# Mainframe Internet Integration

**Prof. Dr. Martin Bogdan**
**Prof. Dr.-Ing. Wilhelm G. Spruth**

**SS2013**

## Java Remote Method Invocation Teil 3
## RMI over IIOP

# Warum RMI zusätzlich zu CORBA ?

**Neben der reinen Java-Lösung RMI gibt es auf dem weiten Feld der Standards noch das komplexere CORBA.**

**RMI setzt voraus, dass Client und Server in Java geschrieben sind. Im Gegensatz zu RMI definiert CORBA ein großes Framework für unterschiedliche Programmiersprachen. Die Definition von CORBA durch die OMG (Object Management Group) geht in das Jahr 1991 zurück, also auf die Zeit vor RMI.**

**Die Frage nach dem Sinn von RMI ist also erlaubt. Die Antwort liegt in der Einfachheit und Integration von RMI.**

**Seit 2001 hat die Firma Sun RMI an den CORBA Standard angepasst. Die Stellvertreter-Objekte (Stubs, Skeletons) unterstützen mittlerweile nicht nur das Java eigene JRMP Protokoll, sondern zusätzlich auch das CORBA eigene Inter-ORB Protocol (IIOP). Diese Lösung heißt RMI/IIOP („RMI over IIOP“).**

**Mit RMI/IIOP lässt sich sowohl eine Verbindung zwischen zwei in Java geschriebenen Objekten, als auch eine Verbindung zwischen Java Objekten und nicht-Java Objekten herstellen.**

# Vergleich CORBA - RMI

- **Corba Anwendungen können in vielen unterschiedlichen Sprachen geschrieben werden, solange für diese Sprachen ein „Interface Definition Language (IDL) mapping“ vorhanden ist. Dies ist der Fall für Cobol, PL/1, C/C++, Ada, Fortran, SmallTalk, Perl, Ruby, Python und viele weitere Sprachen. RMI in der JRMP Version ist auf Java beschränkt.**

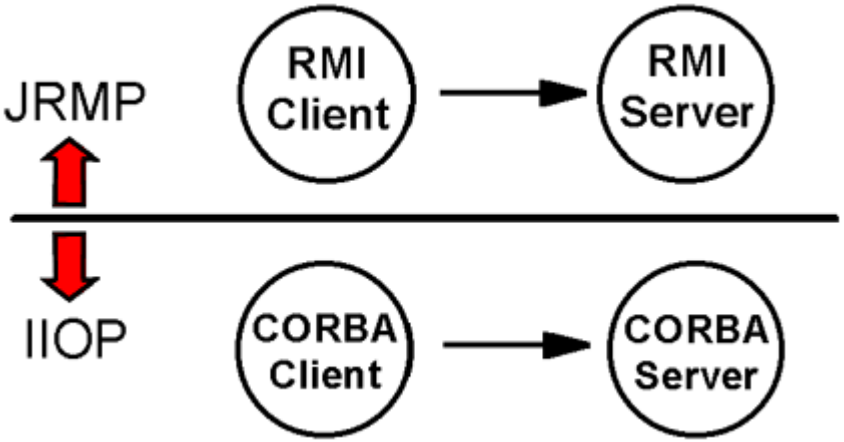
- **Mit der IDL (Interface Definition Language) ist die Interface von der Implementierung sauber getrennt. Es können unterschiedliche Implementierungen unter Benutzung der gleichen Interface erstellt werden.RMI Anwendungen sind einfacher zu programmieren als Corba Anwendungen, weil die Notwendigkeit der IDL Definition entfällt. Die Interface ist bereits ein Sprachkonstrukt von Java.**

- **RMI ermöglicht serialisierbare Klassen. Code und Objekte können über das Netz übertragen werden (can be marshaled), solange der Empfänger über eine Java Virtuelle Maschine (JVM) verfügt. CORBA erlaubt keine Übertragung von Code oder Objekten; es können nur Datenstrukturen übertragen werden.**

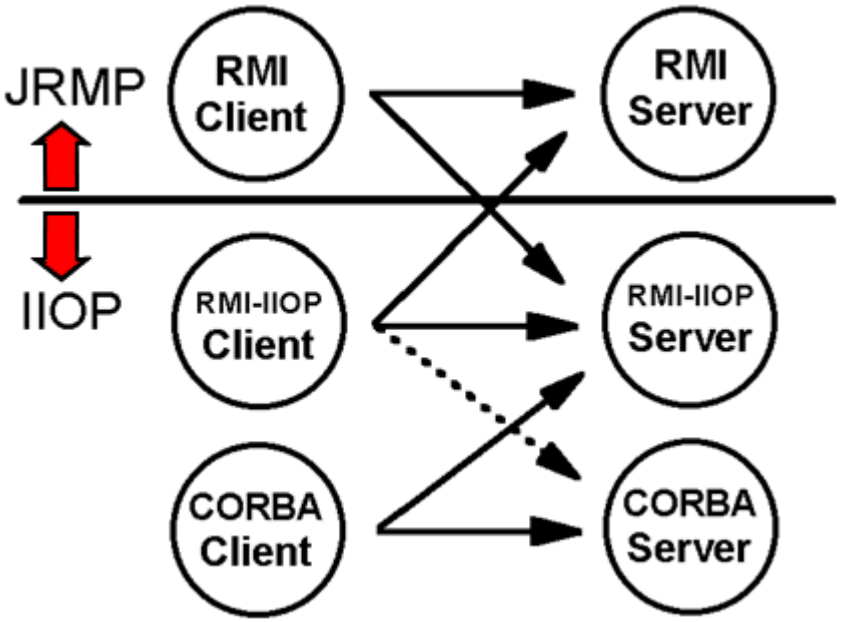
- **Corba hat ein besseres Leistungsverhalten als RMI (keine Interpretation).**

