

**.NET, J2EE & CORBA in enger Beziehung**

# Integrationsmöglichkeiten von .NET-Anwendungen mit J2EE und CORBA

Rainer von Ammon, Klaus Rohe

Zunehmend werden in Unternehmen sowohl J2EE- und CORBA- als auch .NET-Applikationen eingesetzt. Der Artikel stellt zunächst verschiedene Möglichkeiten für die Integration dieser unterschiedlichen Komponentenplattformen vor und diskutiert anschließend eine Implementierung für eine .NET-Java-Integration anhand der Opensource-Implementierung IOP.NET. Weitere konkrete Anwendungsbeispiele zeigen, welche Techniken in der Praxis bereits verwendet werden. Zuletzt bietet der Artikel Entscheidungshilfen für die Auswahl einer geeigneten Integrationstechnik für verschiedene Einsatzkontexte.



## Integrationszenarien und Integrationstechniken

Für die Integration von .NET- und J2EE/CORBA-Applikationen existieren verschiedene Szenarien. Aus den Erfahrungen in der Praxis sind gegenwärtig folgende Szenarien von Bedeutung (s. Abb. 1):

- Integration des Presentation-Tier von .NET mit dem Middle-Tier von J2EE/CORBA: Dabei kann es sich um einen .NET-Smart-Client handeln, der zum Beispiel mit WindowsForms oder Office 2003 entwickelt wurde und der auf eine bestehende J2EE- oder CORBA-Applikation zugreifen muss. Die Integration kann synchron erfolgen oder asynchron über eine „message oriented middleware“ (MOM).

- Integration des Middle-Tier von .NET mit dem von J2EE/CORBA: Die Integration kann wieder, je nach Anforderung, synchron bzw. asynchron mit Hilfe einer MOM durchgeführt werden.

Diese beiden grundsätzlichen Integrationszenarien können mit jeder der folgenden Integrationstechniken realisiert werden. Welche die adäquate ist, muss im Einzelfall auf der Basis der spezifischen Anforderungen entschieden werden. Die meisten dieser Integrationstechniken nutzen Binärprotokolle zur Kommunikation zwischen der .NET- und der Java-Welt,

was derzeit – verglichen mit SOAP – noch zu einer besseren Netzwerk-Performance führt.

Zu den vorgestellten Integrationstechniken werden konkrete Produkte vorgestellt, die Liste ist allerdings nicht umfassend und stellt auch keine Empfehlung der Autoren dar. Ferner soll dargestellt werden, wie sie sich in den Kontext von Web-Services einordnen.

## .NET-Java-Bridging

Mit .NET-Java-Bridges kann eine .NET-Applikation die Methoden von Objekten einer Java-Applikation so nutzen, als handle es sich um lokale .NET-Objekte. Die meisten .NET-Java-Bridges unterstützen bidirektionale Kommunikation, wodurch man umgekehrt in Java-Applikationen auch die Methoden von .NET-Objekten nutzen kann. Allen .NET-Java-Bridges ist gemeinsam, dass sie Proxy-Klassen für den Zugriff auf die gewünschte Zielplattform generieren:

- Sollen aus einer .NET-Applikation heraus Java-Klassen genutzt werden, generiert man aus Java-class- oder Java-jar-Dateien .NET-Proxy-Klassen. Diese Proxy-Klassen und die .NET-Laufzeit-Komponenten der .NET-Java-Bridge müssen zusammen mit dem Applikations-Code kompiliert und gebunden werden. Der Kommunikationsmechanismus zwischen .NET und Java wird über eine Konfigurationsdatei definiert.

- Beabsichtigt man, von einer Java-Applikation aus .NET-Klassen zu nutzen, werden aus den entsprechenden .NET-Assemblies Java-Proxy-Klassen erzeugt. Die aus den Proxy-Klassen kompilierten .class-Dateien und die Java-Laufzeit-Komponenten der .NET-Java-Bridge müssen über den CLASSPATH der Applikation bekannt gemacht werden. Der Kommunikationsmechanismus zwischen Java und .NET wird über eine Konfigurations- oder Property-Datei definiert.

Das Proxy-Generierungswerkzeug sorgt auch dafür, dass Java-Datentypen, die in .NET nicht existieren, auf die entsprechenden .NET-Datentypen abgebildet werden. Ist das nicht möglich, werden entsprechende Hilfs-Klassen erzeugt, die auch als Support-Klassen bezeichnet werden.

