

Einer für alle, alle für einen – verteilte Transaktionen



Mithilfe des Compensating Resource Manager (CRM) kann innerhalb einer Transaktion auf das Dateisystem zugegriffen werden. dotnetpro zeigt, wie sich dadurch Inkonsistenzen im Datensystem vermeiden lassen, und zwar nicht nur in der Microsoft-Welt.

___ Ein Ressourcen-Manager kann den Zugriff auf die von ihm verwalteten Daten über Transaktionen steuern. Nicht für alle Datenbanken existieren Ressourcen-Manager. Wird auf solche Daten innerhalb einer Transaktion zugegriffen, so bleiben Änderungen von einem Commit oder Abort der Transaktion unbeeinflusst. Stellen Sie sich folgende Situation vor: Eine Bestellung wird in eine Datenbank eingetragen und eine Datei auf der Festplatte mit den Bestelldetails erzeugt. Jetzt wird die Transaktion abgebrochen. Der Eintrag aus der Datenbank wird dadurch automatisch vom Ressourcen-Manager der Datenbank zurückgesetzt, aber die Datei auf der Festplatte bleibt bestehen; sie müsste manuell gelöscht werden.

Compensating Resource Manager (CRM)

Für den Zugriff auf ein solches Datensystem könnte ein eigener Ressourcen-Manager implementiert werden, dessen Programmierung aber sehr aufwändig wäre. Als Alternative dazu bietet sich ein Compensating Resource Manager an. Er ist in der Lage, im Falle des Abbruchs einer Transaktion die Änderungen der Transaktion an den Daten rückgängig zu

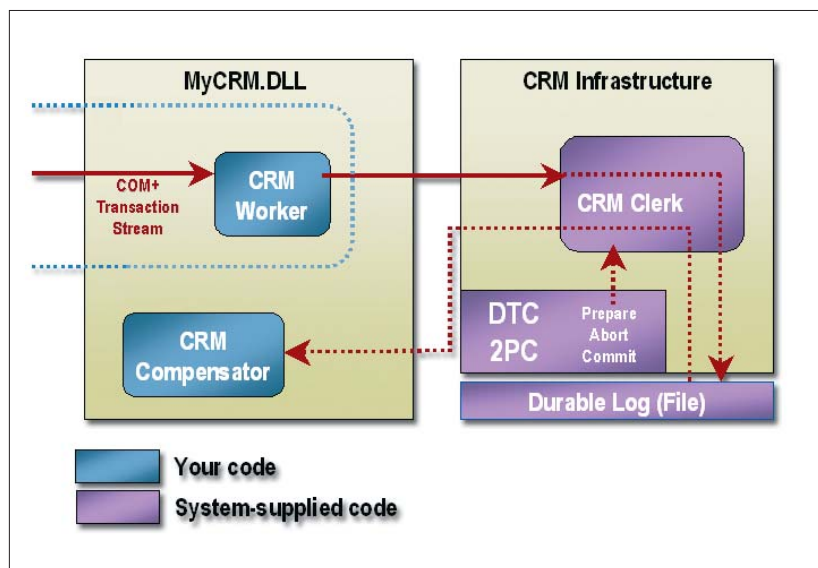


Abbildung 1 | Die Infrastruktur eines CRM.

machen. Dazu führt er kompensierende Aktionen innerhalb der gleichen Transaktion während des Abbruchs durch. Im oben genannten Beispiel löscht er die erzeugte Datei, wenn die Transaktion abgebrochen wird. Das Löschen ist die kompensierende Aktion zum Erzeugen der Datei.

Es ist nicht immer möglich, alle Aktionen rückgängig zu machen. Wird im Rahmen einer Transaktion eine E-Mail versendet, lässt sich dieser Vorgang nicht nachträglich rückgängig machen, aber es kann als Kompensation eine zweite Mail gesendet werden, die den Empfänger auffordert, die erste Mail zu ignorieren.