**Die Vorteile beider Ansätze lassen sich durch den Einsatz des RMI/IIOP Protokolls kombinieren.**

# Vor dem Erscheinen von RMI/IIOP




**Schauen Sie sich die obige Abbildung an. Der Raum über der mittleren horizontalen Linie stellt die ursprünglichen Domäne von RMI und dessen JRMP Protokoll dar; der untere Bereich stellt die Welt von CORBA und IIOP dar. Diese beiden getrennten Welten, die unabhängig voneinander entwickelt wurden, sind historisch nicht imstande gewesen, miteinander zu kommunizieren. Zum Beispiel kann das native RMI Protokoll JRMP (Java Remote Method Protocol) sich nicht mit IIOP (dem Corba Protokoll), oder anderen Protokollen verbinden.**

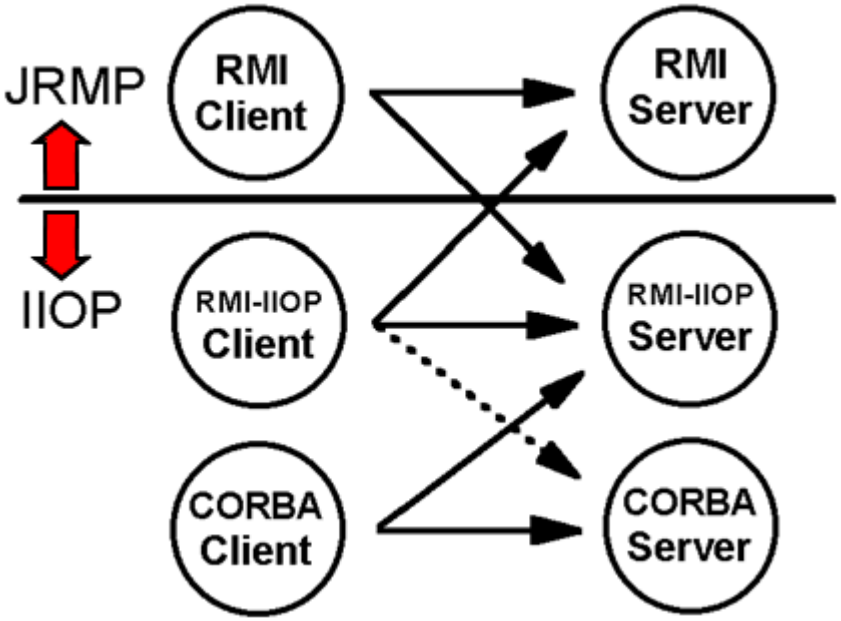



**Wenn Java die einzige Programmiersprache ist, die Sie in einem neuen Projekt benötigen, können Sie RMI und JRMP (JRMP in der obigen Abbildung) verwenden.**

**CORBA auf der anderen Seite ermöglicht bei der Programmierung von Anwendungen mit verteilten Objekten die Benutzung unterschiedlicher Programmiersprachen. Verteilte Objekte werden nicht nur in neuen Projekten benutzt. Sie treten auch bei der Erweiterung älterer Software-Ressourcen (Legacy Software) auf. Legacy-Software ist in den meisten Fällen in anderen Sprachen als Java programmiert. In solchen Situationen müssen die Entwickler das Corba IIOP Protokoll, nicht RMI/JRMP einsetzen.**

**Um die komplexe Corba Programmierung zu vermeiden, können Java Programmierer an Stelle von JRMP das Corba komatible RMI/IIOP Protokoll einsetzen.**

**In der Abbildung stellt der obere Abschnitt der RMI/JRMP Modell, der mittlere Abschnitt der RMI/IIOP Modell, und der untere Abschnitt das CORBA Modell dar. Ein Pfeil stellt eine Situation dar, in der ein Client einen Server anruft. RMI/IIOP gehört in die IIOP Welt unterhalb der horizontalen Linie.**




**Interessant sind die diagonalen Pfeile, die die Grenze zwischen der JRMP Welt und der IIOP Welt überschreiten. Danach kann ein RMI/JRMP Client auf einen RMI/IIOP-Server zugreifen, und umgekehrt. RMI/IIOP unterstützt sowohl JRMP als auch IIOP Protokolle.**

**Eine Server Java Binary (d.h. eine Class File), die mit RMI/IIOP APIs erstellt wurde, kann entweder als JRMP oder als IIOP exportiert werden. Beim Wechsel von JRMP nach IIOP, oder umgekehrt, ist es nicht erforderlich, den Java-Quellcode umzuschreiben oder neu zukompilieren. Es ist nur erforderlich, Parameter wie Java System Properties zu ändern. Alternativ können Sie das zu benutzende Protokoll bestimmen, indem Sie es in dem Java-Quellcode spezifizieren. Die gleiche Flexibilität gilt auch für den RMI/IIOP Client Code.**