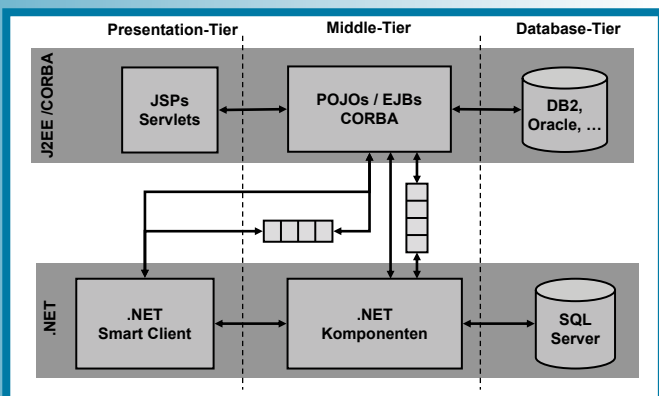


Abb. 1: Integrationszenarien

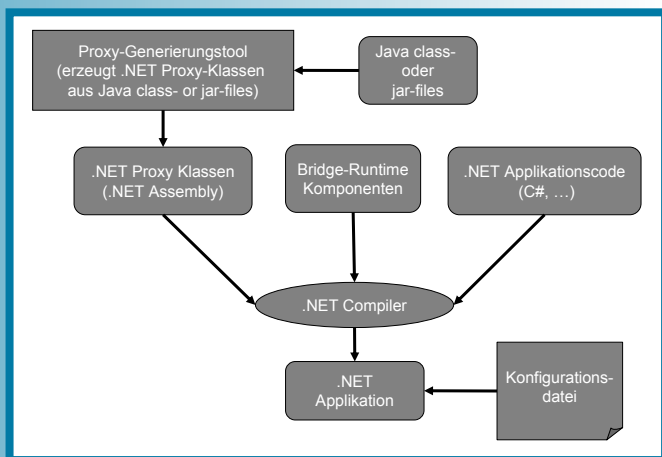


Abb. 2: Entwicklungsprozesse mit .NET-Java-Bridges

Die prinzipielle Vorgehensweise bei der Integration von .NET- und Java-Applikationen mit Hilfe einer .NET-Java-Bridge ist für die Integrationsrichtung „.NET nach Java“ in Abbildung 2 gezeigt, die umgekehrte Richtung funktioniert analog. Für die Kommunikation zwischen .NET- und Java-Komponenten nutzen die .NET-Java-Bridges entweder .NET Remoting [Net-Rem] oder eine Erweiterung der *Java Virtual Machine* mit Hilfe des *Java Native Interface* (JNI).

Microsoft .NET Remoting erlaubt es, verteilte, objektorientierte Systeme zu entwickeln, vergleichbar mit Java RMI. .NET Remoting unterstützt standardmäßig ein binäres Protokoll über TCP/IP oder SOAP über HTTP. Die .NET Remoting-Architektur ist offen und erweiterbar, wodurch spezifische Kommunikationsprotokolle, wie zum Beispiel IIOP, implementiert werden können. Es lassen sich auch neue Transportmechanismen realisieren, wie zum Beispiel die Kommunikation über „shared memory“ oder „named pipes“. Details zu diesem Thema findet man in [LuLTa] und [HoWo04].

## Anwendungsszenarien von .NET-Java-Bridging

Es gibt zwei Szenarien, für die man .NET-Java-Bridging sinnvoll anwenden kann. Im ersten Szenario will man eine Java-Server-Applikation von .NET aus nutzen. Das kann ein Java-RMI-Server sein, der reine RMI-Objekte enthält, oder ein Java-Applikations-Server, der EJBs zur Verfügung stellt. In diesem Szenario kommunizieren die .NET- und Java-Anwendung im Allgemeinen über einen .NET Remoting-Kanal für IIOP.

Im zweiten Szenario will man die Funktionalität von vorhandenen Java-Klassen oder -Klassenbibliotheken in .NET-Applikationen nutzen. Diese Java-Klassen können als .class- oder .jar-Dateien vorliegen. Im Vordergrund steht hier grundsätzlich die Wiederverwendung von vorhandenem Java-Code, um so Portierungsaufwand zu sparen. In diesem Szenario wird die *Java Virtual Machine* im Prozessraum der .NET-Applikation ausgeführt und es wird über Shared-Memory kommuniziert, eventuell unter Nutzung von JNI. Eine andere Möglichkeit besteht darin, dass der Java-Reflection-Mechanismus genutzt wird, um dynamisch einen Java-RMI-Server zu generieren, der dann die gewünschten Klassen als RMI-Objekte verfügbar macht. Die Kommunikation zwischen .NET und Java geschieht dann wieder über einen .NET Remoting-Kanal für IIOP. Der Kasten auf dieser Seite stellt einige .NET-Java-Bridging-Produkte und ihre Eigenschaften vor.