Die CRM-Infrastruktur

COM+ stellt eine Infrastruktur zur Verfügung, die es Komponenten ermöglicht, Aktionen und kompensierende Aktionen durchzuführen, so als wären sie echte Ressourcen-Manager. Dabei unterstützt CRM nur zwei der vier ACID-Merkmale einer klassischen Transaktion: Atomizität und Dauerhaftigkeit. Um Konsistenz und Isolation muss sich der Entwickler der CRM-Komponente selber kümmern. Im Falle eines unerwarteten Systemabsturzes stellt die CRM-Infrastruktur einen Wiederherstellungsmechanismus (Recovery) zur Verfügung, der mithilfe einer Log-Datei versucht, einen konsistenten Zustand zu erreichen.

SUMMARY

Auf einen Blick

Die Serie gibt in drei Teilen eine umfassende praktische Einführung in die transaktionale Datenverarbeitung. Dieser Artikel behandelt den Compensating Resource Manager sowie die Zusammenarbeit von COM+-Transaktionen und Oracle-Datenbanken.

Eingesetzte Anwendungen

.NET Framework, Windows 2000 oder Windows XP

CD-Code

DrillDown06

Serie

Teil 3/3

Autor

Marcel Gnoth ist Senior Consultant bei der Berliner NTeam GmbH. Seine Arbeitsschwerpunkte liegen im Bereich COM, .NET und verteilte Informationssysteme, zu denen er auch Trainings durchführt. Sie erreichen ihn unter marcel@gnoth.net.

Abbildung 1 zeigt die Infrastruktur des CRM. Für die Entwicklung des CRM Workers und des CRM Compensators sind Sie zuständig. Beide Komponenten können in der gleichen DLL (Assembly) realisiert werden. Der Worker arbeitet innerhalb einer COM+-Transaktion, der Compensator wird während des 2-Phasen-Commit außerhalb des Transaktionsstroms aufgerufen. Beide können die dauerhafte oder beständige Logdatei verwenden, die von der CRM-Infrastruktur zur Verfügung gestellt wird. Nach einem Systemabsturz dient die Logdatei dazu, um mit dem CRM Compensator einen konsistenten Systemzustand wieder herzustellen.

Der CRM Worker erzeugt eine neue Instanz eines CRM Clerk und übergibt dabei den Typ des CRM Compensators. Der Compensator überschreibt einige Methoden seiner Basisklasse, meist *PrepareRecord*, *CommitRecord* und *AbortRecord*.

Der CRM Clerk wird von den .NET Enterprise Services implementiert. Jeder Clerk verwaltet ein zusammengehörendes Paar von Worker und Compensator. Der Clerk leitet Prepare-, Abort- und Commit-Nachrichten des DTC an den dazugehörigen Compensator weiter veranlasst und Einträge in die Log-Datei.

Ablauf einer CRM-Transaktion

Eine physische DTC-Transaktion besteht aus zwei Elementen, einer COM+-Transaktion und dem anschließenden 2-Phasen-Commit-Prozess.

In einer existierenden COM+-Transaktion wird neben anderen transaktionalen Komponenten, die auf Datenbanken oder andere Ressourcen-Manager zugreifen, ein CRM Worker erzeugt. Der Worker wiederum instanziiert einen CRM Clerk und regis-

Listing 1 Der CRM Worker

```
<Transaction(Option.Required)> _
Public Class CRMWorker
    Inherits ServicedComponent
    <AutoComplete()> _
    Public Sub PostOrder(ByVal order As OrderInformation)
        Dim orderRec As OrderRecord = New OrderRecord()
        orderRec.tempPath = "C:\CRMDemo\Temp"
        orderRec.fileName = String.Format("{0}-order.xml", order.orderID)

        ' Create clerk object, registering the compensator
        Dim c As Clerk = Nothing
        c = New Clerk(GetType(CRMCompensator), "CRMCompensator", _
            CompensatorOptions.AllPhases)
        If (c Is Nothing) Then
            Throw New Exception("Could not create CRM Clerk")
        End If

        ' Use the clerk to record the given filename to the log
        c.WriteLogRecord(orderRec)
        c.ForceLog()

        ' Create a file with the given filename and write out a message
        Dim fs As FileStream = New FileStream(Path _
            .Combine(orderRec.tempPath, orderRec.fileName), _
            FileMode.Create, FileAccess.Write, FileShare.None)
        Dim formatter As IFormatter = CType(New SoapFormatter(), IFormatter)

        ' Serialize the object graph to stream
        formatter.Serialize(fs, order)
        fs.Close()
    End Sub
End Class
```

