()
        'ErrorSeverity'
        ,  ERROR_STATE() 'Error-
        State'
        ,  ERROR_PROCEDURE()
        'ErrorProcedure'
        ,  ERROR_LINE() 'ErrorLine'
        ,  ERROR_MESSAGE() 'Error-
        Message'
        FOR XML PATH ('Reken-
        serverFout')
      );

      SEND ON CONVERSATION
      @DialogHandle
      MESSAGE TYPE [/SDB/MPlus/
      Rekenserver/Fout]
      (@FoutBericht);

      END CONVERSATION @Dialog-
      Handle;

    COMMIT TRANSACTION
```

```
      IF XACT_STATE() = 1
      BEGIN
        -- huidige transactie is
        wel committable
        COMMIT TRANSACTION
      END
    END CATCH

  END;  -- WHILE
END
GO

GRANT EXECUTE ON dbo.LoonfactorBerekening-
RekenQueueReader TO RekenServerUser
go
```

CODEVOORBEELD 6: STORED PROCEDURE OM BERICHTEN VAN DE QUEUE TE VERWERKEN

Tenslotte moet de stored procedure aan de queue worden gekoppeld en is de Rekenservice is klaar voor gebruik. Codevoorbeeld 7 laat zien hoe dat moet.

```
ALTER QUEUE dbo.LoonfactorBerekeningReken-
Queue
WITH ACTIVATION
(
  STATUS = ON,
  PROCEDURE_NAME = [Loonfactor-
  BerekeningRekenQueueReader],
  MAX_QUEUE_READERS = 4,
  EXECUTE AS 'RekenserverUser'
)
```

CODEVOORBEELD 7: STORED PROCEDURE AAN DE QUEUE KOPPELEN

Daarbij kan worden gekozen om de stored procedure automatisch te laten starten zodra er een bericht op de queue verschijnt (ACTIVATION = ON). Als één instantie van de stored procedure het tempo waarmee berichten op de queue verschijnen niet kan bijhouden, zal Service Broker een extra instantie starten tot het aangegeven maximum (MAX_QUEUE_READERS). Omdat de stored procedure asynchroon wordt uitgevoerd is er geen client connectie en moet er expliciet voor een security context worden gekozen mbv EXECUTE AS.

De Rekenserver implementeert de service, maar aan de initiërende kant zijn ook nog een aantal zaken vermeldenswaardig. Bijvoorbeeld het initiëren van een nieuwe dialoog als onderdeel van een Conversation Group, zie Codevoorbeeld 8. De dialoog wordt gestart WITH RELATED_CONVERSATION_GROUP. Zoals hiervoor al is beschreven is dit om te garanderen dat twee achtereenvolgende berekeningen van hetzelfde rooster in dezelfde volgorde worden afgehandeld, zodat altijd het resultaat van de meest recente berekening zal worden bewaard.

```
DECLARE @RekenserverDialogId
uniqueidentifier;
DECLARE @ConversationGroupId
uniqueidentifier;
DECLARE @DienstverbandID int;   -- niet
getoond wordt hoe deze variabele een waar
de krijgt
DECLARE @PeriodeID int;    -- idem
DECLARE @Vraag xml;

-- Maak een conversation group voor deze
combinatie
-- van periode en dienstverband. Dit voor
komt dat
-- twee Rekenserver opdrachten voor de
zelfde combinatie elkaar gaan 'inhalen'
EXEC Rekenserver.LoonfactorBerekening_
ConversationGroupMaken
  @PeriodeID = @PeriodeID
,  @DienstverbandID = @DienstverbandID
,  @ConversationGroupID = @Conversation
GroupId OUTPUT

-- Maak een dialog voor het versturen van
de vraag
BEGIN DIALOG CONVERSATION @Rekenserver-
DialogId
FROM SERVICE [/Rekenserver/Loonfactor-
Berekening/DataService]
TO SERVICE '/Rekenserver/Loonfactor-
Berekening/RekenService'
ON CONTRACT [/Rekenserver/Loonfactor-
Berekening/DataContract]
WITH RELATED_CONVERSATION_GROUP =
@ConversationGroupId;

-- Stel de vraag samen
EXEC Rekenserver.LoonfactorBerekening_
VraagMaken
  @DienstverbandID = @DienstverbandID
,  @PeriodeID = @PeriodeID
,  @MessageID = @RekenserverDialogId
,  @Vraag = @Vraag OUTPUT

