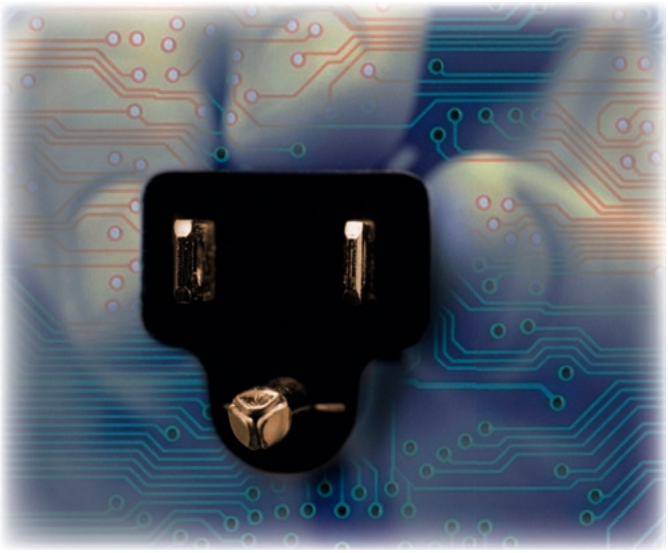
Die polyglotte MOM

Integration mit Apache ActiveMQ

Klaus Rohe

Apache ActiveMQ ist ein zu JMS 1.1 kompatibler Message-Broker, welcher Schnittstellen zu unterschiedlichen Anwendungsentwicklungsplattformen und Programmiersprachen bietet, wie z. B. zu Microsoft .NET (C#, VB.NET, …), C/C++, Python und Ruby. ActiveMQ eignet sich damit sehr gut für die asynchrone, nachrichtenorientierte Integration von Programmen, die in einer der oben genannten Programmiersprachen entwickelt wurden. In diesem Beitrag wird beschrieben, wie man ActiveMQ aus verschiedenen Programmiersprachen nutzen kann, insbesondere wie man damit die asynchrone Kommunikation zwischen Microsoft .NET-Applikationen und solchen auf der Java-Plattform realisieren kann.



Apache ActiveMQ im Überblick

Apache ActiveMQ ist eine nachrichtenorientierte Middleware (MOM), welche die komplette JMS 1.1-Spezifikation implementiert. ActiveMQ ist nicht nur auf der Java-Plattform nutzbar, sondern unterstützt eine große Anzahl von „Cross Language Clients“, dazu gehören die Programmiersprachen, die unter Microsoft .NET verfügbar sind, C/C++, aber auch Skriptsprachen wie Python, Ruby, Perl und PHP. Eine komplette Liste der Sprachen, mit denen man auf ActiveMQ zugreifen kann, ist unter dem Link in [ActiveMQ] zu finden. Von dort kann auch die aktuelle, freigegebene Version heruntergeladen werden.

Aufgrund der Vielzahl von Sprachen und Applikationsentwicklungsplattformen ist ActiveMQ sehr gut für die asynchrone, nachrichtenorientierte Integration in einer heterogenen Applikationslandschaft geeignet.

Installation und Start

Im Folgenden wird die Installation der Binärdistribution unter Microsoft Windows beschrieben. Die zum Zeitpunkt des Verfassens dieses Artikels freigegebene und aktuelle Version ist ActiveMQ 5.3.0.

Für die Plattform Microsoft Windows muss man die Datei `apache-activemq-5.3.0-bin.zip` herunterladen und in ein entsprechendes Verzeichnis, z. B. `D:\apache-activemq-5.3.0`, auspacken. Die resultierende Verzeichnisstruktur der ActiveMQ-Installation sieht wie folgt aus (Ausgabe des PowerShell-Kommandos ls):

```
PS D:\> ls apache-activemq-5.3.0

    Directory: D:\apache-activemq-5.3.0

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        21.10.2009     16:05            bin
d----        21.10.2009     16:06            conf
d----        14.01.2010     14:59            data
d----        21.10.2009     16:06            docs
d----        21.10.2009     16:06            example
d----        21.10.2009     16:06            lib
d----        21.10.2009     16:05            webapps
-a---        21.10.2009     16:05    3390161 activemq-all-5.3.0.jar
-a---        21.10.2009     16:05      41200 LICENSE
-a---        21.10.2009     16:05       3420 NOTICE
-a---        21.10.2009     16:05       2646 README.txt
-a---        21.10.2009     16:05       2790 user-guide.html
-a---        21.10.2009     16:05       2080 WebConsole-README.txt
```