Die Beispielprojekte auf der Heft-CD

Das Beispiel zeigt den Zugriff auf das Dateisystem. Das Szenario der Applikation besteht darin, dass während einer COM+-Transaktion in einem Temp-Verzeichnis eine Datei erzeugt wird, die beim Commit in ein anderes Verzeichnis verschoben oder bei einem Abort gelöscht wird.

Auf der Heft-CD finden Sie vier Projekte. CRMClient ist eine Konsolenanwendung, welche die Transaktion startet. Watcher ist eine Windows-Anwendung, die zwei Verzeichnisse überwacht und Änderungen an den Verzeichnissen anzeigt. Starten Sie diese Anwendung direkt und nicht über den Debugger von Visual Studio.

Das Projekt TransactionWrapper enthält eine Klasse, die eine neue COM+-Transaktion benötigt. Innerhalb dieser Transaktion erstellt die Komponente eine Instanz von CRMDemo.CRMWorker und ruft die PostOrder-Methode auf, welche eine Datei erzeugt. Anschließend ruft die Wrapper-Klasse SetComplete oder SetAbort auf, um die COM+-Transaktion zu testen.

Das CRMDemo-Projekt enthält den CRM Worker und den CRM Compensator.

triert seinen Compensator beim CRM Clerk. Anschließend kann er Aktionen innerhalb der laufenden COM+-Transaktion durchführen. Wichtig ist, dass er vor jeder Aktion einen Eintrag in die Logdatei schreibt. Nur so kann nach einem Absturz mithilfe der Logdatei ein konsistenter Zustand wiederhergestellt werden..

Um die Transaktion abzuschließen, wird als Erstes die Prepare-Phase eingeleitet. Der DTC informiert den Clerk, dass die Prepare-Phase beginnt, dieser informiert den Compensator. Der Clerk liest nun die Logdatei und liefert die Datensätze an den Compensator, der zusätzliche Einträge in die Logdatei schreiben kann. Am Ende der Prepare-Phase könnte der Compensator dem Clerk mitteilen, dass das Prepare nicht erfolgreich war, was einen Abbruch der Transaktion zur Folge hätte. Auch für die Änderungen an Daten anderer Ressourcen-Manager würde beim Abbruch ein Rollback durchgeführt werden.

In der zweiten Phase wird entweder ein Commit oder ein Abort vorgenommen. Bei einem Abort werden die Datensätze der Logdatei in umgekehrter Reihenfolge an den Compensator geliefert. Der Compensator kann jetzt die Aktionen durchführen, um die Änderungen zu etablieren, oder kompensierende Aktionen durchführen, um die Änderungen rückgängig zu machen.

Der Worker

Für das Assembly müssen einige Attribute gesetzt werden, damit COM+-Transaktionen und CRM unterstützt werden, wichtig ist das Attribut *ApplicationCrmEnabled*. Eine COM+-Applikation kann nachträglich im MMC-SnapIn für Komponentendienste als CRM-fähig konfiguriert werden.

```
Imports System.EnterpriseServices.CompensatingResourceManager
<Assembly: ApplicationActivation(ActivationOption.Server)>
<Assembly: ApplicationCrmEnabled()>
```

In Listing 1 sehen Sie eine vereinfachte Implementierung eines CRM Workers. Auf der Heft-CD finden Sie ein umfangreicheres Beispiel.