IF @Vraag IS NOT NULL
BEGIN
  -- Verstuur de vraag
  SEND ON CONVERSATION @Rekenserver-
DialogId
  MESSAGE TYPE [/Rekenserver/Loonfactor-
Berekening/Vraag](@Vraag);
END
ELSE
BEGIN
  -- Er valt niks te versturen
  END CONVERSATION @RekenserverDialogId
END
```

CODEVOORBEELD 8: START VAN EEN NIEUWE DIALOOG ALS ONDERDEEL VAN EEN CONVERSATION GROUP

Bij een Service Broker applicatie moet nadrukkelijk aandacht besteed worden aan foutafhandeling. Als het verwerken van een bericht van de queue vijf keer achter elkaar tot een rollback van de transactie leidt, zal Service Broker automatisch de queue disablen en dan stopt de verwerking volledig. Om dit te voorkomen moet de code zo geschreven worden dat het bericht linksom dan wel rechtsom kan worden verwerkt. En simpelweg committen van de transactie is niet altijd een optie; SQLServer staat niet toe dat de consistentie van de database in het gedrang komt. Codevoorbeeld 9 toont het foutafhandelingsgedeelte

van een queue reader stored procedure. Met de functie XACT_STATE() wordt bepaald wat de mogelijkheden zijn.

```
CREATE PROCEDURE Rekenserver.Loonfactor-
Berekening_DataQueueReader
AS
BEGIN
  IF @@TRANCOUNT > 0
  BEGIN
    RAISERROR('Rekenserver.Loonfactor-
    Berekening_DataQueueReader mag niet
    vanuit een transactie gestart
    worden', 16, 1)
    ROLLBACK TRANSACTION
    RETURN 1
  END

  WHILE (@@ERROR = 0)
  BEGIN
    DECLARE @DialogHandle uniqueidentifier
    DECLARE @ConversationGroupId
    uniqueidentifier
    DECLARE @Message xml
    DECLARE @MessageType sysname
    DECLARE @Foutmelding varchar(255)

    BEGIN TRANSACTION

    BEGIN TRY

      WAITFOR
      (
        RECEIVE TOP(1)
          @ConversationGroupId =
          conversation_group_id,
          @DialogHandle = conversation_
          handle,
          @Message = message_body,
          @MessageType = message_type_
          name
          FROM Rekenserver.Loonfactor-
          BerekeningDataQueue
        ), TIMEOUT 5000

        -- code met verwerking van het
        bericht is weggelaten

      COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
      IF ERROR_NUMBER() IN (1205)
      BEGIN
        -- Er is een herstelbare fout
        opgetreden;
        -- Probeer hetzelfde bericht nog
        een keer te verwerken
        -- N.B. Error 1205 = deadlock
        ROLLBACK TRANSACTION
        CONTINUE
      END
      IF XACT_STATE() = -1
      BEGIN
        -- huidige transactie is niet
        committable
        -- Dit zou nooit mogen gebeuren.
        Gebeurt het wel, dan
        -- wordt de foutmelding in de
        SQLServer log geschreven.
        SET @Foutmelding = 'Fout ' +
          CONVERT(varchar(10), ERROR_
          NUMBER()) + ' op regel ' +
          CONVERT(varchar(10), ERROR_
          LINE()) + ' van procedure ' +
          ERROR_PROCEDURE()
          PRINT @Foutmelding
          PRINT ERROR_MESSAGE()
          ROLLBACK TRANSACTION
          BREAK
        END
        ELSE
        BEGIN
          -- Huidige transactie is wel
          committable; leg de foutmelding
          vast
          UPDATE Rekenserver.Loonfactor-
          BerekeningLog
          SET Foutmelding = ERROR_MESSAGE()
          WHERE SsbDialogHandle = @Dialog
          Handle