**Die diagonalen Pfeile in der Abbildung oben sind möglich, weil die RMI/IIOP APIs sowohl JRMP als auch IIOP Protokolle unterstützen. Dies bedeutet, dass ein RMI/JRMP Server Objekt ohne Umschreiben des Quellcodes durch einen neuen RMI/IIOP Client aufgerufen werden kann. Ebenso kann ein RMI/JRMP Server Objekt durch ein neues RMI/IIOP Objekt ersetzt werden, ohne Umschreiben des Quellcodes eines RMI/JRMP Clients.**

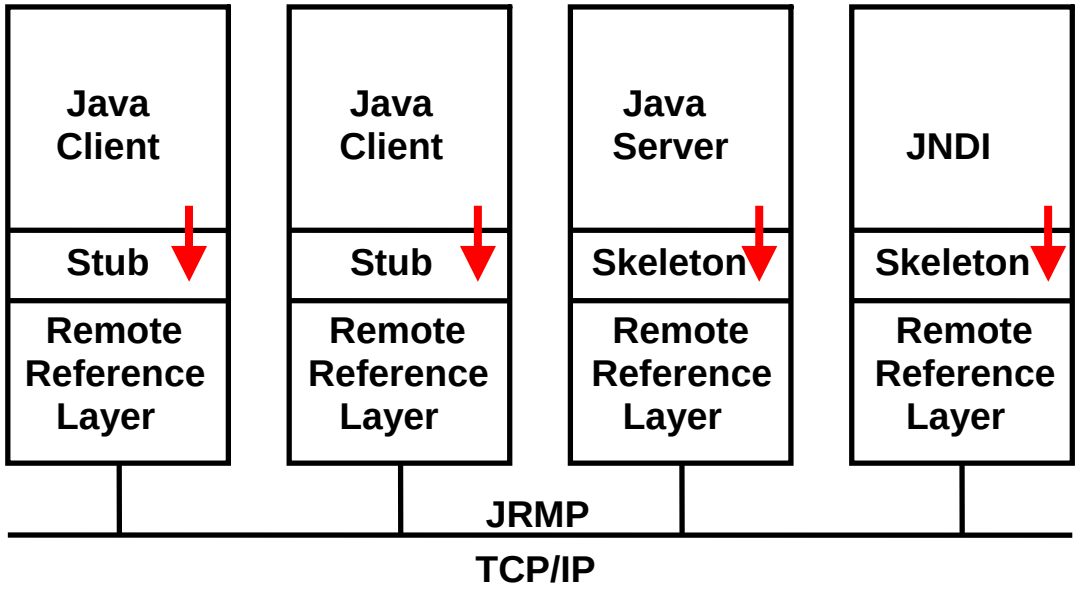
# Interoperability mit CORBA

**Schauen wir wieder auf die obige Abbildung. Der Abschnitt unterhalb der horizontalen Linie ist die IIOP Welt, wo ein RMI/IIOP Client einen CORBA-Server aufruft und ein CORBA Client einen RMI/IIOP Server aufruft. Mit RMI/IIOP Client meinen wir ein Client-Programm, das von einem RMI-Programmierer, der nichts über CORBA oder IDL weis, geschrieben wurde. Ebenso ist ein CORBA-Client ein Client-Programm, das von einem CORBA-Programmierer ohne RMI Kenntnisse geschrieben wurde. Die Trennung der Interface von der Implementierung ist eine gut etablierte Technik. Sie ermöglicht es einem Programmierer auf verschiedene Ressourcen zuzugreifen ohne Wissen wie diese implementiert wurden. Benutzer sowohl von RMI/IIOP als auch von CORBA können die Dienste des anderen Protokolls verwenden , wenn sie Zugang zu seiner Interface bekommen. Ein RMI Java Interface-Datei ist die Interface für RMI/IIOP Benutzer, während IDL die Interface für CORBA Benutzer ist. Die Interoperabilität zwischen RMI/IIOP und CORBA in der obigen Abbildung wird erreicht, indem jedem Benutzer seine erwartete Interface zur Verfügung gestellt wird.**