## .NET-Java-Bridging-Produkte

### J-Integra

J-Integra (<http://j-integra.intrinsyc.com/net/info/>) von Intrinsyc basiert auf .NET Remoting und unterstützt ein binäres Protokollformat sowie SOAP über HTTP. J-Integra unterstützt bidirektionale Integration, d. h. man kann sowohl von .NET auf Java-Applikationen zugreifen als auch von Java auf .NET-Applikationen. J-Integra integriert sich in die Entwicklungsumgebung Visual Studio 2003.

### JNBridge

JNBridge (<http://www.jnbridge.com/index.htm>) von dem gleichnamigen Softwarehaus beruht auch auf .NET Remoting. Es ist eine Kommunikation über ein binäres Protokoll, SOAP über HTTP sowie über ein spezielles Shared-Memory-Protokoll möglich. Bei der .NET-Java-Kommunikation über Shared-Memory wird die Java VM im .NET-Prozess über die *jvm.dll* gestartet.

### IIOP.NET

IIOP.NET (<http://iiop-net.sourceforge.net/>) ist ein Open-source-Produkt und unter der LGPL verfügbar. Es hat seinen Ursprung in einer Diplomarbeit am Institut von Prof. Jürg Gutknecht an der ETH Zürich. IIOP.NET wird von der Schweizer Firma Elca Informatique SA weiterentwickelt. IIOP.NET implementiert einen Remoting-Kanal für IIOP. Es unterstützt bidirektionale Integration zwischen .NET und J2EE/CORBA.

### JuggerNET

JuggerNet (<http://www.codemesh.com/en/JuggerNETCurrentRelease.html>) wird von der Firma Codemesh entwickelt und vertrieben. JuggerNet startet die JVM (*jvm.dll*) im Prozessraum der .NET-Applikation. Die Kommunikation geschieht über Shared-Memory. Die Java-Klassen werden von .NET durch Proxy-Klassen angesprochen, die mit einem Proxy-Generierungswerkzeug erzeugt werden. Das Proxy-Generierungswerkzeug liegt sowohl als GUI als auch als Kommandozeilenversion vor. Mit JuggerNET ist nur die unidirektionale Integration von .NET nach J2EE möglich.

## CORBA/IIOP

Bei dieser Integrationstechnik wird die Verbindung von .NET-, J2EE- und CORBA-Applikationen dadurch erreicht, dass man auf der Basis von Microsoft .NET einen CORBA-Stack implementiert, durch den man auf beliebige CORBA-, Java-RMI- und J2EE-Applikationen von .NET aus zugreifen kann. Das impliziert, dass man auf der .NET-Seite mit dem CORBA-Programmiermodell arbeiten muss. Zur Vereinfachung für den .NET-Programmierer bieten die meisten Produkte in dieser Kategorie die Möglichkeit, anstelle des CORBA-Programmiermodells das .NET-Programmiermodell zu nutzen.

Beim Zugriff von .NET auf eine reine CORBA-Applikation ist die IDL-Datei von CORBA der Ausgangspunkt. Aus der IDL-Datei werden mit dem IDL-Compiler .NET-Client-Stubbs erzeugt, die dann in der .NET-Applikation zum Zugriff auf die CORBA-Anwendung verwendet werden. Handelt es sich nicht um CORBA, sondern um eine J2EE- oder Java-RMI-An-

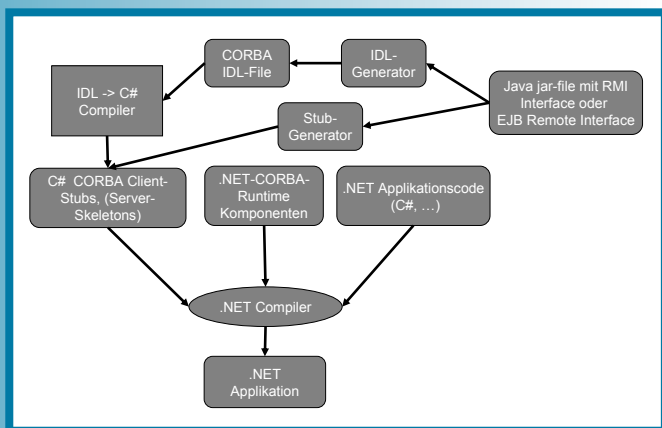


Abb. 3: Entwicklungsprozess mit .NET CORBA

wendung, wird aus der Java-jar-Datei mit den RMI-Interfaces mithilfe eines entsprechenden Generator-Programms eine IDL-Datei erzeugt. Aus dieser IDL-Datei generiert man dann die notwendigen .NET-Client-Stubs für CORBA. Manche Produkte unterstützen auch das direkte Generieren der .NET-Client-Stubs aus den Java-jar-Dateien ohne den Umweg über CORBA-IDL. Die prinzipielle Vorgehensweise bei der Integration von .NET- und Java-Applikationen mit Hilfe von .NET-basierten CORBA-Stacks ist für die Integrationsrichtung „.NET nach CORBA/J2EE“ in Abbildung 3 gezeigt.