Da das Transaction-Attribut der Klasse auf *Required* gesetzt ist, arbeitet diese Klasse immer innerhalb einer COM+-Transaktion. Die einzige Methode der Klasse instanziiert einen CRM Clerk und übergibt ihm den Typ des dazugehörigen Kompensators.

Bevor die Methode die Textdatei erzeugt, schreibt sie diesen Schritt in die Logdatei und ruft die *Clerk.ForceLog*-Methode auf, um sicherzugehen, dass der Logeintrag dauerhaft gesichert ist. Danach wird das *OrderInformation*-Objekt serialisiert und in die neue Textdatei geschrieben. Diese Datei wird in einem temporären Ordner erstellt, da die Transaktion noch nicht abgeschlossen ist und andere Anwendungen diese Datei noch nicht sehen sollen (Isolation).

Der Kompensator

Die Compensator-Klasse überschreibt einige Methoden ihrer Basisklasse. In Listing 2 finden Sie eine einfache Implementierung eines Kompensators. *PrepareRecord* wird während der Prepare-Phase aufgerufen und hat im Beispiel keine besondere Bedeutung. Die Methode *CommitRecord* verschiebt die Datei aus dem temporären Verzeichnis in das Zielverzeichnis, die *AbortRecord*-Methode löscht die Datei aus dem temporären Verzeichnis.

Wichtig ist, dass alle Methoden idempotent sind. Das bedeutet, dass diese Methoden mehrmals aufgerufen werden können, im

Ergebnis die Aktion jedoch nur einmal durchführen. Während des Wiederherstellungsprozesses kann es vorkommen, dass diese Methoden mehrmals aufgerufen werden. Beispiele für idempotente Aktionen sind:

- das Löschen einer Datei. Unabhängig davon, wie oft diese Funktion aufgerufen wird, zählt einzig das Ergebnis (die Datei ist nicht mehr vorhanden).
- das Setzen eines Wertes in der Registry.

Das Erhöhen eines Wertes um 5 ist nicht idempotent, da bei jedem erneuten Aufruf der Wert inkrementiert wird.

Achtung! Der Compensator darf sich nicht darauf verlassen, dass während der Prepare- und der Abort-/Commit-Phase die gleiche Compensator-Instanz verwendet wird. Benötigt der Compensator Informationen aus der Prepare-Phase in der Abort-/Commit-Phase, muss er diese Informationen in die Log-Datei schreiben.

Designüberlegungen

Beachten Sie die Idempotenz der Aktionen, die Sie im Compensator durchführen. Außerdem müssen Sie sich selber um Concurrency-Probleme kümmern. Auf einem System können viele Worker parallel arbeiten. Sie müssen dafür sorgen, dass diese sich nicht gegenseitig behindern. Sie können Ressourcen mit einem Flag versehen, sodass andere Worker erkennen, dass diese Ressource benutzt wird, oder Sie verstecken eine Ressource so lange, bis sie wieder einen konsistenten Zustand hat (temporäres Verzeichnis für eine Datei).

Wird eine COM+-Applikation gestartet, erfolgt eine Kontrolle der Log-Datei daraufhin, ob es nicht abgeschlossene Transaktionen gab. Der DTC wird kontaktiert, um den Ausgang nicht abgeschlossener Transaktionen zu erfahren. Ist der Ausgang bekannt, wird entsprechend der Compensator aufgerufen. Wenn nicht, bleiben die Transaktionen *InDoubt* und müssen manuell gesetzt werden.

Sind an einer Transaktion mehrere DTC beteiligt und ein DTC verliert den Kontakt zum koordinierenden DTC, bleibt die Transaktion im Zustand *InDoubt*.

Unter [1] und in der MSDN finden Sie weitere Informationen zu diesem Thema.

