

Client/Server-Systeme

Prof. Dr.-Ing. Wilhelm Spruth

SS 2006

Teil 14

CORBA, RMI

Wiederverwendbarkeit von Code

Vorbild: Entwurf/Bau einer Brücke im Bauingenieurwesen

Objekttechnologie ermöglicht Code Blöcke mit fest definierter Funktionalität, die in Klassen gekapselt werden. Ein Objekt als Instanz einer Klasse stellt sich dem Entwickler als Black Box mit einer öffentlichen Schnittstelle dar.

Wird nur in einer einzigen Sprache mit identischem Compiler entwickelt ist die Wiederverwendbarkeit von Klassen am leichtesten erreichbar. Beim Einsatz mehrerer Programmiersprachen muß die Objektphilosophie der Programmiersprache beachtet werden. Z.B. unterstützt C++ die Mehrfachvererbung, Java aber nur die Einfach-vererbung.

Um objektorientierte Bibliotheken mit unterschiedlichen Sprachen und Compilern zu realisieren, ergeben sich eine Reihe von Problemereichen:

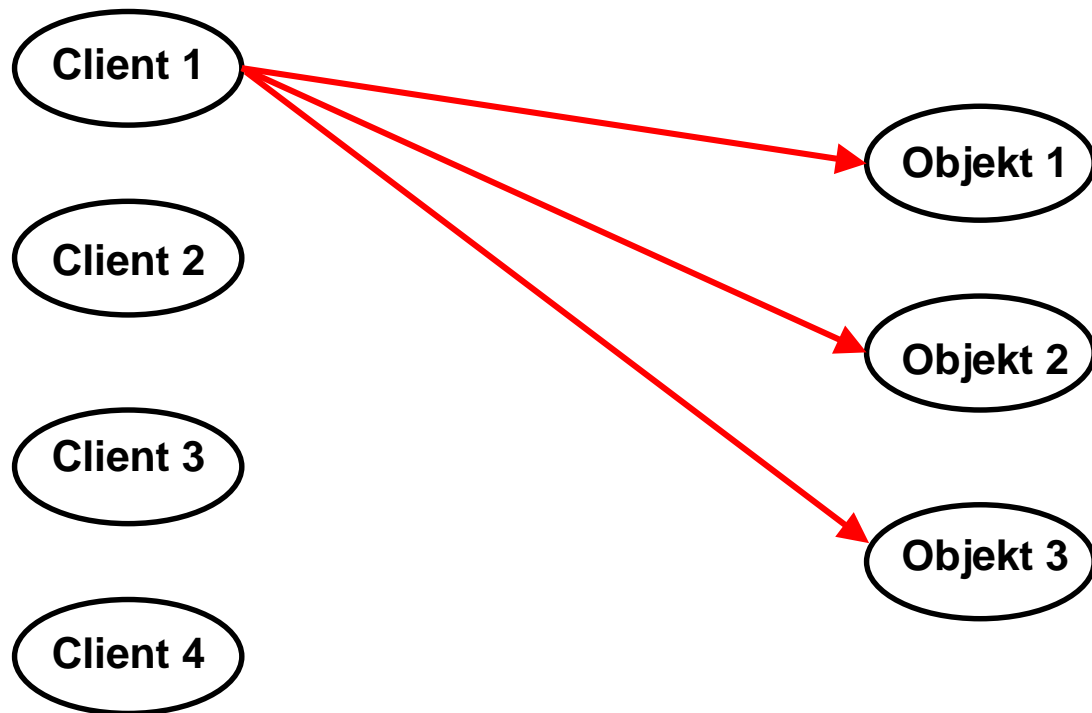
- Sprachunabhängigkeit: Smalltalk- und C++-Objekte verstehen einander nicht
- Compilerunabhängigkeit: Objekte zweier Compiler (z.B. *Watcom* und *GNU C++*) können nicht miteinander kommunizieren, weil die Verwaltung interner Informationen nicht standardisiert ist
- starre Kopplung zwischen Objekten: Wenn sich die Implementierung einer Klasse ändert, müssen alle Teile, die diese Klasse in irgendeiner Form nutzen, neu kompiliert werden. Beispiel: C++ Compiler benutzen Konstanten beim Zugriff auf Daten und Methoden
- Beschränkung auf einen Prozeßraum (Objekte können nicht über Prozeßgrenzen hinweg kommunizieren)

Zielsetzung: Unterschiedliche objektorientierte Klienten-Implementierungen, die auf unterschiedlichen Plattformen (z.B. HP-UX, AIX, Solaris, Linux, NT, OS/400, OS/390) laufen, sollen mit unterschiedlichen objektorientierten Server-Implementierungen (ebenfalls unterschiedliche Plattformen) in Binärform kompatibel zusammen arbeiten.

Klienten sollen maximal portierbar sein, und ohne -Änderungen auf Rechnern/Betriebssystemen unterschiedlicher Hersteller lauffähig sein.

Klienten sollen keine Kenntnis benötigen wie ein Objekt auf einem Server implementiert ist

Anforderung (Request)

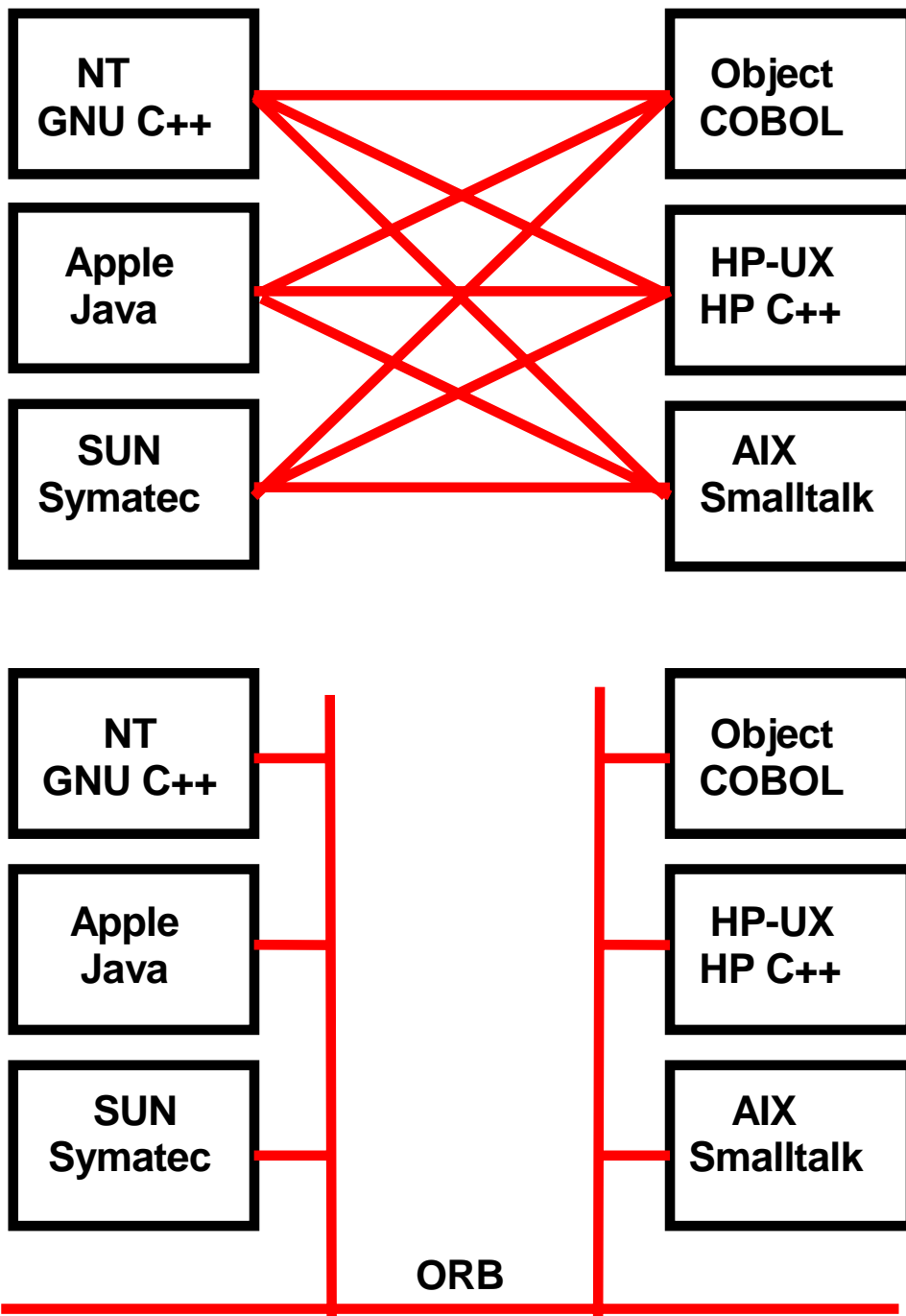


Objekte

bieten Operationen an, durch deren Aufruf sie zu bestimmten Aktionen veranlaßt werden können

Klient

Eine Software-Einheit, die eine Operation eines Objektes aufruft



In unterschiedlichen Programmiersprachen entwickelte binäre Objekte können plattformunabhängig miteinander kommunizieren.

Object Request Broker ORB

Ein ORB (Object Request Broker) ist eine Komponente eines Betriebssystems. Er ermöglicht es, daß Objekte, die in unterschiedlichen Programmiersprachen entwickelt wurden, in binärer Form miteinander zusammenarbeiten. Dies kann über Prozeß- und physische Rechengrenzen hinweg geschehen.