Will man mit dieser Technik von Java bzw. CORBA auf .NET-Applikationen zugreifen, muss man .NET CORBA-Server implementieren, die die benötigte Funktionalität als CORBA-Schnittstellen zur Verfügung stellen, d. h. man muss einen entsprechenden CORBA-Server mit .NET implementieren. Der Kasten auf dieser Seite zeigt entsprechende Produkte für diese Technik.

## Message Queuing

Message-Queue-Systeme oder allgemein „message oriented middleware“ (MOM) eignen sich sehr gut zur Nachrichten-basierten, asynchronen Integration von Applikationen [LuLTa]. Durch diese Art der Integration erreicht man eine lose Kopplung zwischen den Applikationen. Solange das Nachrichtenformat bzw. Nachrichtenschema konstant gehalten wird, kann sich die Implementierung einer integrierten Applikation ändern ohne Auswirkungen auf andere Applikationen. Weiterhin bieten Message-Queue-Systeme Transaktionsunterstützung, sodass man Nachrichten zuverlässig versenden kann. Als konkrete Message-Queue-Systeme kommen für die .NET-J2EE/CORBA-Integration *Microsoft Message Queue* (MSMQ) sowie *WebSphere MQ* (MQSeries) von IBM und auch der *Java Messaging Service* (JMS) in Frage. Diese drei Möglichkeiten werden im Folgenden genauer erläutert.

## Integration von .NET-Applikationen in J2EE/CORBA-Anwendungen über MSMQ

Falls der Integration mit Hilfe von Web-Services keine prinzipiellen Gründe widersprechen, kann man von der J2EE/CORBA-Seite aus die SOAP-Schnittstelle von MSMQ nutzen, die derzeit mit der Version 3.0 zur Verfügung steht. Ist das nicht möglich, kann mit .NET-Java-Bridging-Produkten, die die uneingeschränkte bidirektionale Kommunikation unterstützen,

## .NET-CORBA/IIOP-Produkte

### Janeva

Janeva (<http://www.borland.com/janeva/>, [Stal04a]) von Borland ist eine Portierung der Java-Version von VisiBroker auf J#. Janeva integriert sich in Visual Studio .NET und unterstützt sowohl das CORBA-Programmiermodell als auch .NET Remoting. Mit Janeva können sowohl CORBA-Clients als auch CORBA-Server unter .NET implementiert werden, wodurch die bidirektionale Integration von .NET und J2EE/CORBA unterstützt wird.

### J-Integra Espresso und MinCor

J-Integra Espresso ([http://j-integra.intrinsyc.com/products/corba\\_net.asp](http://j-integra.intrinsyc.com/products/corba_net.asp)) und MinCor von Middsol (<http://www.middsol.de/index.htm>, [Stal04b]) sind CORBA-Implementierungen für .NET. MinCor richtet sich an das .NET Compact Framework. Beide unterstützen sowohl das CORBA-Programmiermodell als auch .NET Remoting. Mit J-Integra Espresso/MinCor können sowohl CORBA-Clients als auch CORBA-Server unter .NET implementiert werden, wodurch die bidirektionale Integration von .NET und J2EE/CORBA unterstützt wird.

J-Integra Espresso von Intrinsyc basiert auf J-Integra Espresso von der Middsol GmbH. Intrinsyc Software International, Vancouver, hat die Technologie, welche unter dem Namen Middcor bekannt war, exklusiv von der Middsol GmbH, Hamburg, lizenziert. Die Details der Vereinbarung sind in der Pressemitteilung vom 16.06.2005 zu finden ([http://j-integra.intrinsyc.com/company/06\\_16\\_2005.asp](http://j-integra.intrinsyc.com/company/06_16_2005.asp)).

eine entsprechende Java-.NET-Bridge realisiert werden. Zur Integration von CORBA-Applikationen mit MSMQ kann man mit den oben besprochenen .NET-CORBA-Implementierungen einen .NET-CORBA-Server implementieren, der eine CORBA-Schnittstelle zu MSMQ bietet, die dann sowohl von CORBA als auch von Java aus genutzt werden kann. Die letzte Option ist zur Verdeutlichung nochmals in Abbildung 4 dargestellt.