Damit ActiveMQ funktioniert, muss mindestens die Java-Laufzeitumgebung ab Version 1.4.x installiert sein. Will man Java-Anwendungen mit ActiveMQ entwickeln, benötigt man mindestens das entsprechende JDK der Java Standard Edition. Im Verzeichnis `D:\apache-activemq-5.3.0\bin` befindet sich die Batchdatei `activemq.bat`, welche man zum Starten des ActiveMQ-Servers ausführen muss. Wichtig ist, dass vor dem Ausführen dieser Batchdatei die Umgebungsvariable `JAVA_HOME` den korrekten Wert hat.

Im Verzeichnis `D:\apache-activemq-5.3.0\bin\win32` befinden sich zwei Batchdateien, die es gestatten, den ActiveMQ-Server als NT-Service zu installieren bzw. zu deinstallieren. Ist der ActiveMQ-Server erfolgreich gestartet, erhält man unter der URL `http://localhost:8161/admin/` eine Web-Applikation mit Namen „ActiveMQ Console“ zur einfachen Administration und Überwachung des ActiveMQ-Servers. Mit ihr kann man auch Nachrichten an einzelne Queues bzw. Topics senden. Man hat damit eine einfache Möglichkeit, ActiveMQ-Consumer bzw. -Subscriber zu testen.

Mechanismen für die Nachrichtenspeicherung

Die Mechanismen, die ActiveMQ zur persistenten Speicherung von Nachrichten benutzt, sind konfigurierbar. Der Speichermechanismus wird durch entsprechende Einträge in der ActiveMQ-Konfigurationsdatei `activemq.xml` konfiguriert, die sich im Verzeichnis `D:\apache-activemq-5.3.0\conf` befindet.

Die Standardeinstellung ist die Speicherung der Nachrichten im Dateisystem, und zwar in Dateien, die sich in Unterverzeichnissen des Verzeichnisses `conf` der ActiveMQ-Installation befinden (siehe oben).

Eine weitere Option ist die Speicherung der Nachrichten in einem relationalen Datenbanksystem, welches JDBC unterstützt, wie z. B. Microsoft SQL Server, Oracle, PostgreSQL und Apache Derby. Die Speicherung von Nachrichten in einem relationalen Datenbanksystem ist dann angebracht, wenn es um hohe Verfügbarkeit geht und wenn eine Datenbankadministration, die für entsprechende Sicherungsmechanismen sorgt, schon existiert sowie Performanz nicht die höchste Priorität hat.

Als dritten Nachrichtenspeichermechanismus bietet ActiveMQ die Speicherung von Nachrichten im Hauptspeicher an. Dieser Mechanismus bietet optimale Performanz, ist allerdings mit der Gefahr von Nachrichtenverlusten verbunden, wenn der Rechner, auf dem der ActiveMQ-Server läuft, ausfällt.

Weitere sehr detaillierte Informationen zu den ActiveMQ-Speichermechanismen sind im Kapitel 5, „Message Persistence“, von [SnDaBo09] zu finden.

Unterstützte Netzwerk- und Datenübertragungsprotokolle

ActiveMQ unterstützt die Netzwerkprotokolle TCP (mit SSL), UDP und HTTP/HTTPS. Das native Datenübertragungsprotokoll (Wire Protocol*) von ActiveMQ ist OpenWire. OpenWire wird für ActiveMQ-Clients genutzt, welche mit Java, einer der .NET-Programmiersprachen (C#, VB.NET, …) oder C/C++ implementiert werden sollen. OpenWire [OpenWire] wurde mit dem Ziel entworfen, maximale Performanz und hohen Durchsatz zu garantieren sowie eine umfangreiche Funktionalität zu gewährleisten.

Als weiteres Datenübertragungsprotokoll wird STOMP unterstützt. STOMP ist ein Akronym und bedeutet *S*treaming *T*ext *O*riented *M*essaging *P*rotocol. STOMP wurde unter dem Gesichtspunkt entworfen, dass es einfach zu implementieren ist und damit von einer großen Zahl von Programmier- und Skriptsprachen unterstützt werden kann. Aktuell gibt es Implementierungen für die Skriptsprachen Python, Ruby, Perl und PHP.