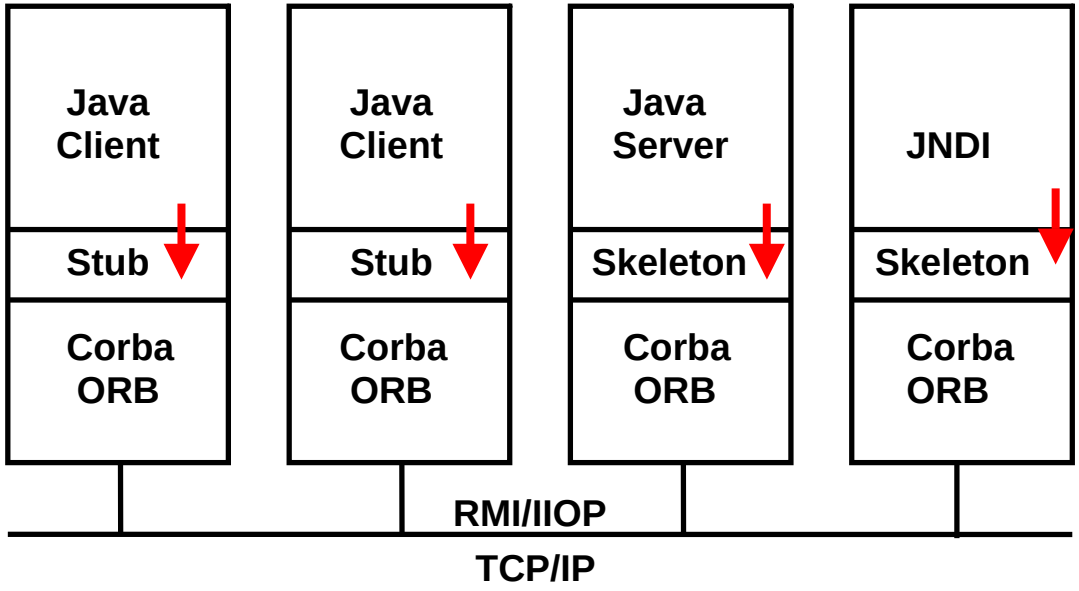
**Als letztes Detail in der obigen Abbildung ist der gepunktete Pfeil zu erklären. Er zeigt einen RMI/IIOP Client, der einen CORBA Server aufruft. Warum ist dieser Pfeil gepunktet ? Ein RMI/IIOP Client kann nicht unbedingt auf alle vorhandenen CORBA Objekte zugreifen. Die Semantik der in IDL definierten CORBA-Objekte kann Funktionen enthalten, die in RMI/IIOP Objekten nicht vorhanden sind. Eine bestehende CORBA Objekt IDL kann nicht immer in einem RMI/IIOP Java-Interface abgebildet werden. Der gepunktete Pfeil zeigt, dass eine Verbindung zu einem bestehende CORBA-Server-Objekt manchmal - aber nicht immer - möglich ist.**

**Diese Einschränkung gilt nicht für neu entwickelte Nicht-Java-CORBA-Server-Objekte (zum Beispiel geschrieben in C/C++). Es ist einfach, ein RMI/ IIOP Interface hierfür zu erzeugen.**

**Um zusammenzufassen: Wenn Sie Ihren Server in RMI/IIOP implementieren, haben Sie die größte Auswahl an Klienten. Ebenso, wenn Sie Ihren Klienten in RMI/IIOP implementieren, können Sie mit der größten Auswahl an Servern kommunizieren. Hierbei kann es einige Beschränkungen im Fall von bestehenden CORBA-Objekte geben.**

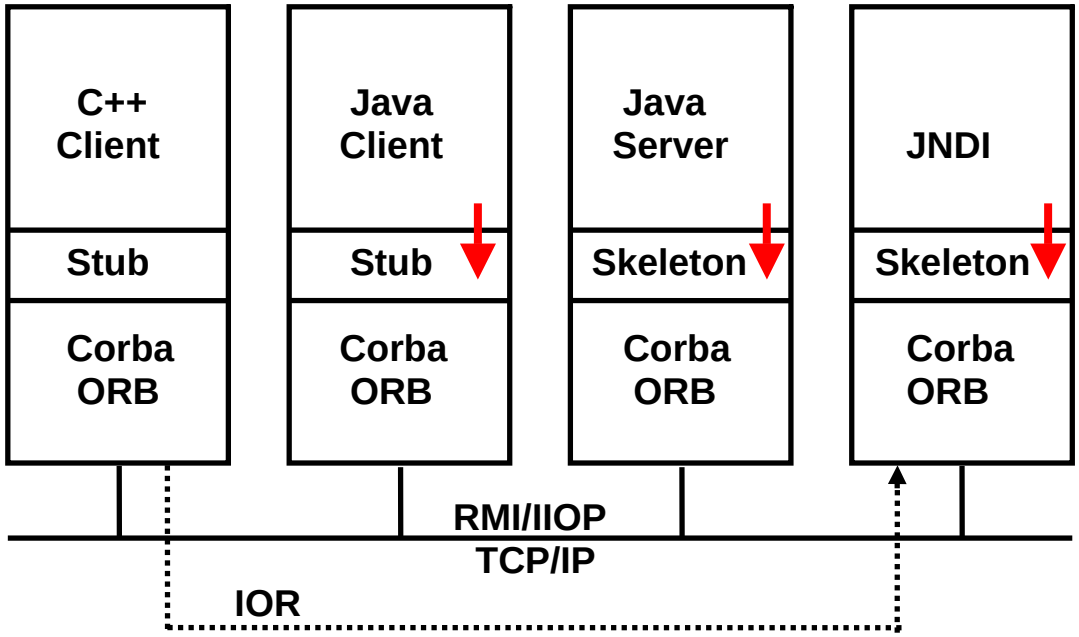



Java Client und Server benutzen die RMI interface ↓ um mit Stub und Skeleton zu kommunizieren. Der JDK enthält die "Remote Reference Layer" Komponente, welche alle Communication Funktionen implementiert. Es stellt auch die Stub und Skeleton Interfacen mit TCP/IP in OSI layer 4 zur Verfügung.




Bei der Benutzung von RMI/IIOP wird die Remote Reference Layer durch den Corba Object Request Broker (ORB) ersetzt.

Die RMI Interface, ↓ die vom Client und Server Object Code für die Kommunikation mit Skeleton und Stub benutzt wird, verändert sich nicht.