Analog könnte man auf der J2EE/CORBA-Seite einen Java-.NET-Bridge-Prozess implementieren, der der J2EE-Applikation einen Zugriff auf MSMQ gestattet.

## Integration von .NET-Applikationen in J2EE/CORBA-Anwendungen über JMS

Von .NET aus kann man JMS am einfachsten über .NET-Java-Bridging integrieren, indem man für die Java-Klassenbibliotheken, die das JMS-API implementieren, .NET-Proxy-Klassen erzeugt. Mit Hilfe dieser Proxy-Klassen und mit den Laufzeitkomponenten der .NET-Java-Bridge kann man dann von .NET-Anwendungen direkt auf JMS zugreifen. Für diese Integration eignen sich aus Gründen der Performanz und der einfacheren Architektur besonders die Bridge-Produkte, mit denen der Java-Code im Prozessraum der .NET-Applikation ausgeführt werden kann (vgl. Abb. 5).

Auf die JMS-Implementierung der Firma TIBCO, den *TIBCO Enterprise Message Service* [TIBCO], kann man mit einem JMS-konformen .NET-API zugreifen.

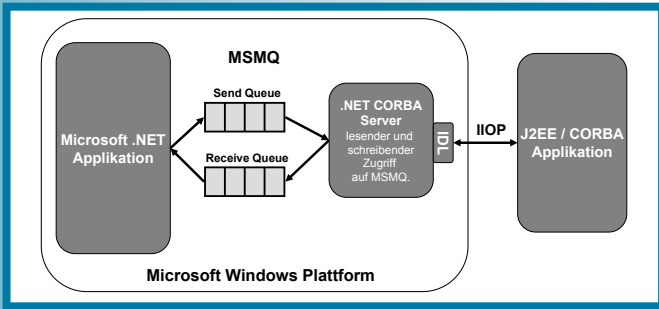


Abb. 4: Integration mit MSMQ und .NET CORBA

### Integration von .NET-Anwendungen in J2EE-Anwendungen über IBM WebSphere MQ

Für WebSphere MQ stellt IBM ein .NET-API zur Verfügung, sodass man dessen Funktionalität direkt aus einer .NET-Applikation nutzen kann. Über das JMS-API von WebSphere MQ [MQ] ist dann eine Integration mit J2EE problemlos möglich. Eine detaillierte Beschreibung dieses Szenarios ist in [LuLTa] und [LuLTb] zu finden. Damit schließen wir die Vorstellung der Integrationstechniken und der entsprechenden Produkte ab.

### Beispiel einer Implementierung für eine .NET-Java-Integration anhand von IIOP.NET

Der Code in den Listings 1 und 2 zeigt ein einfaches Beispiel für die Interoperabilität eines .NET-Client (Programmiersprache C#, Microsoft .NET Framework 1.1) mit einem Java RMI Server (JSE 1.5). Der Java RMI Server `RMISimpleServer` stellt ein RMI-Objekt `RMISimpleImpl` mit der Methode `String simple(String str)` zur Verfügung. Diese Methode liefert als Rückgabewert das aktuelle Datum, konkateniert mit der Zeichenkette, die der Methode als Parameter übergeben wird. Das RMI-Objekt wird unter dem Namen `rmiSimpleImpl` beim Namensdienst registriert.

Der .NET-Client `SimpleRMIClt` benutzt den von IIOP.NET implementierten Remoting-Kanal für IIOP. Dieser befindet sich in dem .NET Assembly „`IIOPChannel.dll`“, das von IIOP.NET zur Verfügung gestellt wird. Dieses Assembly enthält sowohl Klas-

sen zur Registrierung des IIOP-Remoting-Channels als auch Klassen für den Zugriff auf den CORBA-Namensdienst.