          -- breek de dialoog af
          END CONVERSATION @DialogHandle;
          COMMIT TRANSACTION
        END
      END CATCH
    END; -- WHILE
  END
```

CODEVOORBEELD 9: HET FOUTAFHANDELINGSGEDEELTE VAN EEN QUEUE READER STORED PROCEDURE

Tenslotte zal getoond worden hoe makkelijk het is sinds SQLServer 2005 om met XML-data te werken.

Om XML-data te genereren is het SELECT statement met FOR XML PATH clause de meest flexibele variant. Een dataveld kan als ofwel een XML-attribuut dan wel als een XML-element worden opgenomen. Het wordt een attribuut als de alias met een @ begint. Het maken van geneste XML-elementen is met deze variant ook heel eenvoudig. Begin gewoon een geneste subselect op de plaats waar het subelement moet komen. De TYPE subclause in de SELECT zorgt ervoor dat de data als XML-data wordt gezien en niet als tekst. Als een XML-element dieper genest moet worden dan de (sub)select die het dataveld selecteert, kan met een relatieve xpath expressie in als alias de nesting expliciet gestuurd worden. Vandaar de naam FOR XML PATH. Codevoorbeeld 10 toont beide technieken. Let vooral op het dieper geneste Salaris element. De XML-namespace wordt bepaald via de WITH XMLNAMESPACES clausule.

```
DECLARE @Vraag xml, @MessageID
uniqueidentifier;
SET @MessageID = NewId();

WITH XMLNAMESPACES (DEFAULT 'http://
www.sdbgroep.nl/MPlus/Rekenserver/2008'),
SELECT @Vraag = (
  SELECT @MessageID '@Id', CURRENT_
  TIMESTAMP '@Timestamp',
  (
    SELECT DienstverbandID '@Id',
    (
      SELECT CaoCodeID '@Id'
      , CaoCodeCode 'Code'
      , CaoCodeIs4Weken 'IsVierWeken'
      , CaoCodeNaam 'Naam'
      , CONVERT(int, CaoCodeWerkweek-
      Uren) 'WerkweekUren'
      FROM dbo.tblCaoCode
      WHERE CaoCodeID = dv.fkCaoCodeID
      FOR XML PATH ('Cao'), TYPE
    )
    , dv.DienstverbandDagenPerWeek
    'DagenPerWeek'
    , CONVERT(char(10), dv.Dienstverband-
    DatumInDienst, 120) 'DatumInDienst'
    , CONVERT(char(10), dv.Dienstverband-
    DatumUitDienst, 120) 'DatumUitDienst'
    , ISNULL(NULLIF(dv.DienstverbandPart-
    timeUren,0), cao.CAOcodeWerkweekUren)
    'ParttimeUren'
    , dv.DienstverbandGarantieToeslag
    'Salaris/@GarantieToeslag'
    , gb.CAOInpassingCodeBedrag 'Salaris/
    @GarantieTredeBedrag'
    , dv.DienstverbandLoonToeslag
    'Salaris/@LoonToeslag'
    , dv.DienstverbandSalaris 'Salaris/
    NominaalSalaris'
    FROM Ö
    WHERE Ö
    FOR XML PATH ('Dienstverband'), TYPE
  ),
  Ö
```

CODEVOORBEELD 10: XML GENEREREN MET SELECT FOR XML PATH

Codevoorbeeld 11 toont een bijpassend XML-fragment.

```
<LoonfactorBerekeningVraag xmlns="http://
www.sdbgroep.nl/MPlus/Rekenserver/2008"
Id="300B830A-EA57-DD11-82FF-000C294CDAD3"
Timestamp="2008-07-22T14:31:05.487">
  <Dienstverband xmlns="http://
  www.sdbgroep.nl/MPlus/Rekenserver/2008"
  Id="3062">
    <Cao xmlns="http://www.sdbgroep.nl/
    MPlus/Rekenserver/2008" Id="100">
      <Code>4</Code>
      <IsVierWeken>0</IsVierWeken>
      <Naam>CAO Ziekenhuizen</Naam>
      <WerkweekUren>36</WerkweekUren>
    </Cao>
    <DagenPerWeek>4.00</DagenPerWeek>
    <ParttimeUren>23.00</ParttimeUren>
    <Salaris GarantieToeslag="121.00"
    LoonToeslag="0.00">
      <NominaalSalaris>1234.5600</
      NominaalSalaris>
    </Salaris>
    <Soort>PT</Soort>
  </Dienstverband>
</LoonfactorBerekeningVraag>
```

CODEVOORBEELD 11: XML NAAR AANLEIDING VAN DE SELECT UIT CODEVOORBEELD 10

De gegenereerde XML wordt vervolgens naar de Rekenservice gestuurd en die komt uiteindelijk met een XML-antwoord. Dat XML-antwoord kan vrij eenvoudig weer tot een resultset worden omgevormd met behulp van de ingebouwde functies op het XML-datatype. Zie Codevoorbeeld 12. Met de nodes() functie wordt een XML-nodelist gemaakt. Vervolgens worden de waarden uit elke node geselecteerd met de value() functie. De resultset kan tenslotte gewoon worden gejoined aan andere tabellen.