**Wenn ein Corba Object Request Broker (ORB) für alle Java Komponenten benutzt wird, kann ein non-Java Member der Gruppe beitreten (ein C++ Klient in in diesem Beispiel.**

**Die RMI Interface, ↓ die vom Client und Server Object Code für die Kommunikation mit Skeleton und Stub benutzt wird, verändert sich nicht.**

**Der vorhandene Java JEE Standard legt fest, dass alle Java Software Produkte RMI/IIOP unterstützen. Die Unterstützung für JRMP ist optional.**

**In Übereinstimmung mit dieser Spezifikation unterstützen IBM WebSphere und CICS unter z/OS nur RMI/IIOP. Ihre EJB Container haben die Funktionalität eines CORBA ORB.**

**Interessanterweise ist die Performance von RMI/IIOP deutlich besser als die von JRMP.**

# JRMP Entwicklung

**In der folgenden Abbildung unten ist die die Entwicklung für den JRMP Server und den JRMP Klient en dargestellt.. Folgen Sie den roten Pfeilen:**

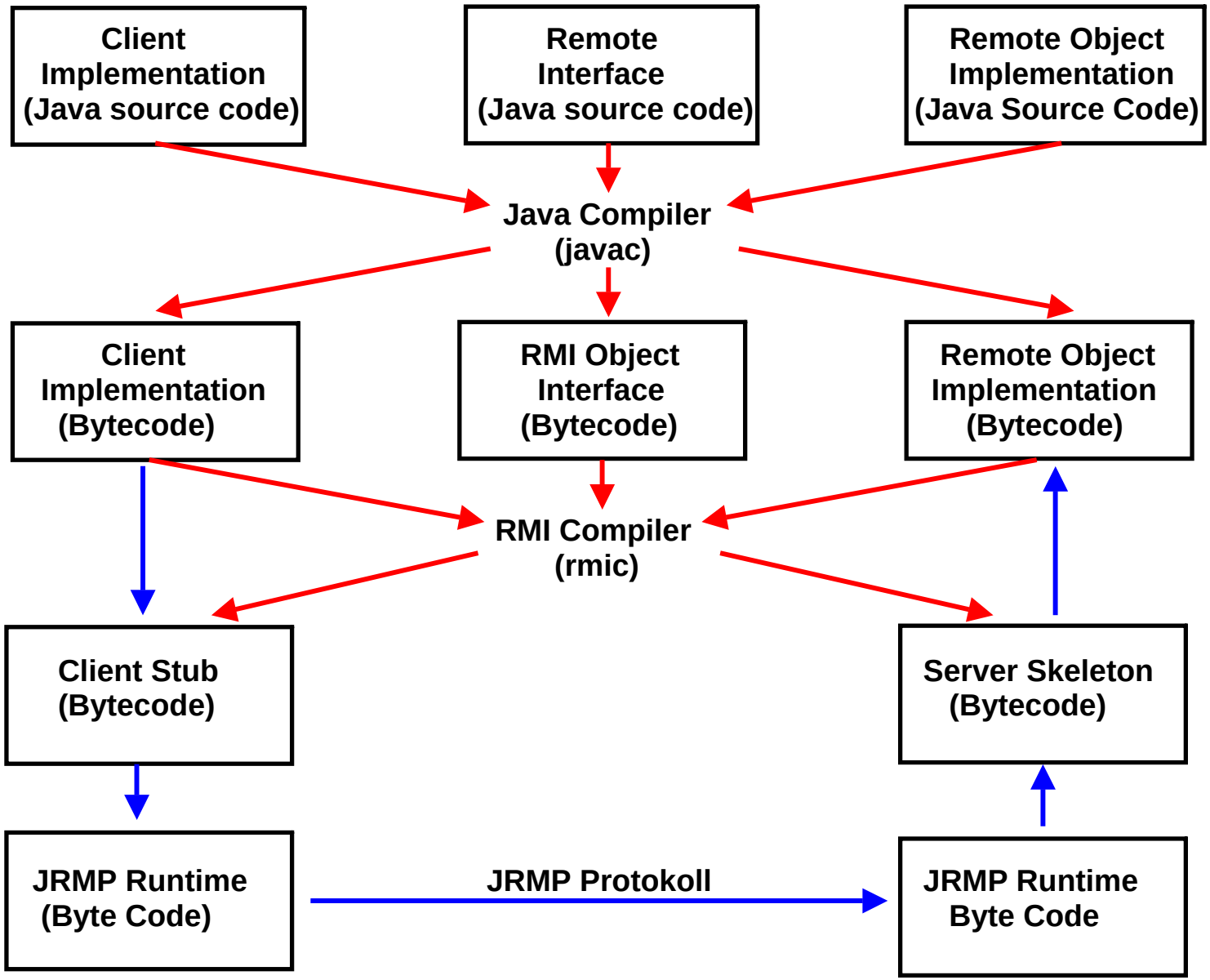
**Der Quellcode der Server-side Object Implementierung und deren Java-Interface werden wie gewohnt mit dem regulären Java-Compiler (`javac`) übersetzt. Dies generiert Byte-Code.**

**Der Byte-Code wird als Eingabe für den Java RMI-Compiler (`rmic`) verwendet, zusammen mit der Remote-Interface Beschreibung des Java-Server-Objekts. Hiermit wird eine Byte Code Version sowohl des Server Skeletons als auch des Client-Stubs generiert.**