Hierfür stellt der ORB ein einheitliches Klassenmodell sowie Standards für die Organisation von Klassenbibliotheken und die Kommunikation zwischen Objekten zur Verfügung.

Es existieren mehrere Alternativen einen ORB zu implementieren:

- **CORBA Standard der OMG**
- **Remote Method Invocation (RMI),
 Teil des JDK 1.1 der Fa. SUN**
- **COM+ / DCOM / Activex / DNA / DotNet der Fa. Microsoft**

**In eine ähnliche Richtung geht die Internet orientierte
SOAP / UDDI / WSDL Initiative.**

CORBA

Common Object Broker Architecture

Standard der OMG (Open Management Group)

Unterstützt viele Sprachen, darunter C/C++, COBOL und Java.

Implementiert von zahlreichen Herstellern

Begriffe

Object Management Group (OMG)

1989 gegründeter, internationaler, nicht profit-orientierter Zusammenschluß von zunächst 8 (Anfang 1996: Ca. 600) Softwareentwicklern, Netzbetreibern, Hardwareproduzenten und kommerziellen Anwendern von Computersystemen (*ohne Microsoft*).

Object Management Architecture (OMA)

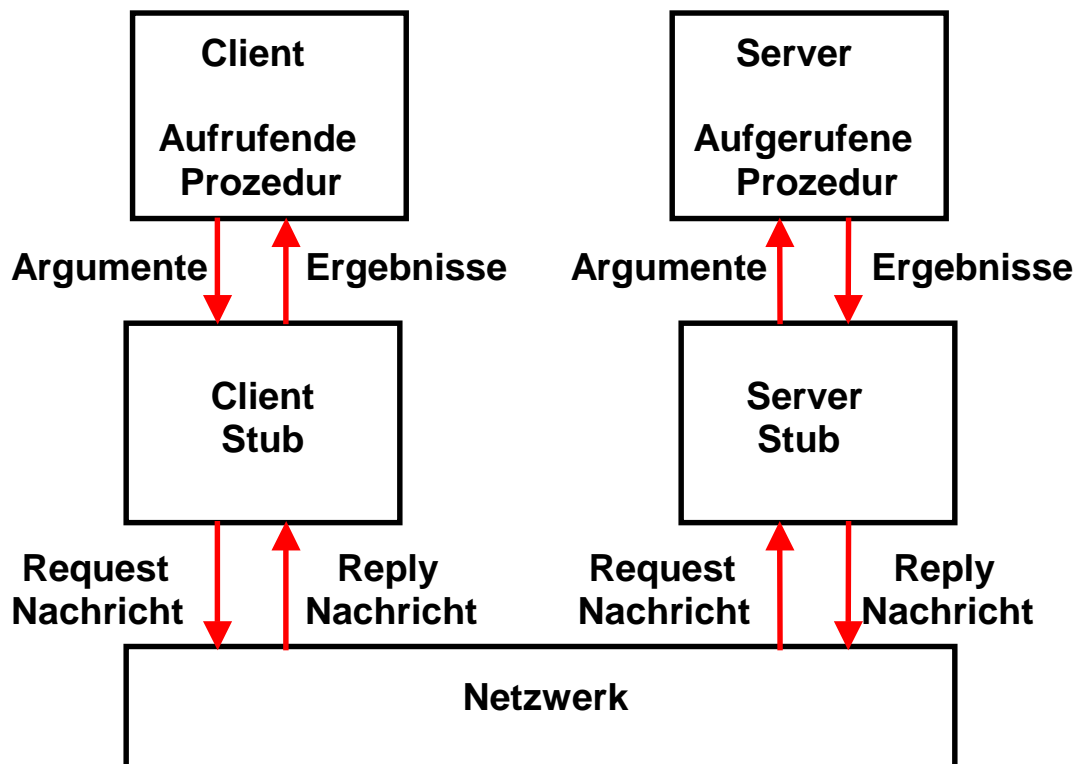
Von OMG spezifizierte Architektur, die das Zusammenwirken von Anwendungen verschiedener Hersteller unabhängig von Betriebssystem, Programmiersprache und Hardware ermöglichen soll.

Common ORB Architecture (CORBA)

Universelles Kommunikationsmedium für beliebig geartete Objekte in verteilten heterogenen Systemen, Kernstück der OMA, 1991 von OMG in Version 1.1 spezifiziert, aktuelle Version 3.0 (Sommer 1999).

Literatur

R. Orfali, D. Harkey: „Client/Server Programming with Java and Corba“. John Wiley, 2nd edition, 1998.



Remote Procedure Call

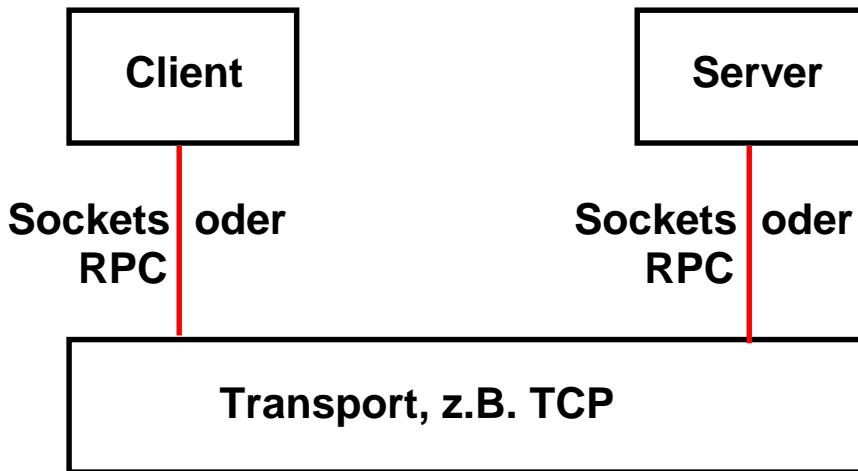
Client und Server laufen als zwei separate Prozesse.

Die beiden Prozesse kommunizieren über Stubs. Stubs sind Routinen, welche Prozedur Aufrufe auf Netzwerk RPC Funktionsaufrufe abbilden.

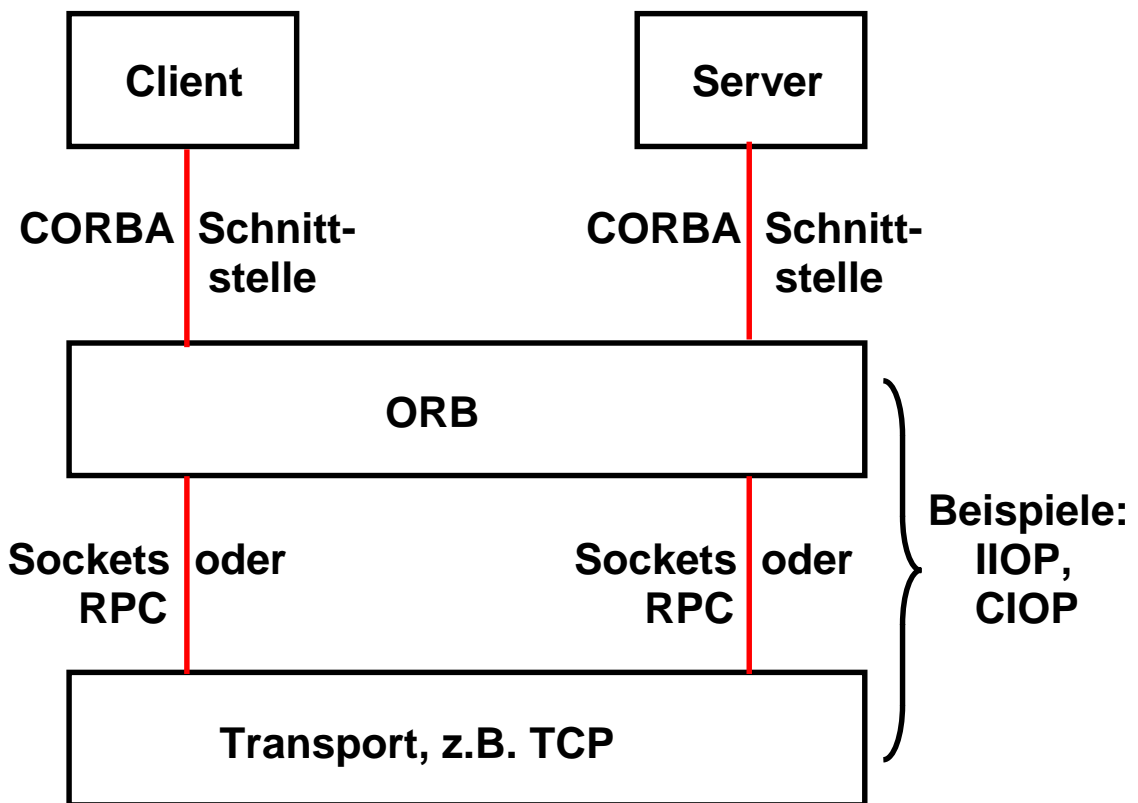
Ein Server-seitiges RPC Programm definiert seine Interface (Schnittstelle) unter Benutzung einer [Interface Definition Language \(IDL\)](#).