Die Netzwerk- und Datenübertragungsprotokolle werden in der ActiveMQ-Konfigurationsdatei **activemq.xml** definiert. Der Eintrag für OpenWire und STOMP sieht wie folgt aus:

```
<transportConnectors>
  <transportConnector name="openwire" uri="tcp://0.0.0.0:61616"/>
  <transportConnector name="stomp" uri="stomp://0.0.0.0:61613"/>
</transportConnectors>
```

Der Eintrag für „OpenWire“ ist standardmäßig vorhanden, alle anderen müssen gesondert eingetragen werden. Wie in Abbildung 1 dargestellt, können Clients, die „OpenWire“ nutzen, mit solchen kommunizieren, die STOMP als Datenübertragungsprotokoll verwenden. Das gilt sowohl für die Punkt-zu-Punkt-Kommunikation als auch für das Publish/Subscriber-Modell.

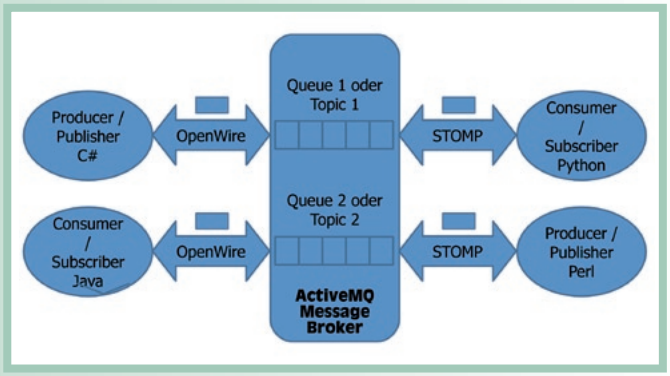


Abb. 1: Kommunikationsoptionen mit ActiveMQ

.Net Message Service API (NMS)

Die NMS-Programmierschnittstelle definiert eine .NET-Schnittstelle für Message-Queue-Systeme. Für ActiveMQ gibt es eine auf OpenWire und C# basierende Implementierung der NMS-Programmierschnittstelle. Es existiert nicht nur die Implementierung für ActiveMQ sondern auch für die Microsoft Message Queue (MSMQ) und TIBCO Enterprise Message Service (EMS). Die aktuelle Version von NMS für ActiveMQ ist 1.1.0. Unter [NMS] gibt es eine binäre Version sowie den Quellcode, der als Visual Studio 2008 Project vorliegt.

Die binäre Version besteht aus den beiden ZIP-Dateien **Apache.NMS-1.1.0-bin.zip** und **Apache.NMS.ActiveMQ-1.1.0-bin.zip**, welche die .NET-Assemblys **Apache.NMS.dll** und **Apache.NMS.ActiveMQ.dll** enthalten. Diese müssen beim Linken eines .NET-ActiveMQ-Clients eingebunden werden bzw. in einem Visual-Studio-Projekt den Referenzen hinzugefügt werden. Für die .NET-Versionen 1.1, 2.0, 3.5 und das .NET Compact Framework 2.0 sowie Mono sind jeweils eigene Implementierungen vorhanden.

```
import javax.jms.*;
import org.apache.activemq.ActiveMQConnectionFactory;

public class VerySimpleAMQConsumer {
  public static void main(String args[]) {
    try {
      ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory(
                                        "tcp://localhost:61616");
      Connection connection = cf.createConnection();
      connection.start();
      Session session = connection.createSession(false,
                          Session.AUTO_ACKNOWLEDGEMENT);
      Destination destination = session.createQueue("TestQ");
      MessageConsumer msgConsumer =
        session.createConsumer(destination);
      System.out.println(
        "SimpleAMQConsumer connected to ActiveMQ queue 'TestQ'.");
      System.out.println(
        "Waiting for Messages in JMS Queue 'TestQ'.\n");
      TextMessage msg = (TextMessage)msgConsumer.receive();
      System.out.println("Message received:     " + msg.getText());
      msgConsumer.close();
      session.close();
      connection.close();
      System.out.print("\nPress <Enter> to exit.");
      System.in.read();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Listing 1: Java-Implementierung eines ActiveMQ-Consumers