# JRMP Ausführung

**Folgen Sie nun den blauen Pfeilen.**

**Wenn der Bytecode der Client-Implementierung das Remote-Objekt auruft, überträgt es eine eine Nachricht an den Client-Stub. Die Client-Stub-Klassen übernehmen die Funktionen, die unique für diese Client-Implementierung sind. Die JRMP Laufzeitklassen führen die generischen JRMP Funktionen aus und starten die Kommunikation durch die Übertragung eines JRMP Nachricht an die TCP-Komponente des Client-Systems.**


Client Implementation (Java source code)
Remote Interface (Java source code)
Remote Object Implementation (Java Source Code)
Java Compiler (javac)
Client Implementation (Bytecode)
RMI Object Interface (Bytecode)
Remote Object Implementation (Bytecode)
RMI Compiler (rmic)
Client Stub (Bytecode)
Server Skeleton (Bytecode)
JRMP Runtime (Byte Code)
JRMP Protokoll
JRMP Runtime Byte Code

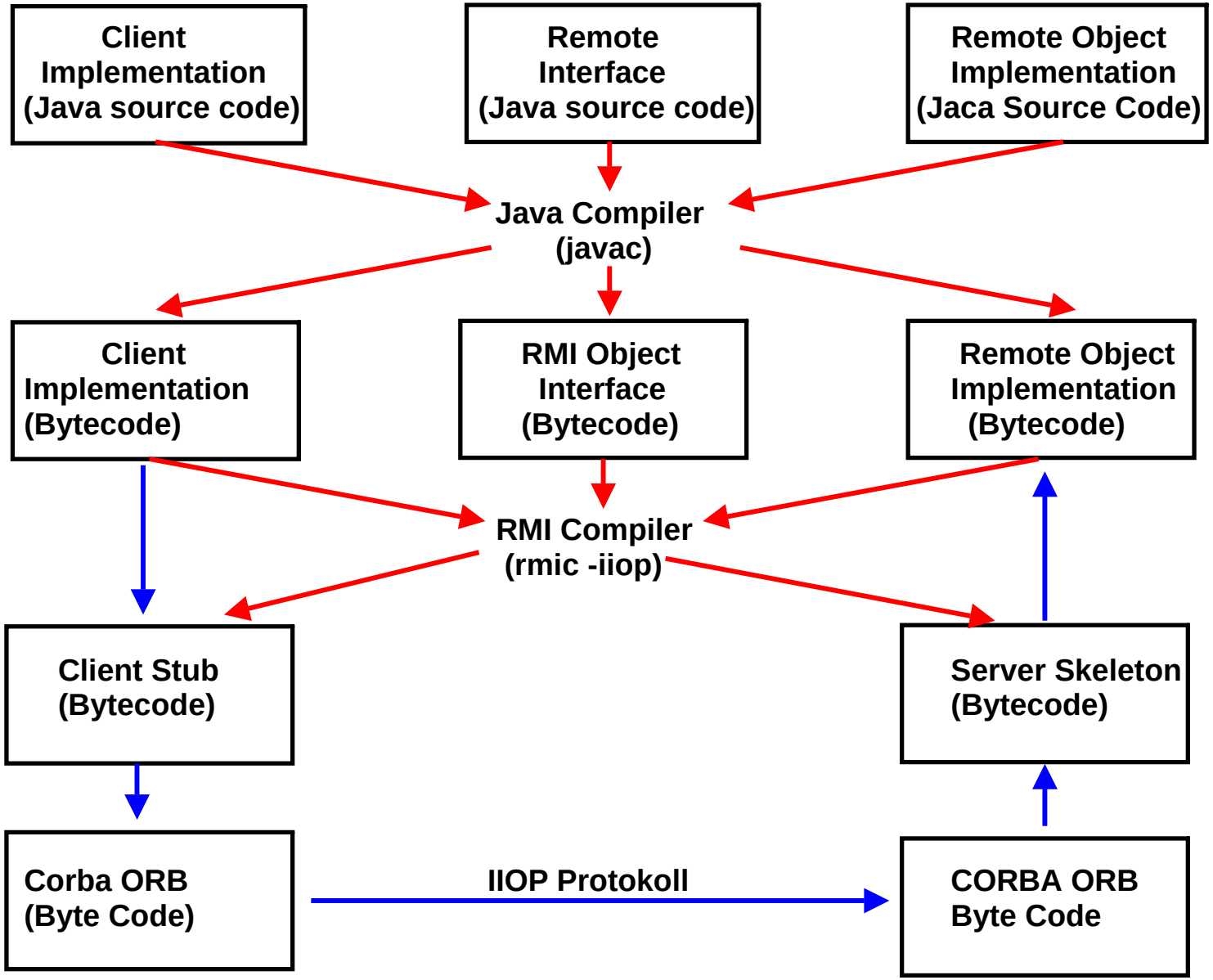
# RMI/IIOP Entwicklung

**Die folgende Abbildung zeigt das Entwicklungs-Verfahren für den RMI/IIOP Server und Klienten. Es entspricht der letzten Abbildung. Sie werden feststellen, es ist fast das gleiche Verfahren wie bei RMI/JRMP.**

**Genauso wie bei RMI/JRMP enthält eine RMI Java-Interface die verteilte Server-Objekt-Definition. Ein Unterschied ist der `iiop` Parameter des `rmic` Compilers. Diese Option wird benutzt, damit `rmic` Stubs erzeugt werden, und diese mit dem IIOP Protokoll anstelle des JRMP Protokolls verknüpft werden. Obwohl die Entwicklungsverfahren für RMI/IIOP fast die gleichen wie für RMI/JRMP sind, ist die Laufzeitumgebung verschieden. Die Kommunikation erfolgt über einen CORBA-konformen ORB. IIOP wird für die Kommunikation zwischen Server und Client benutzt.**