Ein Stub Compiler liest die IDL Schnittstellenbeschreibung und produziert zwei [Stub Prozeduren](#) für jede Server Procedure (client side stub und server side stub).

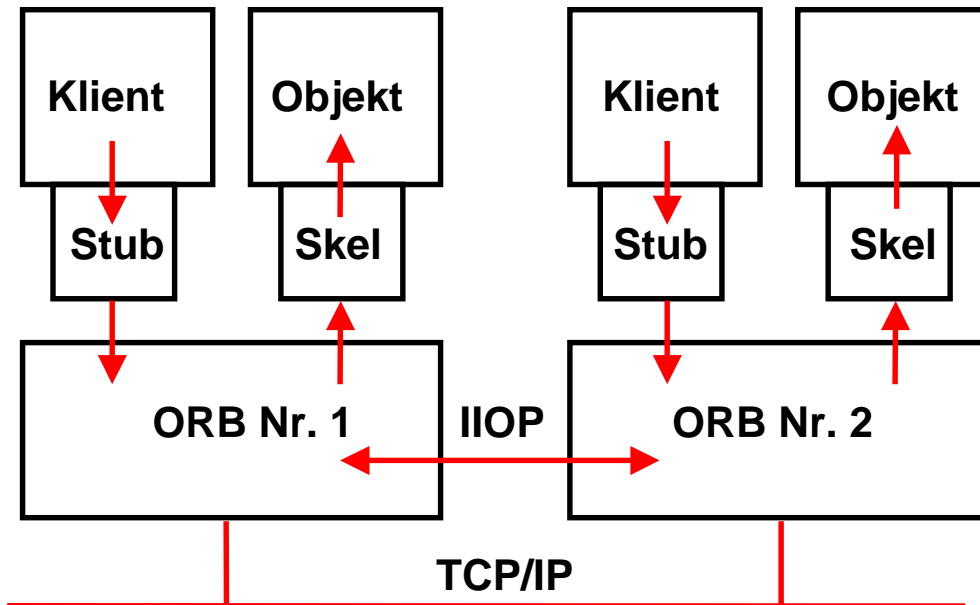
Beim Sun RPC wird die Interface Definition Language als RPC Language oder XDR Language bezeichnet.



Mit Prozeduren arbeitendes Client/Server-System



Objektorientiertes Client/Server-System



ORB stellt eine Schnittstelle zwischen Client und Server dar, die ähnlich der Socket- oder RPC-Schnittstelle arbeitet, aber

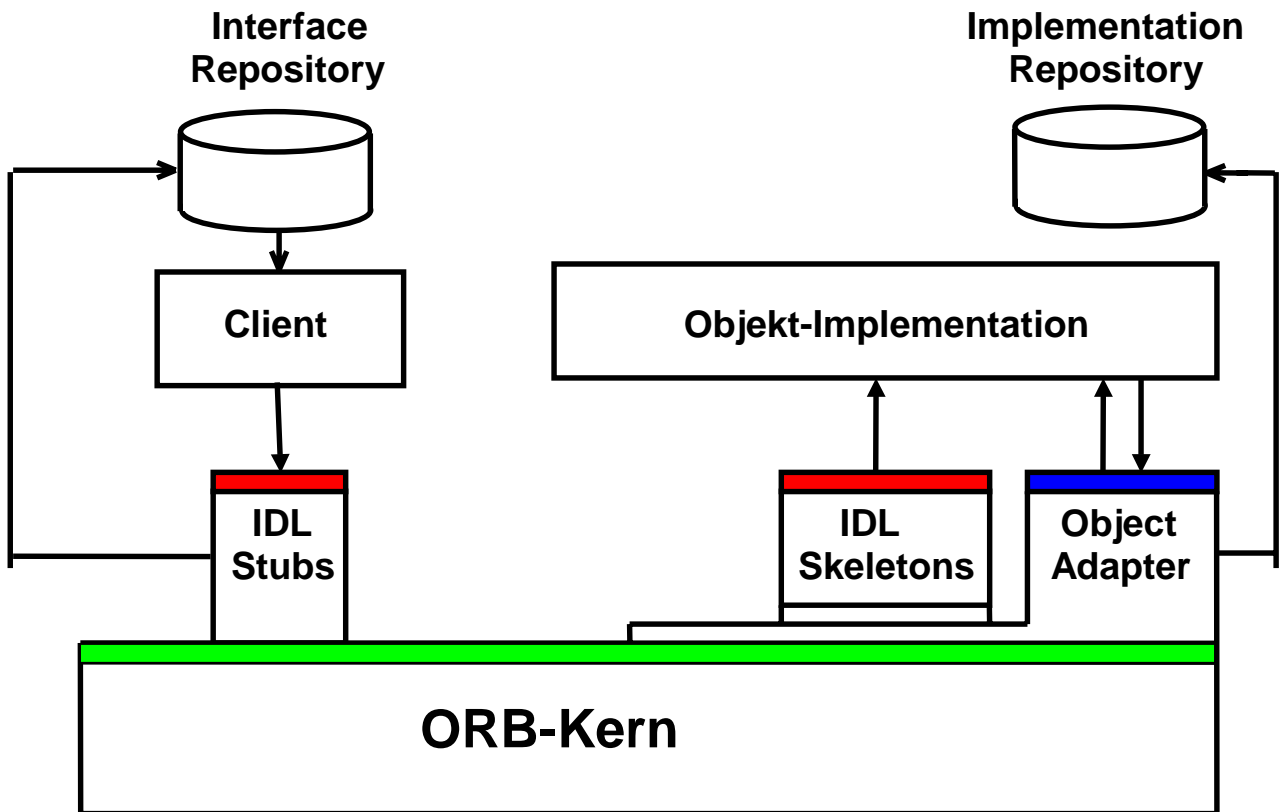
- auf einer höheren Ebene arbeitet (und deshalb evtl. Sockets oder RPC für die eigentliche Kommunikation verwendet)
- objektorientiert arbeitet
- für die Kommunikation das IOP Protokoll verwendet (andere)





Klienten und Server Objekte kommunizieren mit ihrem ORB über Stubs und Skeletons. Sie können transparent auf dem gleichen oder auf entfernten Rechnern arbeiten

Eine eindeutige Objekt Referenz (IOR) wird dem Objekt bei der Entstehung zugeordnet. Der Client nimmt die Dienste eines Objektes in Anspruch, indem er einen Methodenaufwurf ausführt. Der Methodenaufwurf enthält:

- IOR
- Methodennamen
- Parameter
- Kontext (enthält Information über die Position des Aufrufers und nimmt Fehlermeldungen entgegen)

Wichtige Schnittstellen des ORB zum Client und zur Objekt-Implementation



-  Standardisierte Schnittstelle (identisch für alle ORBs)
-  Mehrere, jeweils auf einen Objekttyp spezialisierte Schnittstellen
-  Es kann mehrere Object Adapter geben
-  ORB-spezifische Schnittstelle

IDL Stubs
IDL Skeletons } }

Objektspezifische Schnittstellen, werden von IDL-Compiler generiert

Implementation Repository

Das Implementation Repository ist Bestandteil des ORBs. Hier befindet sich Run-Time Informationen, die der ORB benötigt, um den Aufenthaltsort einer Objekt-Implementation festzustellen. Der Request eines Client enthält zwar eine Objekt Referenz, diese kann jedoch nicht den physischen Aufenthaltsort eines Objektes beinhalten. Diese Information erhält die Objekt Referenz erst durch das Implementation Repository.

Der Vorteil dieser Trennung ist, dass einem Administrator vollkommen freie Hand gelassen wird, wo er die Objekte vorhält. Es ist z.B. möglich einen Server während des Betriebs zu wechseln. Dazu muss nur im Repository ein neuer Aufenthaltsort des Objektes eingetragen werden und nach Beendigung der aktuellen Aktivitäten kann der alte Server ausgeschaltet werden.

Das Implementation Repository enthält Information über die Klassen, die ein Server unterstützt, die aktivierten Objekte und deren ID's. Speichert zusätzliche administrative Daten wie Trace Information, Audit Trails und relevante Daten für die Sicherheit.

Kann auch mehrere Lokationen für ein und dasselbe Objekt (d.h. dieselbe Referenz) bereitstellen: Etwa zur Lastverteilung auf mehrere Server oder als Vorkehrung für den Ausfall eines Servers. Für den Client stellen sich dabei sämtliche Objekt-Implementierungen als ein einziges Objekt dar.

Enthält weiterhin die Interface-Beschreibungen der vorhandenen Objekte. Wird deshalb auch genutzt, um DII und DSI die erforderlichen Informationen bereitzustellen.

Der genaue Aufbau des Implementation Repository ist ORB-spezifisch gelöst (wird nicht von OMG definiert).

Interface Repository

Run-Time verteilte Datenbank, Bestandteil des ORB.

Enthält Schnittstellen Beschreibungen aller registrierten verteilten Objekte. Dies schließt Beschreibungen der
Methode
Parameter
der Objekte ein.