Oracle und der DTC

Zum Zugriff auf andere DBMS sind keine eigenen Ressourcen-Manager erforderlich. Die Zusammenarbeit des Microsoft Distributed Transaction Coordinator mit Ressourcen-Managern, die nicht aus dem Hause Microsoft kommen, ist nicht trivial.

Sowohl Microsoft als auch Oracle erklären, dass die Zusammenarbeit zwischen Oracle-Datenbanken und dem Distributed Transaction Coordinator (COM+) möglich ist. Beide Firmen zeigen unterschiedliche Wege auf und beide sind nicht ganz leicht.

Der Weg Microsofts

In [2] und [3] beschreibt Microsoft, wie COM+ und Oracle eine Transaktion durchführen können. Die Oracle Services for MTS werden laut Microsoft nicht benötigt. Die Beschreibung erklärt ausführlich, welche Versionen des Servers und des Oracle-Clients notwendig sind und welche Änderungen an der Konfiguration

Listing 2 Der Compensator.

```
Public Class CRMCompensator
    Inherits Compensator
    Private _fileName As [String]

    Public Overrides Function PrepareRecord(ByVal rec As LogRecord) As Boolean
        MessageBox.Show("PrepareRecord with object of type " + _
            rec.Record.GetType().ToString(), "Phase 1")
        Dim o As [Object] = rec.Record
        _fileName = o.ToString()
        Return False
    End Function

    Public Overrides Function CommitRecord(ByVal rec As LogRecord) As Boolean
        Dim orderRec As OrderRecord
        MessageBox.Show("CommitRecord", "Phase 2")
        orderRec = CType(rec.Record, OrderRecord)
        Dim strPathFinal As String = "C:\CRMDemo\Final"

        'Check to see if file already exists in Final folder
        If File.Exists(Path.Combine(strPathFinal, orderRec.fileName)) Then
            'Since file already exists, just delete temp
            File.Delete(Path.Combine(orderRec.tempPath, orderRec.fileName))
        Else
            File.Move(Path.Combine(orderRec.tempPath, orderRec._
                fileName), Path.Combine(strPathFinal, orderRec.fileName))
        End If
        Return True
    End Function

    Public Overrides Function AbortRecord(ByVal rec As LogRecord) As Boolean
        MessageBox.Show("AbortRecord", "Phase 2")

        Dim orderRec As OrderRecord
        orderRec = CType(rec.Record, OrderRecord)
        File.Delete(Path.Combine(orderRec.tempPath, orderRec.fileName))
        Return True
    End Function
End Class
```

vorgenommen werden müssen, bis hin zu den Schlüsseln in der Registry. Es stellt sich jedoch die Frage, ob die Vielzahl von Einstellungen nach dem nächsten Oracle-Patch immer noch funktionieren.

Der Weg Oracles

Oracle hat für den DTC und COM+ die Oracle Services for MTS entwickelt, die auf dem Client installiert sein müssen. Der Server muss nicht unter Windows laufen. Seit einigen Monaten gibt es einen eigenen Treiber von Oracle für .NET, der aber nicht mit dem Microsoft Provider für Oracle in Zusammenhang steht. [4], [5] und [6] beschreiben den Oracle-Weg. Es müssen eine Reihe von Oracle-Komponenten sowohl auf dem Server als auch auf dem Client installiert sein, aber insgesamt erscheint der Weg zuverlässiger, da davon auszugehen ist, dass ihn Oracle bei seinen Updates berücksichtigen wird.

Welcher Weg ist nun der beste?

Eine ausführliche Untersuchung, welcher Weg der beste ist, um mit einer Oracle-Datenbank zusammenzuarbeiten, bleibt einem zukünftigen Artikel vorbehalten. Es gibt eine ganze Reihe von Datenzugriffsbibliotheken, um von .NET aus auf Oracle zuzugreifen. Mir erscheinen die Provider für .NET als die besten, da sie speziell auf Oracle ausgerichtet sind. Wie stark die Unterschiede zwischen dem Provider von Microsoft und dem von Oracle sind, wird die Zukunft zeigen.