**RMI/IIOP verwendet den Java CORBA Object Request Broker (ORB) und IIOP. Sie können den gesamten Code in der Programmiersprache Java schreiben, und den `rmic` Compiler (mit der `iiop` Option) benutzen. Die resultierende Java Anwendung kann über das Internet InterORB Protocol (IIOP) mit anderen in einem CORBA-kompatiblen Sprache geschrieben Anwendungen kommunizieren.**

**Der J2EE-Standard bietet zwei alternative Protokolle: JRMP und RMI/IIOP. Einige Hersteller unterstützen beide RMI Protokolle. IBM hat beschlossen, ausschließlich RMI/IIOP in ihren Produkten zu verwenden. Dies ist konform mit dem RMI-Standard.**


Client Implementation (Java source code)
Remote Interface (Java source code)
Remote Object Implementation (Jaca Source Code)
Java Compiler (javac)
Client Implementation (Bytecode)
RMI Object Interface (Bytecode)
Remote Object Implementation (Bytecode)
RMI Compiler (rmic -iiop)
Client Stub (Bytecode)
Server Skeleton (Bytecode)
Corba ORB (Byte Code)
IIOP Protokoll
CORBA ORB Byte Code

# RMI/IIOP und CORBA

**CORBA und RMI/IIOP verwenden den gleichen Internet Inter-ORB Protocol Kommunikationsstandard. Wenn erforderlich ist es möglich, die IDL-Definitionen für die beteiligten RMI/IIOP Datenstrukturen erzeugen. Diese Definitionen können benutzt werden, um die Interoperabilität zwischen den RMI/IIOP Anwendungen und normalen CORBA-Anwendungen zu gewährleisten.**

**Mit RMI/IIOP können Entwickler Remote Interfaces in der Java-Programmiersprache schreiben und nur mit Hilfe von Java-Technologie und den Java RMI APIs implementieren. Diese Interfaces können in jeder anderen von CORBA unterstützten Sprache implementiert werden, vorausgesetzt es existiert ein ORB für diese Sprache. Ebenso können Klienten, die in anderen Sprachen geschrieben sind, IDL benutzen, die von den Remote-Java-Technologie basierenden Interfaces abgeleitet ist.**

**Verwenden Sie den RMI/IIOP Compiler mit der `iiop` Option un Stubs und Skeletons für Remote-Objekte zu erzeugen, welche das IIOP-Protokoll verwenden. Der `rmic` Compiler kann auch benutzt werden, um IDL für CORBA Entwicklungen auf der Java-Plattform zu erzeugen.**

# Coexistence Beispiel: C++ client ruft ein Java RMI Object auf

**Als Beispiel betrachten wir den Fall, wo ein C++ CORBA Client ein RMI-Server-Objekt aufruft**

**Der `rmic-iiop` Compiler generiert Stubs und Skeletons für Java-Clients und Server.**

**Die C++ CORBA Client benötigt einen C++ CORBA IIOP Stub um das RMI-Server Objekt aufzurufen. Stubs werden mit Hilfe der Interface Definition des Server-Objektes erzeugt. Hierzu brauchen wir eine normale Corba IDL Beschreibung der Server Interface. Die Beschreibung der Server Interface liegt aber als Java Interface, und nicht als Corba IDL Interface vor.**

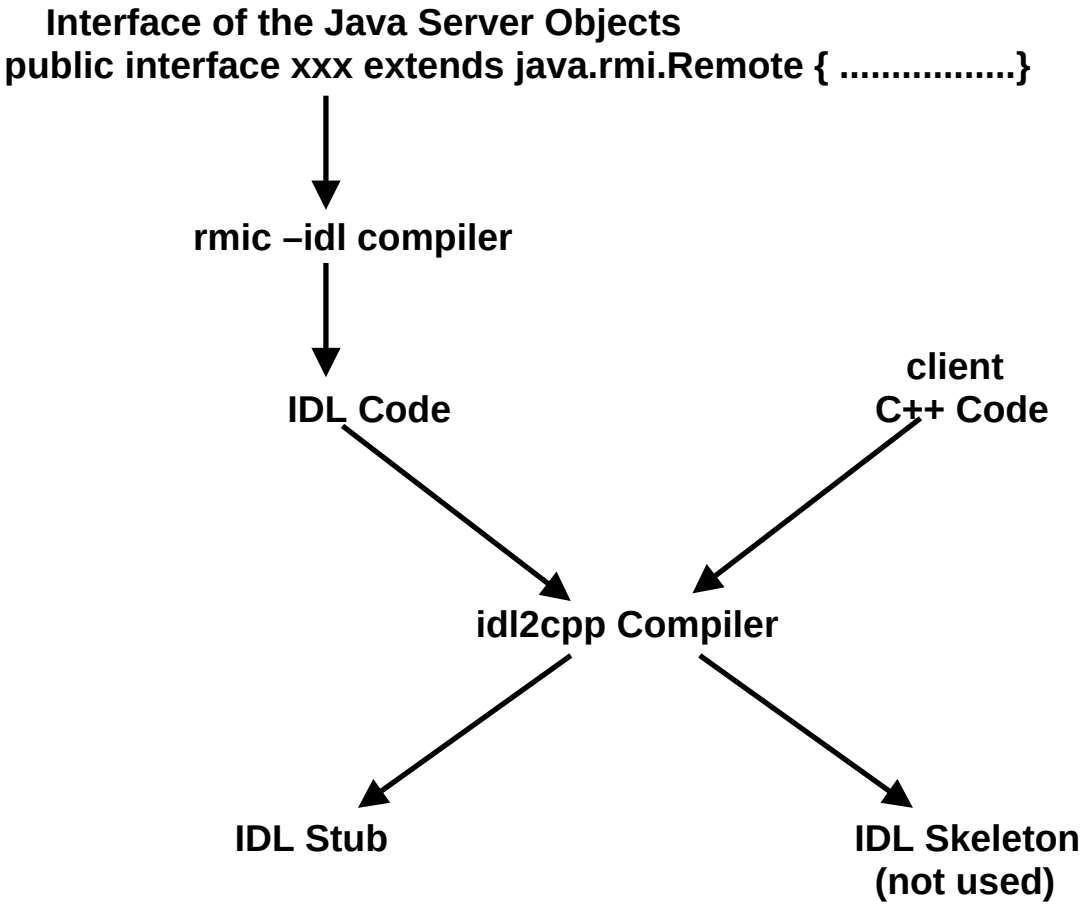
**Um den Client-Stub zu generieren, beginnen wir mit dem Java-Server-Interface.**