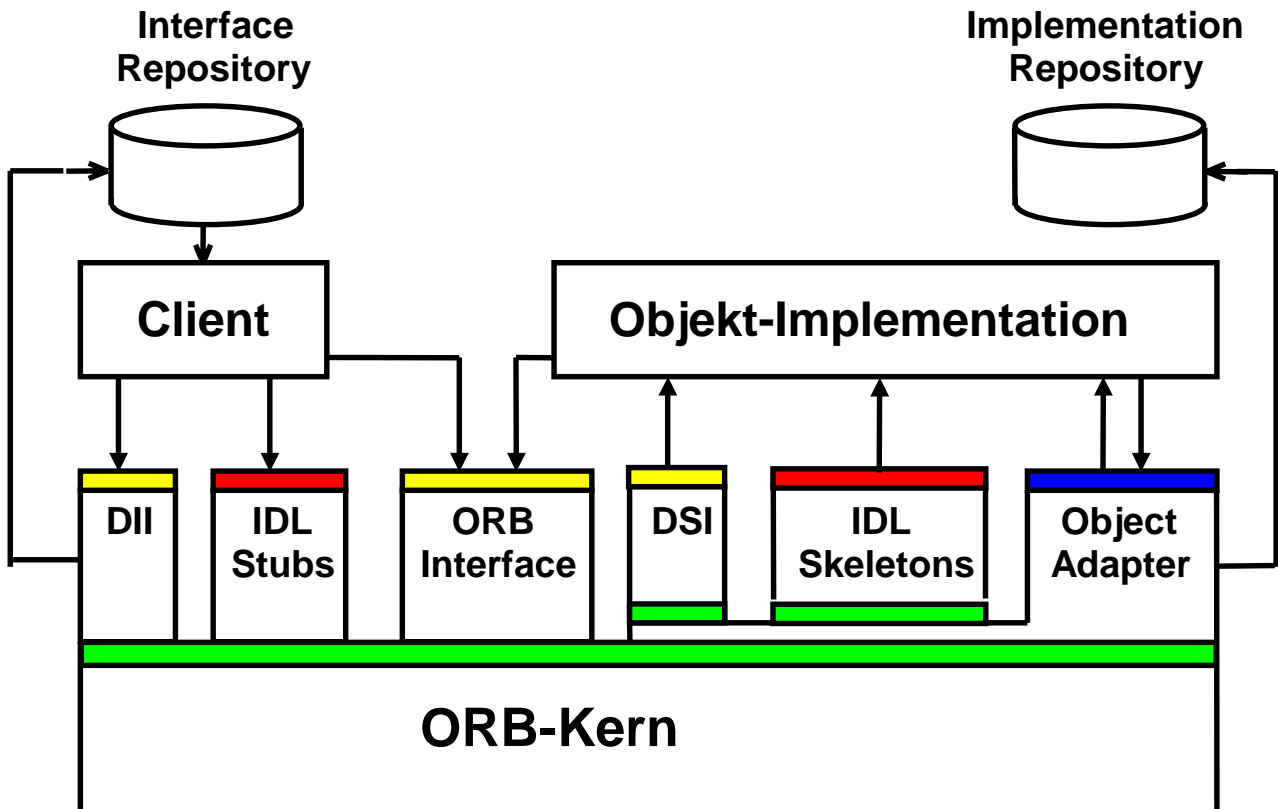
Die Schnittstellen Beschreibungen können als Maschinenlesbare Versionen der in IDL definierten Schnittstellen betrachtet werden. CORBA bezeichnet diese Beschreibungen als „Methoden Signaturen“. Beim Aufruf (Invocation) eines Objektes ermöglicht die Signatur eine Typen Überprüfung. Bei dynamischen Aufrufen für die Erstellung des Klienten Stubs genutzt.

Die Interface Repository Inhalte werden durch den IDL Compiler erstellt.

Ein Klient erhält die Bezeichnung (den Namen) der Interface eines Objektes, indem er die *get_interface* Methode eines Objektes aufruft (Introspection). Diesen Namen kann der Klient benutzen, um aus dem Interface Repository die vollständige Beschreibung der Interface, z.B. die unterstützten Methoden, auszulesen.

Interface Repositories können lokal verwaltet werden, oder als Department- oder unternehmensweite Ressourcen eingesetzt werden.

Schnittstellen des ORB zum Client und zur Objekt-Implementation



- Standardisierte Schnittstelle (identisch für alle ORBs)
- Mehrere, jeweils auf einen Objekttyp spezialisierte Schnittstellen
- Es kann mehrere Object Adapter geben
- ORB-spezifische Schnittstelle

DII: Dynamic Invocation Interface
 DSI: Dynamic Skeleton Interface

Gehören mit Object Adapter und ORB-Kern zur CORBA-Plattform

IDL Stubs
 IDL Skeletons

Objektspezifische Schnittstellen, werden von IDL-Compiler generiert

Objekt

Menge von Daten, die nur über wohldefinierte Operationen (Methoden) zugänglich sind

Objekt-Instanz, Objekt-Exemplar

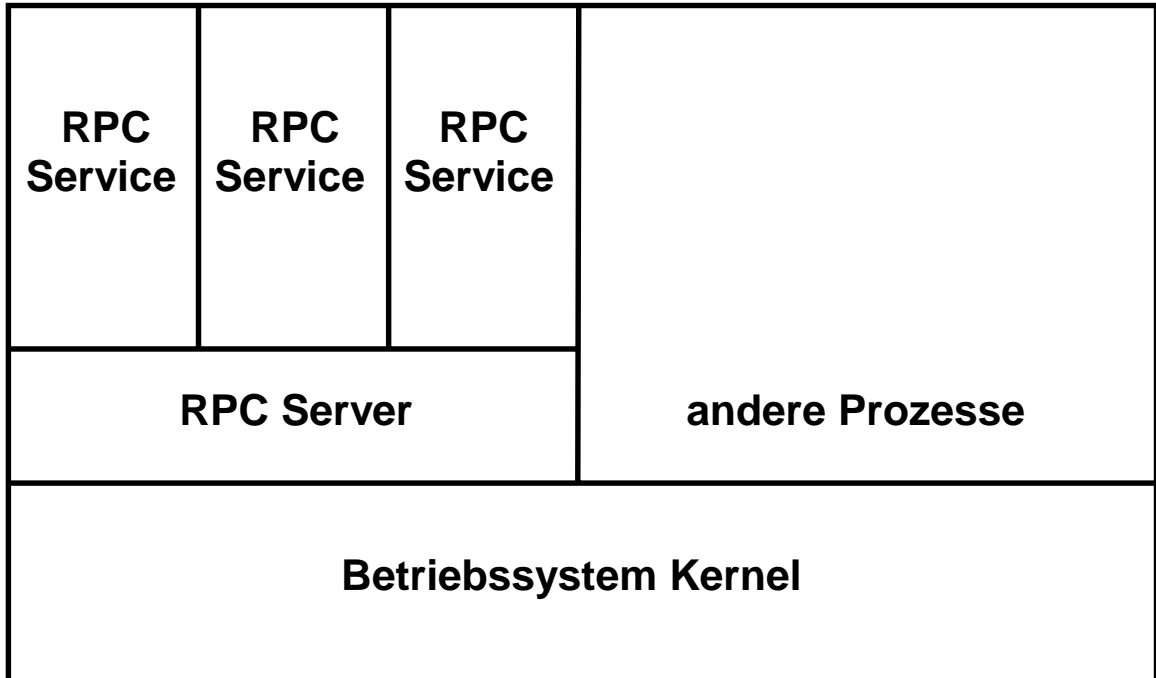
Ausprägung eines Objektes, welches zu einer (vorher definierten) Klasse gehört

Objekt-Implementierung (Objekt Klasse, Objekt-Typ, Servant)

spezifiziert Operationen und Datenstrukturen für gleichartige Objekte

Typ Abstrakte Spezifikation der Funktionalität

Klasse Implementierung dieser Funktionalität



Die RPC Services (RPC Dienstprogramme) können entweder in getrennten virtuellen Adressräumen laufen, oder alternativ als Threads innerhalb eines virtuellen Adressraums implementiert werden.