Weiterhin benutzt der .NET-Client das Assembly „`Simple.dll`“, das die .NET-Proxy-Klasse enthält, mit der auf die Methode `simple` des RMI-Objekts `RMISimpleImpl` zugegriffen wird. Zur Erzeugung des Assembly „`Simple.dll`“ muss man dann wie folgt vorgehen:

- ▼ Erzeugen einer OMG-IDL-Datei aus der Java-.class-Datei „`RMI-SimpleImpl.class`“ mit `rmic`-Utility des Java SDK durch das Kommando: `rmic -idl -classpath .\classes Simple.RMISimpleImpl`
- ▼ Erzeugen des Assembly „`Simple.dll`“ mit dem IIOP.NET-Utility `IDLToCLSCompiler` durch Ausführen des Kommandos: `IDLToCLSCompiler Simple IRemoteSimple.idl`

Das ausführbare .NET-Programm `SimpleRMIClt.exe` wird dann durch Compilieren mit dem C#-Compiler (`csc`) erzeugt:

```
csc /r:Simple.dll /r:IIOPChannel.dll /out:SimpleRMIClt.exe SimpleRMIClt.cs
```

Es wird hier der Einfachheit halber davon ausgegangen, dass die Dateien `IRemoteSimple.idl`, `IIOPChannel.dll`, `Simple.dll` und `SimpleRMIClt.cs` alle im gleichen Verzeichnis liegen. Zur Veranschaulichung ist der Entwicklungsprozess für dieses Beispiel als Diagramm in Abbildung 6 dargestellt.

### Anwendungsbeispiele

J-Integra der Firma Intrynsic wird in dem *SAP Portal Development Kit for Microsoft .NET* eingesetzt. Eine weitere Referenz ist JPMorgan mit einem Trading-System, das aus Programmen auf der Basis von J2EE- und .NET-Applikationen besteht, die .NET Remoting nutzen. Für die Kommunikation zwischen diesen Systemen wird J-Integra verwendet.

JNBridge wird unter anderem bei E-Crossnet, einem Londoner Unternehmen aus dem Finanzdienstleistungssektor, eingesetzt, um eine mit .NET entwickelte Reportingsoftware über JMS mit einem Handelssystem zu verbinden, das mit J2EE implementiert wurde.

Das .NET-CORBA-Produkt J-Integra Espresso von Middsol wird am Münchener Flughafen im Terminal II eingesetzt. Das „Flight-Information-Display-System“ im Terminal II wurde mit Microsoft .NET implementiert. Die dargestellten Fluginformationen werden aus einem Fluginformationssystem geholt, das auf CORBA basiert. Für die Integration dieser beiden Systeme wird J-Integra Espresso verwendet. Die .NET-Applikation fungiert als CORBA-Client.