```
using System;
using Apache.NMS;
using Apache.NMS.Util;
using Apache.NMS.ActiveMQ;
using Apache.NMS.ActiveMQ.Commands;

class VerySimpleAMQConsumer {
  static void Main(string args[]) {
    try {
      ConnectionFactory cf = new ConnectionFactory(
                               "tcp://localhost:61616");
      Connection connection = (Connection)cf.CreateConnection();
      connection.Start();
      Session session = (Session)connection.CreateSession(
                          AcknowledgementMode.AutoAcknowledge);
      IDestination destination = session.GetQueue("TestQ");
      MessageConsumer msgConsumer =
       (MessageConsumer)session.CreateConsumer(destination);
      Console.WriteLine(
      "VerySimpleAMQConsumer connected to ActiveMQ queue 'TestQ'.");
      Console.WriteLine(
        "Waiting for Messages in JMS Queue 'TestQ'.\n");
      ActiveMQTextMessage msg =
       (ActiveMQTextMessage)msgConsumer.Receive();
```

➔

* Die Bezeichnung „Wire Protocol“ bezieht sich auf das Format, wie Daten zwischen Applikationen ausgetauscht werden (siehe **http://encyclopedia2.thefreedictionary.com/wire+protocol** und **http://en.wikipedia.org/wiki/Wire_protocol**). Die Bezeichnung ist vor allem bei der direkten Übersetzung ins Deutsche – „Drahtprotokoll“ – irreführend.

```
        Console.WriteLine("Message received:     {0}" + msg.Text);
        msgConsumer.Close();
        session.Close();
        connection.Close();
        ConsoleWriteLine("\nPress <Enter> to exit.");
        Console.ReadKey();
      } catch (Exception e) {
        Console.Error.WriteLine(e.Message);
      }
    }
  }
```

Listing 2: .NET-C#-Implementierung eines ActiveMQ-Consumers

Das Assembly **Apache.NMS.dll** besteht aus den Namensräumen **Apache.NMS** und **Apache.NMS.Util**. Der erste Namensraum enthält die Definition der eigentlichen NMS-Programmierschnitttstelle, d. h. die .NET-Interfaces für Queues, Topics und Messages sowie die Klassen für die Ausnahmebehandlung. Der zweite Namensraum stellt Hilfsklassen bereit, welche das Verwalten von URIs (Universal Resource Identifier), Nachrichten, Queues und Topics unterstützen.

Das Assembly **Apache.NMS.ActiveMQ.dll** stellt die speziellen Implementierungen der Schnittstellen für ActiveMQ zur Verfügung. Es enthält unter anderem die für die Applikationsentwicklung wichtigen Namensräume **Apache.NMS.ActiveMQ** und **Apache.NMS.ActiveMQ.Commands**. Der Namensraum **Apache.NMS.ActiveMQ** enthält die in Abbildung 2 gezeigten Klassen.

Eine für die Anwendungsentwicklung wichtige Klasse ist die **ConnectionFactory** zum Erzeugen einer Instanz der Klasse **Connection**. Diese hat eine Methode **Start()** zum Öffnen der Verbindung zu ActiveMQ sowie eine Methode **CreateSession(…)** zum Erzeugen einer Instanz der Klasse **Session**. Diese Klasse hat Methoden **GetQueue(string name)** und **GetTopic(string name)**, um sich ein **Queue**- bzw. **Topic**-Objekt erzeugen zu lassen. Die Klassen **Queue** und **Topic** implementieren beide die Schnittstelle **IDestination**, die im Namensraum **Apache.NMS** definiert ist. Mit diesen Informationen kann man dann mit den Methoden **CreateConsumer(IDestination destination)** und **CreateProducer(IDestination destination)** Instanzen der Klassen **MessageConsumer** bzw. **MessageProducer** erzeugen, welche Methoden zum Empfangen bzw. Versenden von Nachrichten bereitstellen (s. Listing 2). Zum transaktionalen Versenden und Empfangen von Nachrichten stellt die Klasse **TransactionContext** entsprechende Methoden zur Verfügung.

Eine Auswahl wichtiger Klassen des Namensraums **Apache.NMS.ActiveMQ.Commands** ist in Abbildung 3 zu sehen. Dieser Namensraum enthält die konkreten Implementierungen für ActiveMQ-Objekte, wie Queues, Topics und die verschiedenen Nachrichtenarten (Text-, Stream-, Object- und Map-Messages).