Abschließend sei noch auf einen interessanten Artikel aus der MSDN verwiesen, in dem die Portierung einer Datenzugriffsschicht nach Oracle beschrieben wird [7].

Andere Welten

Wenn Sie im Internet nach Informationen suchen, wundern Sie sich nicht über die verschiedenen Bezeichnungen für Transaktionstechnologien von Microsoft. Der Software-Riese macht es uns mit seinen Umbenennungen nicht leicht. Der MTS ist inzwischen in COM+ aufgegangen und der DTC ermöglicht die Zusammenarbeit mit mehreren MTS-/ COM+-Anwendungen, die auf

verschiedenen Rechnern installiert sind. Laut Microsoft kann der DTC an folgenden Transaktionen teilnehmen:

- X/Open DTP (X/Open Distributed Transaction Processing)
- XA-konforme Transaktionsverarbeitungsmonitore
- Encina
- TopEnd
- Tuxedo
- IBM DB/2
- Transaction Internet Protocol (TIP)

In drei Artikeln aus der Microsoft Knowledge Base wird eine Zusammenarbeit mit dem DB2-System von IBM beschrieben: K197208, K218590, K216428.

Aber wie auch immer: Wenn wir Entwickler Systeme verschiedener Hersteller zur Kooperation bewegen müssen, befinden wir uns nicht auf einer ebenen grünen Wiese, sondern auf einem Stück Erde, durchsetzt mit majestätischen Maulwurfshügeln und einigen Büscheln Gras ☺. |||||

-
- [1] Microsoft Official Curriculum, Course 2557A: Building COM+ Applications Using Microsoft® .NET Enterprise Services
 - [2] Using Oracle with Microsoft Transaction Server and COM+: <http://support.microsoft.com/support/complus/mtsandoracle.asp>
 - [3] Best Practices for Using Oracle and COM+: http://msdn.microsoft.com/library/en-us/dncomser/html/complus_best.asp
 - [4] Handling Distributed Transactions using MS Transaction Server through Oracle Data Provider for .Net (ODP.Net): http://otn.oracle.com/sample_code/tech/windows/odpnet/DistributedTransactionSample/Readme.html
 - [5] Oracle Services for Microsoft Transaction Server: http://otn.oracle.com/tech/windows/ora_mts/content.html
 - [6] Oracle Data Provider for .NET <http://otn.oracle.com/tech/windows/odpnet/content.html>
 - [7] Fitch & Mather Stocks: Data Access Layer for Oracle 8 http://msdn.microsoft.com/library/en-us/dnfmstock/html/fmstocks_da_orcl.asp
 - [8] MSDN, Microsoft SQL Server, Technical Articles: "An Overview of Transaction Processing Concepts and the MS DTC"
 - [9] „Distributed Transactions: What you need to know to stay out of trouble?“ http://otn.oracle.com/tech/java/architect/distributed_transactions.html

Installation

Die DLLs der beiden zuletzt genannten Projekte TransactionWrapper und CRMDemo müssen sowohl im Global Assembly Cache als auch im COM+-Katalog registriert werden. Dazu starten Sie die Visual-Studio-Befehlszeile und rufen die beiden Tools gacutil und regsvcs für beide DLLs auf:

```
gacutil -i c:\.....\TransactionWrapper.dll
regsvcs c:\.....\TransactionWrapper.dll
gacutil -i c:\.....\CRMDemo.dll
regsvcs c:\.....\CRMDemo.dll
```

Jetzt starten Sie die kompilierte Exe des Watcher-Projekts, setzen CRMClient als Startprojekt und starten den Debugger von Visual Studio. Möchten Sie auch die beiden COM+-DLLs debuggen, müssen Sie mit Menü/_Debug/_Processes den DLL-Host-Prozess in dem die COM+-Anwendung ausgeführt wird, manuell zum Debugger hinzufügen. Dann hält Visual Studio auch an den Haltepunkten in diesen DLLs an.