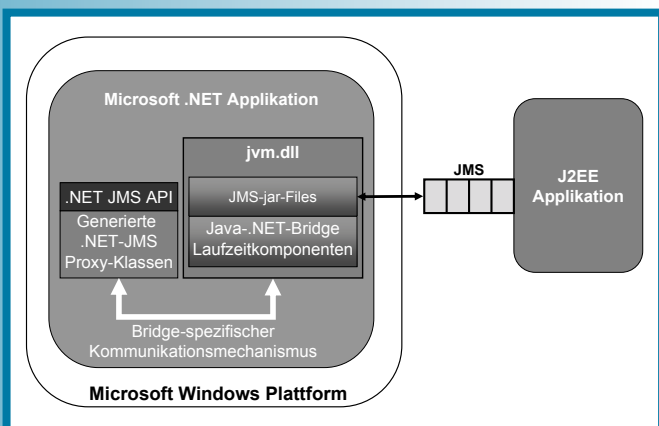


Abb. 5: Integration von JMS .NET-Java-Bridges

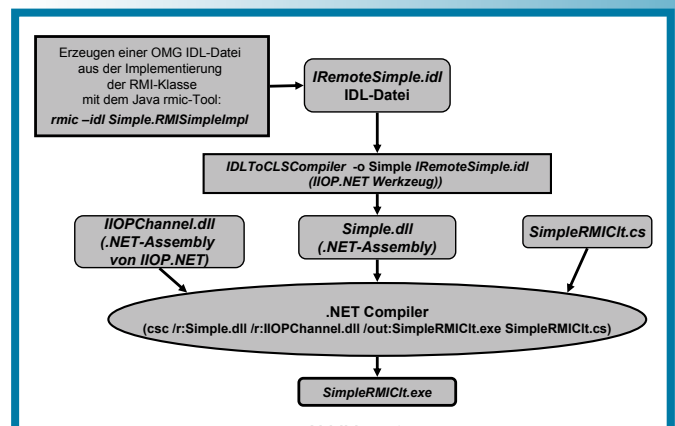


Abb. 6: Entwicklungsprozess mit IIOP.NET

```
// File: IRemoteSimple.java
package Simple;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IRemoteSimple extends Remote {
    public String simple(String str) throws RemoteException;
}

// File: RMISimpleImpl.java
package Simple;
import java.util.*;
import java.text.*;
import java.rmi.RemoteException;
import javax.rmi.PortableRemoteObject;

public class RMISimpleImpl extends PortableRemoteObject
implements IRemoteSimple {
    public RMISimpleImpl() throws java.rmi.RemoteException {
        super();
    }

    public String simple(String str) throws java.rmi.RemoteException {
        String res = null;
        try {
            System.out.println(
                "-----> executing RMISimpleImpl.simple(\"+str+\");");
            Date today = new Date();
            DateFormat df = DateFormat.getDateInstance(
                DateFormat.LONG, DateFormat.LONG);
            res = df.format(today) + " " + str;
            System.out.println("-----> Returning: " + res);
        }
        catch(Exception e) {
            String excmsg = "Exception in RMISimpleImpl.simple(\"+str+\");";
            throw new java.rmi.RemoteException(excmsg);
        }
        return res;
    } // end simple
} // end class RMISimpleImpl

// File: RMISimpleServer.java
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject;

import Simple.*;

public class RMISimpleServer {
    public static void main(String args[]) {
        try {
            RMISimpleImpl rmiSimpleImpl = new RMISimpleImpl();
            Context initialNamingContext = new InitialContext();
            initialNamingContext.rebind("rmiSimpleImpl", rmiSimpleImpl);
            System.out.println(
                "---> RMI simple server ready to process requests");
        }
        catch (Exception e) {
            System.err.println("Exception in RMISimpleServer: " + e);
            e.printStackTrace();
        }
    } // end main
}
```

Listing 1: Implementierung des Java RMI Servers

## Vergleich der Integrationstechniken

Der Unterschied zwischen der .NET–Java-Bridge-Technik und der Integration über CORBA/IIOP besteht darin, dass bei letzterer die Applikation als CORBA-, J2EE- oder Java-RMI-Server vorliegen muss. Die Bridge-Technologie kann im Gegensatz dazu auch zur reinen Wiederverwendung von

```
// File: SimpleRMIClt.cs
using System;
using System.Runtime.Remoting.Channels;
using Ch.Elca.Iiop;
using Ch.Elca.Iiop.Services;
using omg.org.CosNaming;

class SimpleRMIClt {
    static void Main(string[] args) {
        int nameServicePort = 4500;
        string nameServiceHost;
        string nameOfImpl;

        // Register the channel
        IiopClientChannel iiopChannel = new IiopClientChannel();
        ChannelServices.RegisterChannel(iiopChannel);
        // Access COS naming service
        CorbaInit init = CorbaInit.GetInit();
        nameServiceHost = "localhost";
        NamingContext nameService = init.GetNameService(
            nameServiceHost, nameServicePort);
        // 'nameOfImpl' is the name registered by the Java RMI server
        // with the naming service
        nameOfImpl = "rmiSimpleImpl";
        NameComponent[] name = new NameComponent[] {
            new NameComponent(nameOfImpl, "")
        };
        //Get the reference to the simpleImpl object
        Simple.IRemoteSimple simpleClt =
            (Simple.IRemoteSimple)nameService.resolve(name);
        Console.WriteLine("String: ");
        string str = Console.ReadLine();
        // Call method simple of RMI server
        Console.WriteLine("--> RMI simple server returned: {0}",
            simpleClt.simple(str));
        Console.ReadLine();
    }
}
```

Listing 2: Implementierung des .NET RMI Clients (Programmiersprache C#)

Java-Klassenbibliotheken genutzt werden, die als .jar-Dateien vorliegen.

Ob man die besprochenen Bridging- und .NET-CORBA-Implementierungen, Message-Queueing oder Web-Services einsetzen soll, ist von den spezifischen Integrationsanforderungen abhängig. Tabelle 1 (siehe nächste Seite) und Abbildung 7 geben eine erste Hilfestellung, welche Kriterien beachtet werden sollten.

## Literatur und Links

- [Gue04] S. Guest, Microsoft .NET and J2EE Interoperability Toolkit, Microsoft Press, 2004
- [HoWo04] G. Hohpe, B. Woolf, Enterprise Integration Patterns, Addison-Wesley, 2004
- [J-Integra] NET–Java-Bridging-Produkt J-Integra von Intrinsic, <http://j-integra.intrinsic.com/net/info/>
- [JNBridge] NET–Java-Bridging-Produkt JNBridge, <http://www.jnbridge.com/index.htm>
- [LuLTa] B. Lublinsky, D. Le Tien, Implementing J2EE-.NET Interoperability using WebSphere MQ Part 1, in: Websphere Journal, Volume 02, Issue 10
- [LuLTb] B. Lublinsky, D. Le Tien, Implementing J2EE-.NET Interoperability using WebSphere MQ Part 2, in: Websphere Journal, Volume 02, Issue 11