```
using System;
using Apache.NMS;
using Apache.NMS.Util;
using Apache.NMS.ActiveMQ;
using Apache.NMS.ActiveMQ.Commands;

class VerySimpleAMQConsumer
{
    static void Main(string[] args)
    {
        try
        {
            ConnectionFactory cf = new ConnectionFactory("tcp://localhost:61616");
            Connection connection = (Connection)cf.CreateConnection();
            connection.Start();
            Session session = (Session)connection.CreateSession(AcknowledgementMode.AutoAcknowledge);
            IDestination destination = session.GetQueue("TestQ");
            MessageConsumer msgConsumer = (MessageConsumer)session.CreateConsumer(destination);
            Console.WriteLine("VerySimpleAMQConsumer connected to ActiveMQ queue 'TestQ'.");
            Console.WriteLine("Waiting for Messages in JMS Queue 'TestQ'.\n");
            ActiveMQTextMessage msg = (ActiveMQTextMessage)msgConsumer.Receive();
            Console.WriteLine("Message received:     {0}", msg.Text);
            msgConsumer.Close();
            session.Close();
            connection.Close();
            Console.WriteLine("\nPress <Enter> to exit.");
            Console.ReadKey();
        }
        catch (Exception e)
        {
            Console.Error.WriteLine(e.Message);
        }
    }
}
```

Abb. 3: Ausschnitt aus dem Inhalt des Namensraums Apache.NMS.ActiveMQ.Commands, dargestellt im Visual Studio Object-Browser

```
import javax.jms.*;
import org.apache.activemq.ActiveMQConnectionFactory;

public class VerySimpleAMQConsumer
{
    public static void main(String args[])
    {
        try
        {
            ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61616");
            Connection connection = cf.createConnection();
            connection.start();
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            Destination destination = session.createQueue("TestQ");
            MessageConsumer msgConsumer = session.createConsumer(destination);
            System.out.println("SimpleAMQConsumer connected to ActiveMQ queue 'TestQ'.");
            System.out.println("Waiting for Messages in JMS Queue 'TestQ'.\n");
            TextMessage msg = (TextMessage)msgConsumer.receive();
            System.out.println("Message received:     " + msg.getText());
            msgConsumer.close();
            session.close();
            connection.close();
            System.out.print("\nPress <Enter> to exit.");
            System.in.read();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Abb. 2: Inhalt des Namensraums Apache.NMS.ActiveMQ, dargestellt im Visual Studio Object-Browser

Noch ein wichtiger Hinweis: In dem Verzeichnis der .NET-Applikation, welche auf ActiveMQ zugreift, müssen nicht nur die oben genannten Assemblys vorhanden sein sondern auch die Datei **nms-provider-tcp.config**, die den folgenden Inhalt hat:

```
<configuration>
  <provider assembly="Apache.NMS.ActiveMQ.dll"
    classFactory="Apache.NMS.ActiveMQ.ConnectionFactory"/>
</configuration>
```

Befindet sich diese Datei nicht in dem Verzeichnis, wird beim Aufbau der Verbindung zu ActiveMQ eine NMS-Ausnahme mit der folgenden Fehlermeldung ausgelöst: **No IConnectionFactory implementation found for connection URI: tcp://localhost:61616/**. Diese Datei befindet sich in dem Verzeichnis, in dem sich auch das Assembly **Apache.NMS.ActiveMQ.dll** befindet.

NMS und das JMS API

Die NMS-Programmierschnittstelle hat sehr große Ähnlichkeit mit der von JMS (Java Message Service). Diese Ähnlichkeit ist beim Betrachten der Programmcodes in den Listings 1 und 2 deutlich zu erkennen. In Listing 1 ist die

Implementierung eines Java-Messages-Consumers mit dem JMS API für ActiveMQ zu sehen, in Listing 2 ist das äquivalente Programm mit C# und dem NMS API dargestellt. NMS bietet somit für Microsoft .NET und Mono ein einheitliches Programmiermodell für die Entwicklung asynchroner, nachrichtenorientierter Applikationen auf der Basis der Message-Queue-Systeme ActiveMQ, Microsoft Message Queue (MSMQ) und TIBCO EMS. Die große Ähnlichkeit mit dem JMS API erleichtert Java-Programmierern, die bis jetzt mit JMS gearbeitet haben, den Einstieg in dieses Gebiet der .NET-Programmierung.

ActiveMQ und das SPRING.net-Framework

SPRING.net ist ein Open-Source-Framework für Applikationsentwicklung mit Microsoft .NET, welches sich sehr stark an das Framework Spring für Java/JEE anlehnt. Es bietet mit dem Modul `Spring.Messaging.Nms` eine weitere Möglichkeit, von Microsoft .NET-Applikationen auf ActiveMQ zuzugreifen. Das Modul basiert auf der oben vorgestellten NMS API-Implementierung in C#. Die Web-Site von SPRING.net ist unter [Spring] zu finden. Dort sind auch die Links zum Herunterladen der Quellen sowie der binären Versionen zu finden.