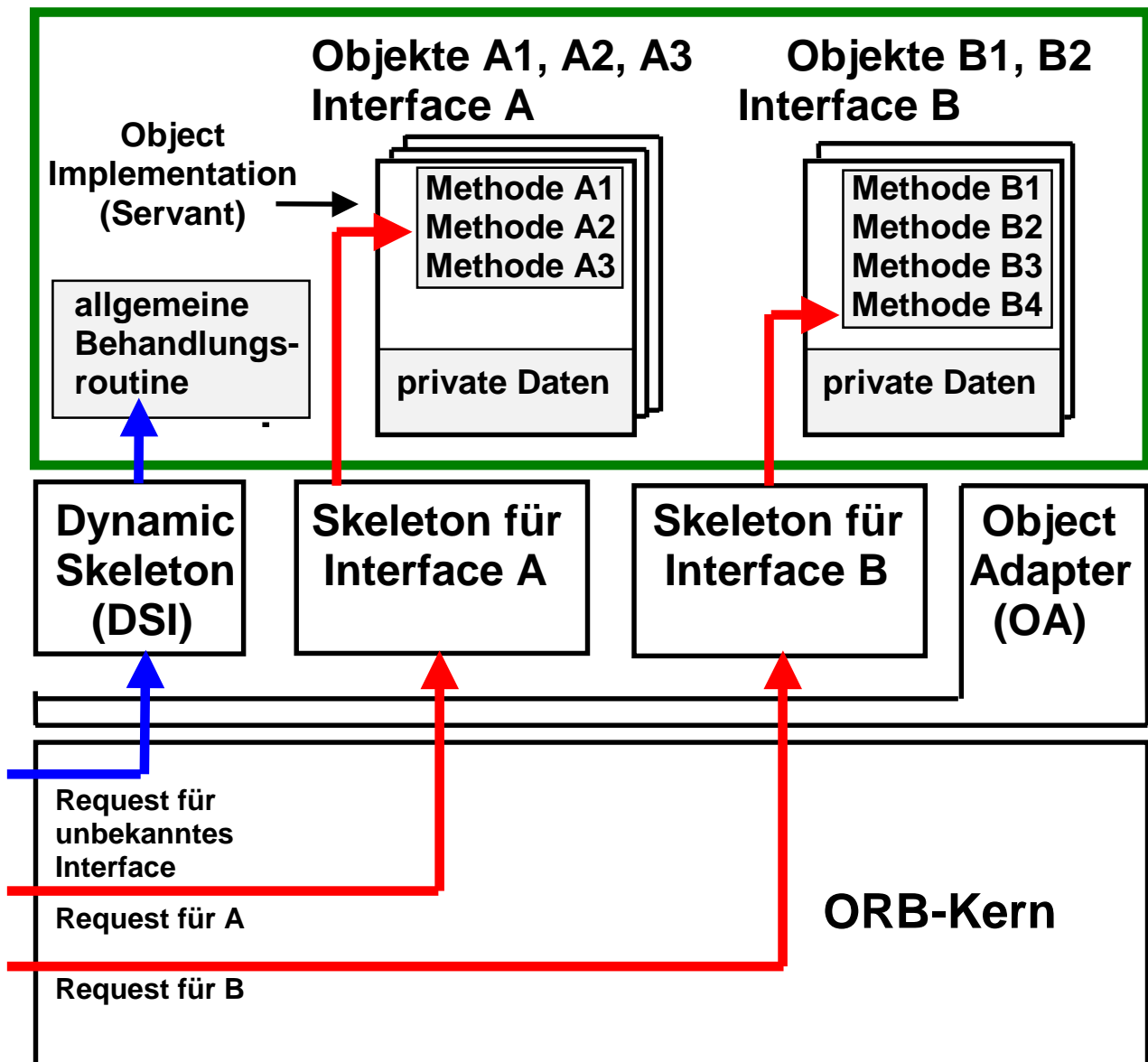
Ein RPC Service wird auch als *Implementation* bezeichnet.

Häufig Hunderte oder Tausende unterschiedlicher RPC Services auf dem gleichen Rechner.

Der RPC Server stellt Verwaltungsdienste für seine RPC Services zur Verfügung, z.B. das *Binding*.

Auf einem Rechner können mehrere RPC Server laufen, die z.B. unterschiedliche Arten von RPC Services gruppieren.

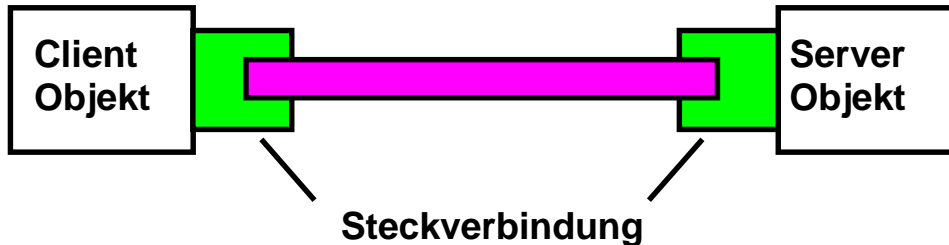
Aufbau eines Corba Servers



CORBA unterscheidet zwischen Server und Objekt. Der Server beinhaltet typischerweise mehrere Objekte, und läuft als Prozess in einem virtuellen Adressenraum. Gleichzeitige Ausführung mehrerer Objekte mit Hilfe von Threads. Mehrere Threads können das gleiche Objekt für mehrere Klienten gleichzeitig zugänglich machen (Locks).

Je 1 Skeleton für Objekte mit identischer Interface.

IDL (Interface Definition Language)



Die Idee ist:

Über ein (binäres) Server Object ist nicht bekannt
wie es implementiert ist
in welcher Sprache es implementiert ist

Nur die Schnittstellen des Server Objektes werden veröffentlicht
Methoden
Parameter

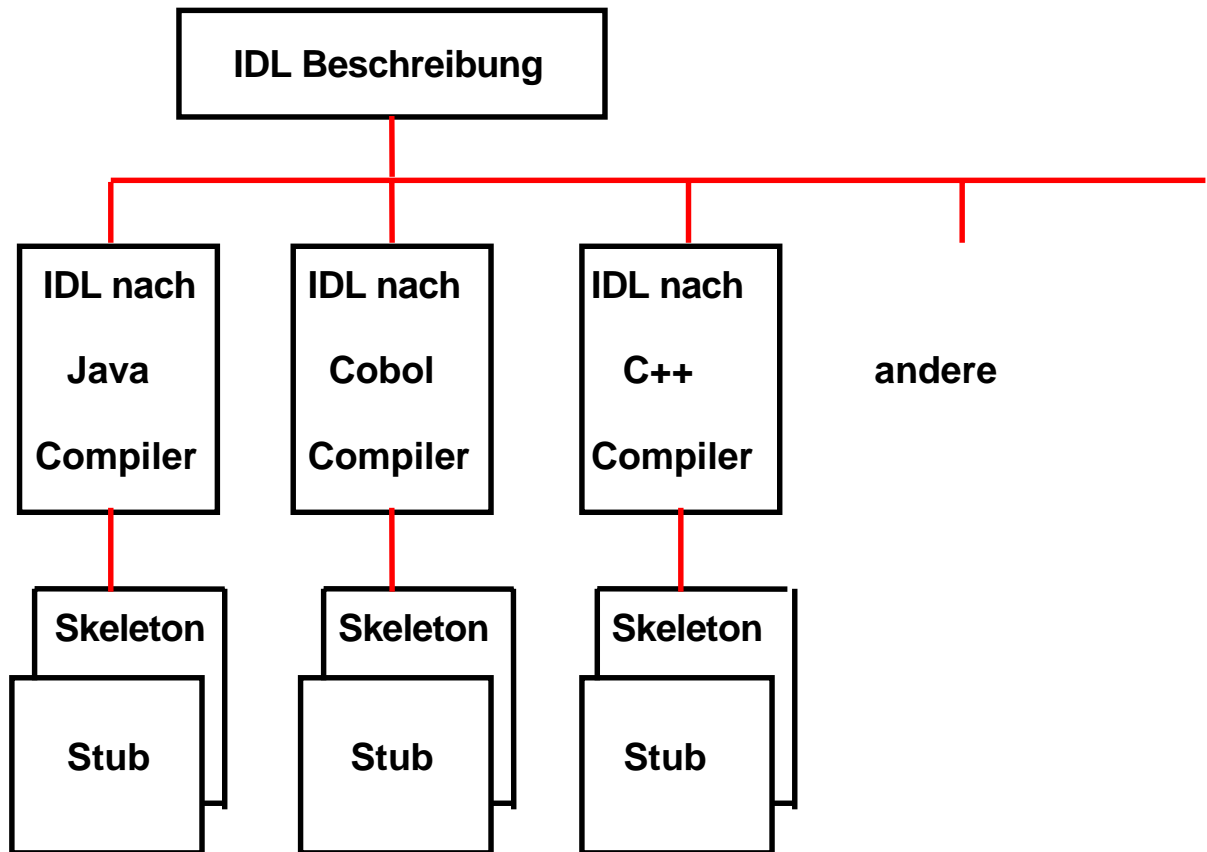
Um die Programmiersprachenunabhängigkeit zu erreichen, wird die
Schnittstellenbeschreibung von der binären (Maschinencode)
Implementierung der Klienten- und Serveranwendung getrennt

Die Beschreibung der Schnittstellen erfolgt in einer einheitlichen
Sprache: IDL . Damit entsteht eine Programmiersprachen-
unabhängige Class Description. Die „Language Mappings“ der
OMG definieren die Umsetzung

Wenn der Programmierer die Schnittstellenbeschreibung des
Server Objektes kennt, kann er sie benutzen um eine
Klientenanwendung zu schreiben

Die Compilierung der IDL Schnittstellenbeschreibung erzeugt auf
der Klienten- und Serverseite ORB spezifische Skeletons und Stubs

Der Skeleton und Stub Code muß Sprachen-spezifisch sein, um mit
der Klienten - und Serveranwendung in deren Sprache zurecht zu
kommen



Language Mappings:

- Werden für verschiedene Programmiersprachen spezifiziert.
- Legen die den IDL-Konstrukten äquivalenten Konstrukte der jeweiligen Programmiersprache fest. Dazu gehören:
 - Einfache und zusammengesetzte Datentypen, Konstanten, Objekte,
 - Operationsaufrufe incl. Parameterübergabe,
 - Setzen und Abfragen von Attributen,
 - Auslösen und Behandeln von Exceptions.
- Garantieren die Zugriffsmöglichkeit auf Objekte unabhängig von der bei der Entwicklung der Client- und Server-Programme benutzten Programmiersprache.
- Sind zum Teil von OMG standardisiert (für C, C++, COBOL, Ada, Java und Smalltalk). Weitere Standards sind z.Z. in Arbeit (für Fortran und PL/1).

IDL

Interface Definition Language

Beschreibt die Schnittstelle, welche von Client-Objekten aufgerufen wird, und die von einer Objekt-Implementierung zur Verfügung gestellt wird.