Frage	Ja	Nein
Handelt es sich um hohes Datenvolumen und ist kurze Latenzzeit gefordert?	Kommunikation auf Binärlevel (.NET-J2EE- Bridges, Janeva, J-Integra Espresso usw.)	Web-Services nutzen
Kann der Code des Zielsystems modifiziert/angepasst werden? Schnittstellen nicht zu feingranular?	Schnittstellen als Web-Services kapseln	Kommunikation auf Binärlevel (.NET – J2EE Bridges, Janeva, J-Integra Espresso usw.)
Ist zuverlässige, asynchrone, nachrichtenbasierte Kommunikation vorteilhaft?	MOM nutzen, eventuell in Verbindung mit Kommunikation auf Binärlevel (.NET-J2EE -Bridges, Janeva, J-Integra Espresso usw.)	Web-Services nutzen
Gibt es eine komplexe Folge von Aufrufen an das Zielsystem und sind aufwändige Datentransformationen nötig?	Microsoft BizTalk Server 2004 mit den entsprechenden Adaptern	Web-Services oder Kommunikation auf Binärlevel (.NET-J2EE-Bridges, Janeva, J-Integra Espresso usw.)

Tabelle 1: Auswahl der geeigneten Integrationstechnik

[J-Integra Espresso] NET-CORBA/IIOP-Produkt J-Integra

Espresso von Middsol, <http://www.middsol.de/index.htm>

[MQ] WebSphere MQ classes for Microsoft .NET,

[http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24004732&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24004732&loc=en_US&cs=utf-8&lang=en)

[Net2J2EE1] Application Interoperability: Microsoft .NET and

J2EE, Microsoft Corporation 2004, <http://www.microsoft.com/downloads/details.aspx?FamilyId=5FBA8E7A-B896-4E5F-B3C0-FCF7FF1B9D29&displaylang=en>

[NETRem] S. McLean, J. Naftel, K. Williams, Microsoft .NET Remoting, Microsoft Press, 2003

[New] T. Neward, .NET and Java: A Study in Interoperability, <http://www.theserverside.net/articles/showarticle.tss?id=Interoperability>

[Ram02] I. Rammer, Advanced .NET Remoting (C# Edition),

APress, 2002

[Schw04] H. Schwichtenberg, Brückenschlag zwischen den Systemen, in: Computer Zeitung Nr. 33/34, 16. August 2004

[Stal04a] M. Stal, Über-Brückung Janeva: Zugriff von .Net- auf J2EE- und CORBA-Objekte, review iX 2/04, Seite 62

[Stal04b] M. Stal, Join them J-Integra Espresso.Net: Zusammenspiel von .Net mit J2EE und CORBA, review, iX 10/04, Seite 60

[TIBCO] TIBCO Enterprise Message Service, [http://www.tibco.com/software/enterprise\\_backbone/enterprisemessageservice.jsp](http://www.tibco.com/software/enterprise_backbone/enterprisemessageservice.jsp)

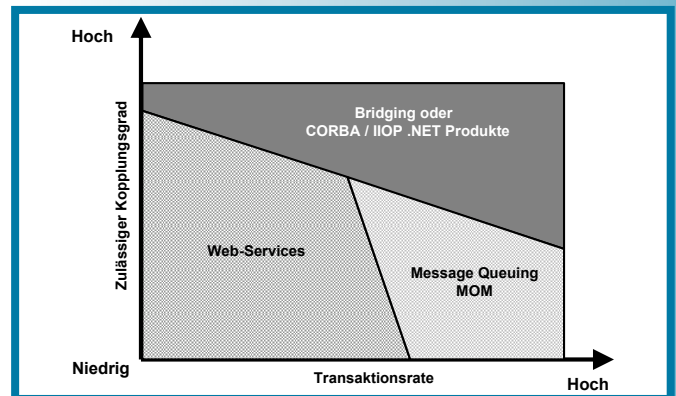


Abb. 7: Einsatzbereiche der .NET-J2EE/CORBA-Interoperabilitätstechnologien



**Dr. Rainer von Ammon** ist Geschäftsführer der Centrum für Informations-Technologie-Transfer (CITT) GmbH. E-Mail: [rainer.ammon@citt-online.com](mailto:rainer.ammon@citt-online.com).



**Dipl.-Phys. Klaus Rohe** arbeitet als .NET-Architekturberater bei der Microsoft Deutschland GmbH. E-Mail: [klrohe@microsoft.com](mailto:klrohe@microsoft.com).