**Wie in der folgenden Abbildung dargestellt, verwenden wir die Java-Interface und den `rmic` Compiler mit der `idl` Option um eine IDL-Beschreibung der Interface (IDL-Datei) zu generieren.**

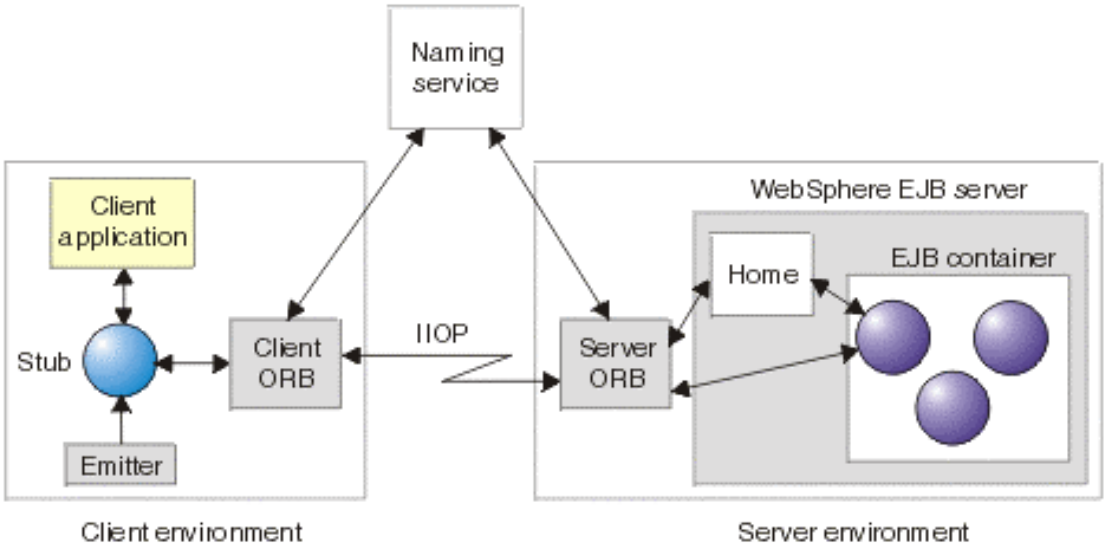
**Mit dem IDL Beschreibung der Interface können wir jetzt den Client-Stub generieren.**

**Die IDL-Beschreibung, zusammen mit dem Client C++-Code wird von dem regulären CORBA-Compiler (der idl2cpp Compiler) verwendet, um den Code für CORBA C++ kompatible Stubs und Skeletons zu erzeugen. Der Stub wird mit dem C++-Client integriert, das Skelett wird verworfen, da das Java RMI-Objekt das regulären Skeleton verwendet, das durch `die rmic-iiop` Compiler generiert wurde.**

# Generating the Client Stub Code




**Der idl2cpp Compiler wird von CORBA standardmäßig benutzt, um für ein C++ Objelt Skeleton und Stub zu erzeugen. Ähnliche Comiler existieren, um für Cobol, ADA, PL/1 Objekte Skeleton und Stub zu erzeugen.**




# WebSphere CORBA Support

**Die IBM WebSphere Web Application Server beinhaltet CORBA-Unterstützung. Diese ermöglicht den Einsatz von CORBA-Interfaces zwischen einem Server-Objekt, das einen Service anbietet, und einem Client, der diesen Service benutzt. In der Praxis bedeutet dies:**

- **WebSphere C++ CORBA-Server und WebSphere EJB-Services können von CORBA-Clienten aufgerufen werden, und**
- **WebSphere CORBA-Clienten können auf CORBA-Server zugreifen.**

**Als Teil der WebSphere JEE-Umgebung bietet die C++ CORBA-Unterstützung eine Basis CORBA Umgebung an. Diese kann in den JEE Namensraum integriert werden, und kann J2EE-Transaktionen aufrufen.**