Angelehnt an ANSI C++:

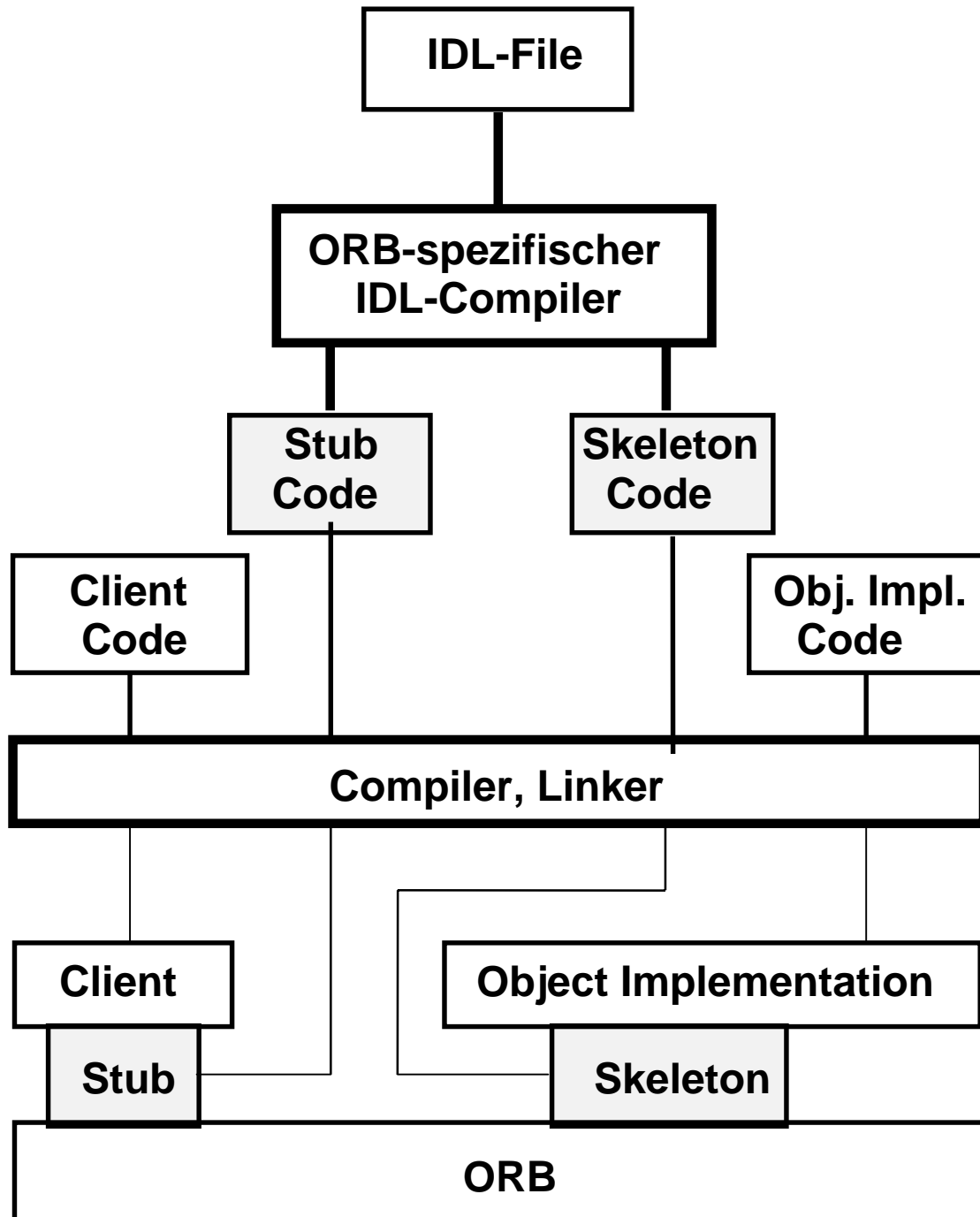
- **Preprocessing im C++ - Stil,**
- **Lexikalische Regeln (mit zusätzlichen Schlüsselwörtern),**
- **Syntax ist Untermenge der C++ - Syntax (Konstanten-, Typ-, Operationsdeklarationen).**

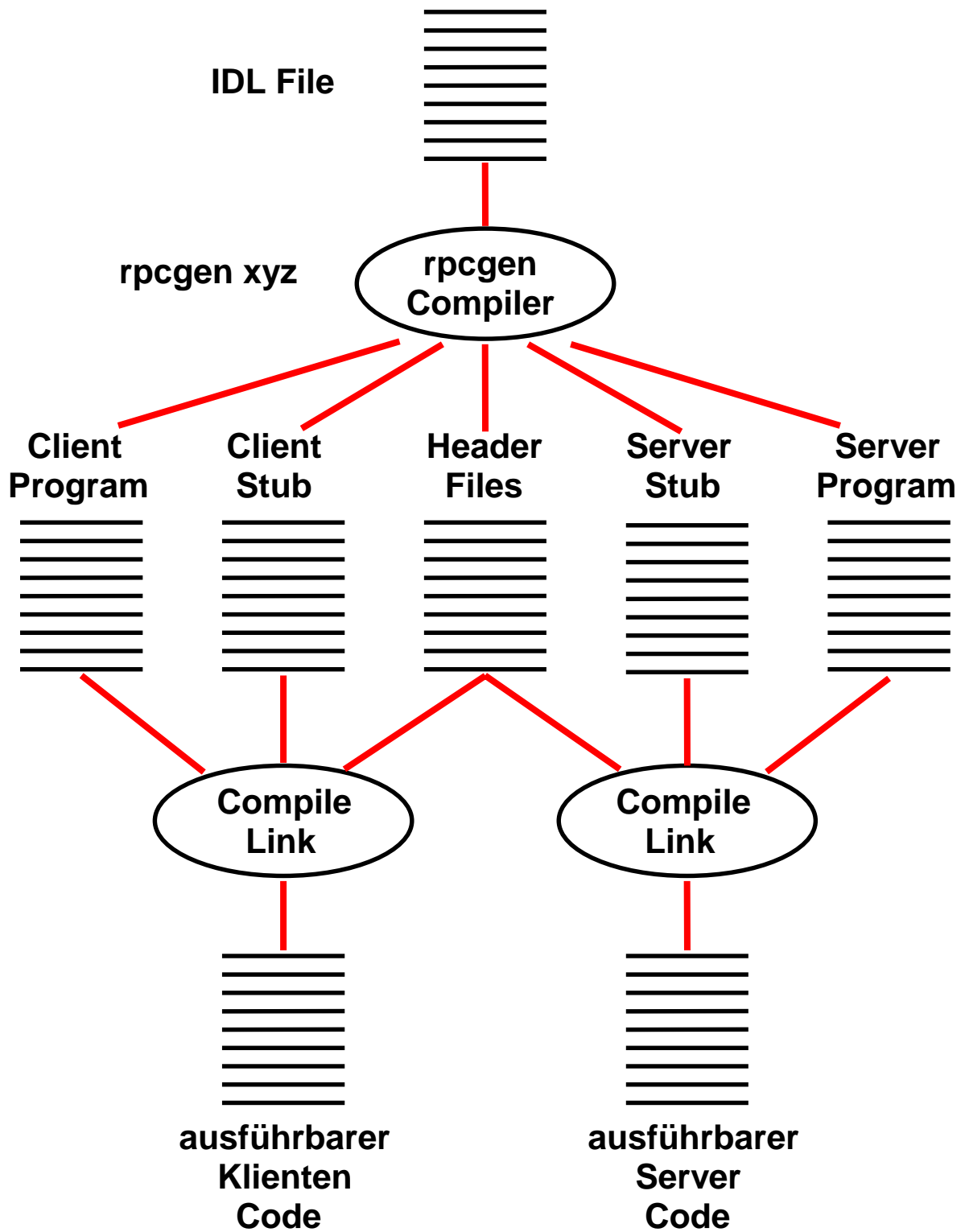
Aber auch mit Unterschieden zu C++, z.B.:

- **Rückkehrtyp von Funktionen muß angegeben werden,**
- **Alle formalen Parameter in Operationsdeklarationen müssen einen Namen haben,**
- **...**

OMG IDL

Erstellung von Client und Objekt-Implementation
mittels IDL-Compiler





Erstellung einer Lower Level Remote Procedure Call Anwendung

Erstellen eines CORBA Programms

- 1. Definition der Server Schnittstelle unter Benutzung der Interface Definition Language (IDL)**
- 2. IDL Definition mit Hilfe des IDL Precompilers übersetzen**
- 3. Interface Definition in das Interface Repository binden**
- 4. Code für die Server Implementierung schreiben**
- 5. Server Code kompilieren**
- 6. Run-Time Objekte mit dem Implementation Directory registrieren**
- 7. Zur Start-up Zeit Objekte auf dem Server instanziiieren**
- 8. Code für die Client Implementierung schreiben**
- 9. Client Code übersetzen**
- 10. Run**

Interoperable Object Reference IOR

Zeichenkette, die ein Objekt innerhalb eines verteilten CORBA Systems eindeutig kennzeichnet.