Zielsetzung des SPRING.net-Messaging-Frameworks ist es, eine einheitliche, einfach zu benutzende Klassenbibliothek unter .NET zu bieten, welche auf bewährten Vorgehensweisen und Mustern für die Entwicklung von nachrichtenorientierten Applikationen beruht. Die Module `Spring.Messaging`, `Spring.Messaging.Ems` und `Spring.Messaging.Xms` unterstützen MSMQ, TIBCO EMS und IBM Websphere MQ. Wer also nicht mit einer JMS ähnlichen Programmierschnittstelle wie NMS arbeiten möchte, sollte dieses Framework nutzen.

ActiveMQ und Python

Es existieren auch verschiedene Python-Module, mit denen man Clients für ActiveMQ entwickeln kann. Eines der unter [Python] genannten Module ist `pyactivemq`. Es bildet einen Python-Wrapper um die ActiveMQ-C++-Bibliothek, daher wird sowohl das OpenWire- als auch das STOMP-Protokoll von `pyactimemq` unterstützt. Es kann von der in [PyActiveMq] genannten Webseite heruntergeladen werden. Eine einfache Beispielimplementierung eines ActiveMQ-Clients, der Textnachrichten versendet, ist in Listing 3 zu sehen.

Auch an diesem Python-Beispiel ist die große Ähnlichkeit mit dem JMS API zu erkennen.

```
from pyactivemq import ActiveMQConnectionFactory
from pyactivemq import AcknowledgeMode
from pyactivemq import DeliveryMode
from pyactivemq import TextMessage

n = 1000

cf = ActiveMQConnectionFactory(
       'tcp://localhost:61616?wireFormat=openwire')
connection = cf.createConnection()
session = connection.createSession(
            AcknowledgeMode.DUPS_OK_Acknowledge)
destination = session.createQueue(destination)
producer = session.createProducer(destination)
producer.deliveryMode = DeliveryMode.NON_PERSISTENT

connection.start()
print "SimplePyProducer connected to ActiveMQ queue 'TestQ'.\n"
```

```
for i in range(n)
  msg = session.createTextMessage("Hello from Python: " + str(i))
  print "Sending: %s" %msg.text
  producer.send(msg)

endmsg = session.createTextMessage("EOM")
producer.send(endmsg)

producer.close()
connection.close()
```

Listing 3: Einfaches Python-Skript zum Versenden von Nachrichten

Zusammenfassung

Apache ActiveMQ ist ein zu JMS 1.1 konformes MOM-System, welches nicht nur die Java/JEE-Plattform unterstützt, sondern für das es Schnittstellen für Microsoft .NET und viele Skriptsprachen wie Python, Ruby, Perl und PHP gibt. Für .NET stehen das reine NMS API zur Verfügung, welches große Ähnlichkeit mit dem JMS API besitzt, aber auch Module aus dem SPRING.net-Framework. Diese .NET-Komponenten ermöglichen es, ActiveMQ als Message-Broker von allen Programmier- und Skriptsprachen, die es für .NET gibt, zu nutzen.

Damit ist ActiveMQ ein hervorragendes Werkzeug für die asynchrone, nachrichtenorientierte Integration in einem heterogenen Applikationsumfeld.

Links

[ActiveMQ] Apache Software Foundation, `http://activemq.apache.org/`

[NMS] Apache Software Foundation, The .NET Messaging API, `http://activemq.apache.org/nms/`, Download unter `http://activemq.apache.org/nms/activemq-net-110-release.html`

[OpenWire] Apache Software Foundation, OpenWire Version 2 Specification, `http://activemq.apache.org/openwire-version-2-specification.html`

[PyActiveMq] Python module for communication with the ActiveMQ message broker, `http://code.google.com/p/pyactivemq/`

[Python] Liste von Python-Modulen für ActiveMQ, `http://stomp.codehaus.org/Python`

[SnDaBo09] B. Snyder, R. Davies, D. Bosanac, Active MQ in Action, Manning Publications, 2009, Manning Early Access Program, `http://www.manning.com/snyder/`

[Spring] SPRING.net, `http://springframework.net/`, speziell Kapitel 29, `http://www.springframework.net/docs/1.3.0/reference/html/messaging.html`

Klaus Rohe arbeitet bei der Microsoft Deutschland GmbH. E-Mail: klaus.rohe@microsoft.com.