```
DECLARE @Antwoord xml;
SET @Antwoord = '
<LoonfactorBerekeningAntwoord xmlns=
"http://www.sdbgroep.nl/MPlus/Reken-
server/2008" Id="300B830A-EA57-DD11-82FF-
000C294CDAD3" Timestamp="2008-07-
22T14:31:05.487">
  <VariabeleMutatie>
    <DienstverbandId>5031</DienstverbandId>
    <KaartSoort>5</KaartSoort>
    <LoonfactorCode>093</LoonfactorCode>
    <PeriodeId>463</PeriodeId>
    <Waarde>4</Waarde>
  </VariabeleMutatie>
  <VariabeleMutatie>
    <DienstverbandId>5031</DienstverbandId>
    <KaartSoort>5</KaartSoort>
    <LoonfactorCode>094</LoonfactorCode>
    <PeriodeId>463</PeriodeId>
    <Waarde>1</Waarde>
  </VariabeleMutatie>
</LoonfactorBerekeningAntwoord>';

WITH XMLNAMESPACES ('http://www.sdbgroep.nl/
MPlus/Rekenserver/2008' as rs)
SELECT
  vm.KaartSoort
, vm.Waarde
, vm.DienstverbandId
, lf.LoonfactorID
, vm.KostenPlaatsId
, vm.KostenSoortId
, vm.PeriodeId
FROM
(
  SELECT
    vm.value('(rs:DienstverbandId)[1]',
    'int') as DienstverbandId
  , vm.value('(rs:KaartSoort)[1]', 'int')
  as KaartSoort
  , vm.value('(rs:KostenSoortId)[1]',
  'int') as KostenSoortId
  , vm.value('(rs:KostenPlaatsId)[1]',
  'int') as KostenPlaatsId
  , vm.value('(rs:LoonfactorCode)[1]',
  'varchar(3)') as LoonfactorCode
  , vm.value('(rs:PeriodeId)[1]', 'int')
  as PeriodeID
  , vm.value('(rs:Waarde)[1]', 'numeric
  (7,2)') as Waarde
  FROM @Antwoord.nodes('/rs:Loonfactor-
  BerekeningAntwoord/rs:Variabele-
  Mutatie') VariabeleMutaties(vm)
) as vm
INNER JOIN dbo.Loonfactor lf ON lf.Loon-
factorCode = vm.LoonfactorCode
```

CODEVOORBEELD 12: HET OMZETTEN VAN EEN XML FRAGMENT TOT EEN RELATIONELE RESULTSET

Conclusie

In dit artikel hebben we laten zien dat SQLServer 2005 of 2008 een uitstekend platform is om services te bouwen die dicht tegen de database aanliggen. Service Broker heeft geen op webservice standaard gebaseerde interfaces zoals WCF, dus is niet geschikt als publieke service. Maar de transactionele, betrouwbare en zeer snelle services die gemaakt kunnen worden met Service Broker zijn uitstekend geschikt als service achter de schermen. De SQLCLR maakt het mogelijk om die services gewoon met het vertrouwde en krachtige .Net framework te realiseren. De nieuwste T-SQL mogelijkheden sluiten daar naadloos op aan.

Links

XsdObjectGen: hulpmiddel om op basis van een XML Schema een C# object model te genereren. Te downloaden via: http://www.microsoft.com/downloads/details.aspx?FamilyID=89E6B1E5-F66C-4A4D-933B-46222BB01EB0&displaylang=en

XgenPlus: hulpmiddel om XML Serializers te genereren op basis van een .Net object model. Te downloaden via: http://www.codeplex.com/xgenplus

How To: Activate Service Broker networking: http://msdn.microsoft.com/en-us/library/ms166113.aspx

Frans van der Geer, is werkzaam als software architect bij VX Company in Baarn en kan worden bereikt via e-mail op fvdgeer@vxcompany.com.

André van Leeuwen, is werkzaam als software engineer bij SDB Groep te Leidschendam en kan worden bereikt via e-mail op a.vanleeuwen@sdbgroep.nl.


Your potential. Our passion.®
Microsoft®
LET WEL: GEBRUIK DEZE MAGISCHE
KRACHTEN ALLEEN VOOR HET GOEDE.