Vergleich:

Internet

CORBA

134.2.2.102

Object Reference (IOR)

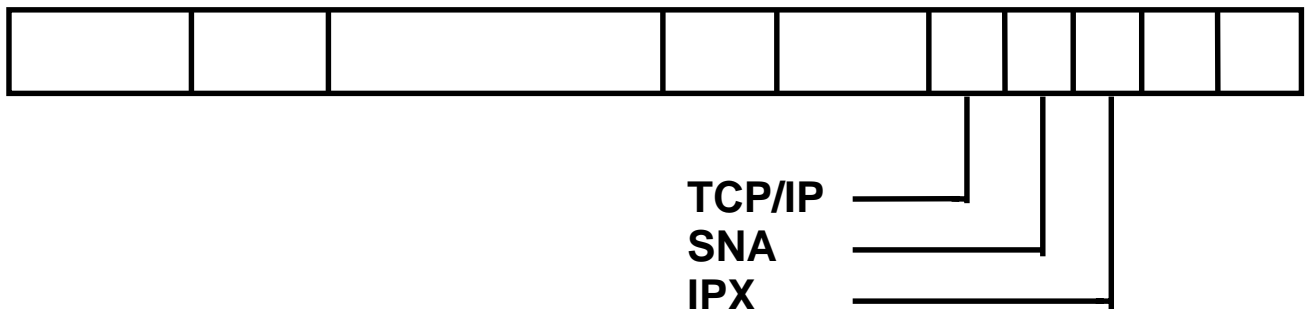
informatik.uni-tuebingen.de

String (Name)

CORBA ORB's unterschiedlicher Hersteller übermitteln Object References nach dem IOR (Interoperable Object Reference) Standard.

Anwendungen verwenden String Namen. Es ist die Aufgabe eines Name Servers, String Namen in Object References zu übersetzen.

Interoperable Object Reference (IOR) Format



- **OMG Standard**
- **IOR's sind unique (eindeutig).**
- **IOR enthält u.A. ein Feld für jedes benutzbare Transport Protokoll.**
- **Für IIOP (TCP/IP) schließt dies die Internet Adresse, eine Portnummer und eine intern vom ORB erzeugte und verwendete Objektidentifikation ein. Die Netzwerk Adresse bezeichnet die zuletzt bekannte Lokation des Objekt ORB's.**

Wurde das Objekt zwischenzeitlich verlagert, verweist der ursprüngliche ORB auf den neuen ORB mit einer "*location_forward*" IIOP Nachricht.

Unterschied DNS – Corba Namensdienst

DNS Namen werden hierarchisch und zentral verwaltet:

Network Information Center (NIC)

DENIC

RZ Uni Leipzig

jedi.informatik.uni-leipzig.de

Milliarden von Corba Objekten, werden ad hoc erzeugt.
Zentrale Verwaltung nicht praktikabel. Erzeugung der IORs
durch einen Zufallszahlengenerator.

Wenn Zufallszahl genügend lang, denn geringe
Wahrscheinlichkeit, dass zwei identische IORs erzeugt
werden. Falls es doch passiert: Konfliktauflösung per
email.

An Interoperable Object Reference

***IOR:000000000000003249444c3a494e616d696e674d616e616
765644f626a656374434453496d706c2f4e616d696e67436f6e7
46578743a312e300000000000001000000000000168010101
000d000000392e3136342e3138322e353700008403bc0000000
432333645303145332d373234442d313644442d454646332d30
3134393039413442363339000200005203000033000f0069424f
494d436f6e7461696e65720003000e00695472616e735379734f
626a73000e00536f6d436f6e7461696e65724900580016006943
44534e616d696e67436f6e74657874486f6d65003e002f2e2e2e2
f706f6f6c33345f63656c6c2f4342432d6c6f63616c2d726f6f747
32f706f6f6c33342e626f65626c696e67656e2e64652e69626d2e
636f6d0003000000044d4249050000000005000101000000090
300006200000001510000004342436f6e6e6563746f722d706f6
f6c33342e626f65626c696e67656e2e64652e69626d2e636f6d2d
706f6f6c33342e626f65626c696e67656e2e64652e69626d2e636
f6d2d4e616d652d5365727665724753535f444345004300dc0e0
0001400000008000000016f7e007e00bc0b***

OMG CORBA Namensdienst

Mechanismus, mit dem Objekte auf dem ORB andere Objekte lokalisieren

Ein „Name-Binding“ ist eine Zuordnung Name - Objekt.

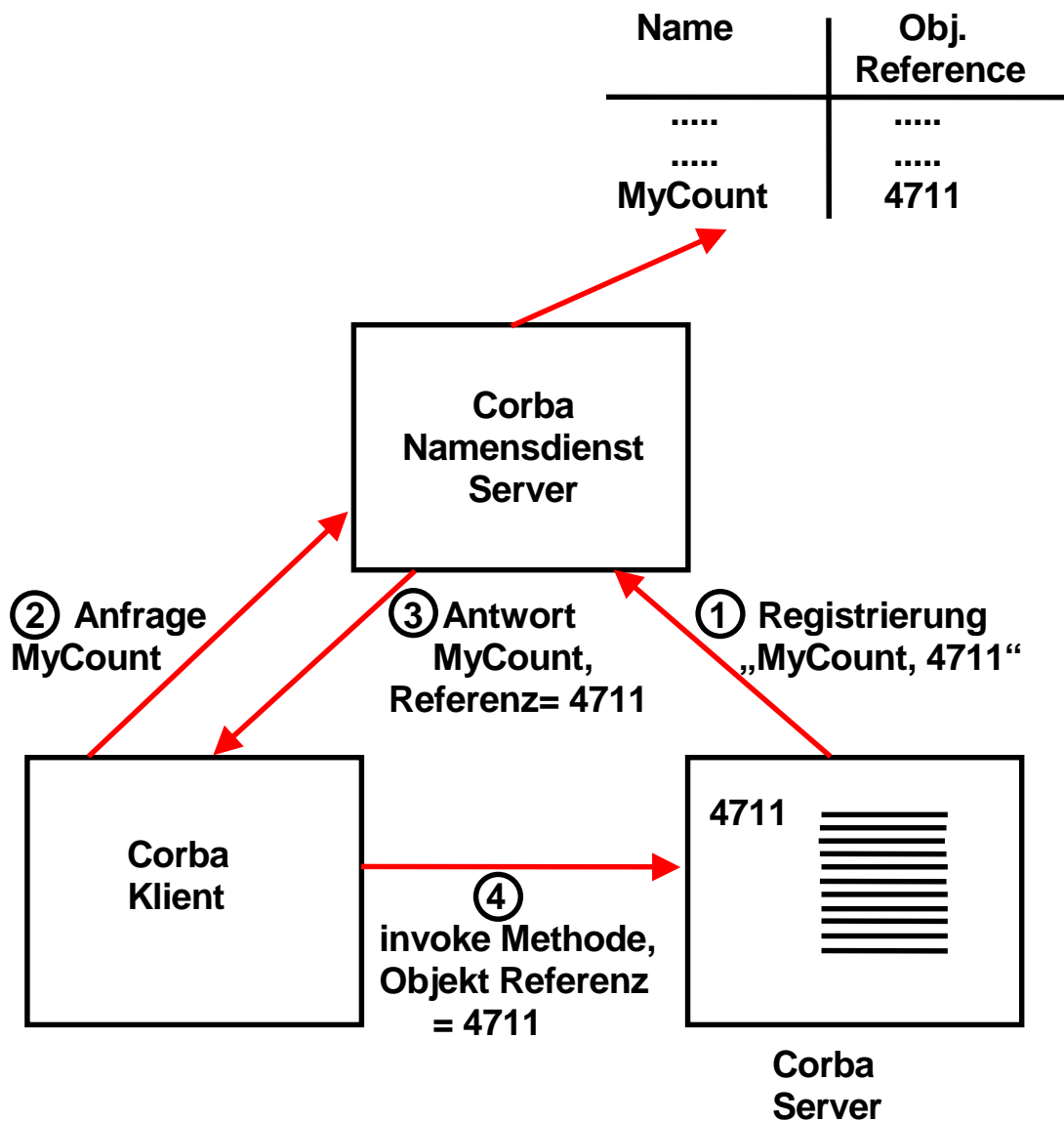
Verwendet hierarchische Struktur. Basiert auf transparenter Kapselung existierender Namensdienste wie LDAP, X.500, DNS oder Sun NIS.

Bildet den Namen eines Objektes auf eine eindeutige, maschinenlesbare Objekt Referenz ab.

Es kann mit absoluten Namen oder mit Namen innerhalb ihres Context gearbeitet werden. spruth, informatik, uni-tuebingen und de könnten Komponenten eines CORBA Namens sein.

Eine Komponente besteht aus 2 Attributen: identifier, kind. Das Attribute kind kann eine Beschreibung der Komponente enthalten, z.B. file-typ.

Einige ORB Hersteller, z.B. Inprise, Orbix, bieten zusätzlich eigene, proprietäre Namendienste an, die Broadcasts verwenden und nur innerhalb des gleichen IP Subnetzes arbeiteten.



Corba Namensdienst

Corba Object Reference : Zeiger auf ein Objekt zur Laufzeit

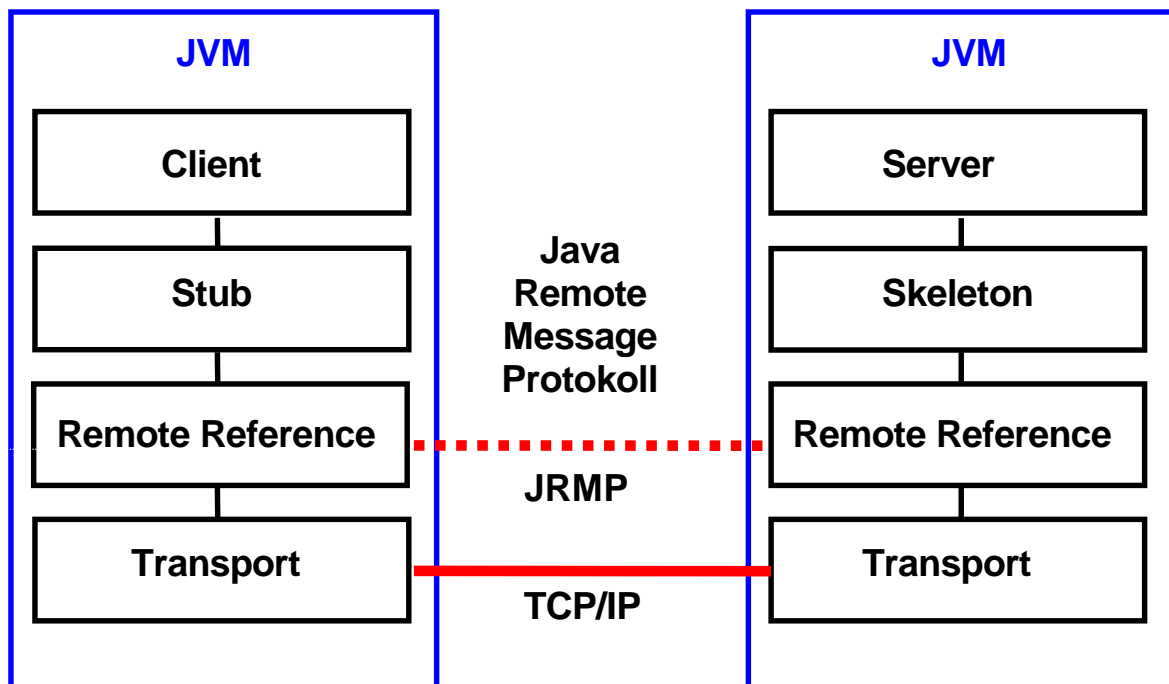
Die vom Programmierer verwendete Repräsentation (Namen) einer Objekt Referenz ist nicht mit der vom ORB verwendeten Repräsentation identisch

Common Object Services (COS)

Common Object Services sind modulare zusätzliche CORBA System-Dienstleistungen, welche die Funktionalität des ORB ergänzen. Sie stellen Bausteine für die Anwendungsentwicklung dar. Es existieren Spezifikationen der OMG für:

- **Naming Service**
- **Persistence Service**
Speicherung von Objekten in relationalen oder Objekt Datenbanken oder einfachen Dateien
- **Concurrency**
Lock Manager Dienste für Transaktionen oder Threads
- **Transaction Service**
Two-Phase Commit Koordination für flache oder nested Transactions

sowie weitere Dienste wie Security, Message, Time, Event, Trader,

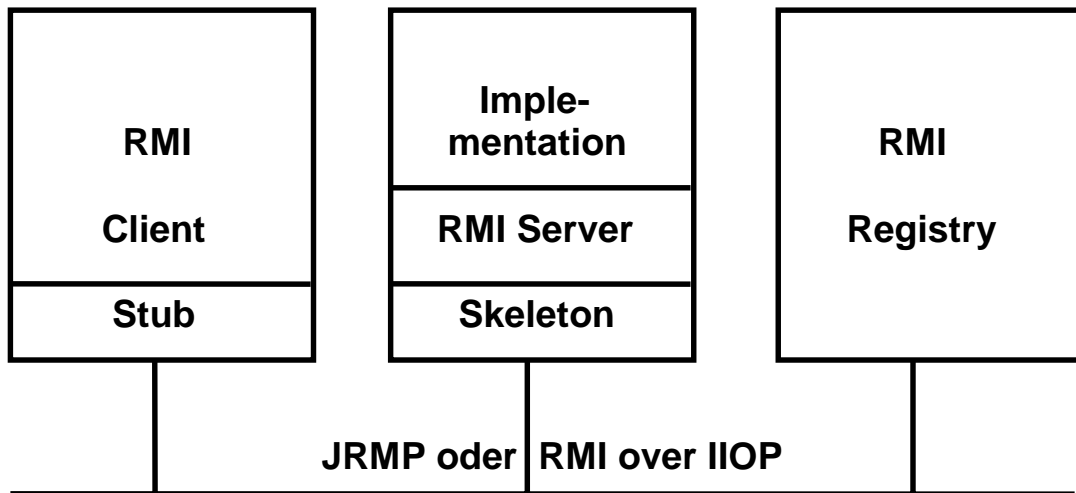


Remote Method Invocation (RMI)

Client/Server-API für den Aufruf von Java Programmen auf geographisch entfernten Rechnern

- Normalfall für Java: Es werden nur Methoden von Objekten innerhalb der gleichen JVM aufgerufen.
- Objekte in einer lokalen JVM können Methoden von Objekten in einer entfernten JVM mit Hilfe von RMI aufrufen.
- Diese JVMs können auf verschiedenen Rechnern im Netz laufen.
- Realisierung: Erzeugen eines Stellvertreters (Client-Stub) des entfernten Objekts in der lokalen JVM. Dieser kommuniziert mit dem Skeleton des entfernten Objektes.

Remote Method Invocation (RMI)



Drei verschiedene Prozesse, die auf dem gleichen Rechner oder auf entfernten Maschinen laufen können. RMI Registry ist ein einfacher RMI Namensdienst. Die Alternative ist JNDI

Der Klient besorgt sich eine Handle für das entfernte Objekt, indem er RMI Registry aufruft (//URL/registered name).

Eine Referenz auf das entfernte Objekt wird zurückgegeben. Jetzt kann eine Methode des entfernten Objektes aufgerufen werden.

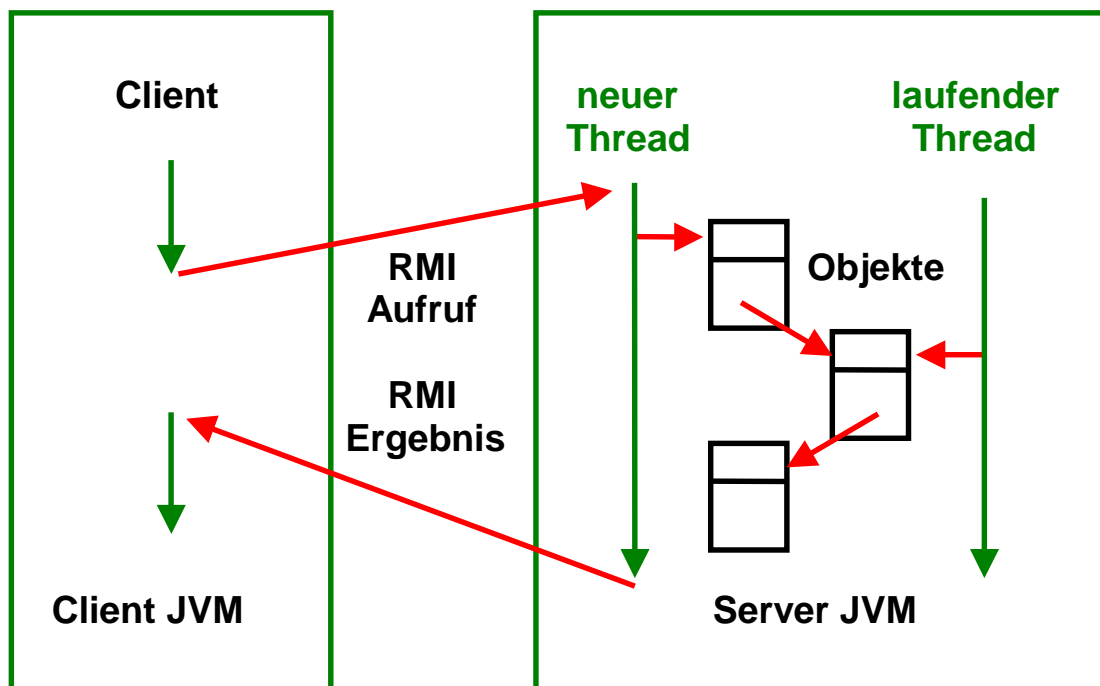
Dieser Aufruf erfolgt zum Skeleton, der das entfernte Objekt repräsentiert.

Der Stub verpackt die Argumente (marshaling) in einen Datenstrom, der über das Netzwerk geschickt wird.

Das Skeleton unmarshals die Argumente, ruft die Methode auf, marshals die Ergebnisswerte und schickt sie zurück.

Der Stub unmarshals die Ergebnisswerte und übergibt sie an das Klientenprogramm.

Threads auf der Server-Seite



Jeder Aufruf durch einen RMI-Klienten erzeugt auf der Server-Seite einen neuen Thread. Das bedeutet, dass dort mehrere solcher Threads gleichzeitig laufen können, zusammen evtl. mit weiteren dauerhaft laufenden Threads des Servers (z.B. Garbage Collector).

Bei einer Transaktionsverarbeitung ist Synchronisation erforderlich (mit `synchronized`-Blocken, ggf. auch mit `wait()` und `notify()`).

Erstellen einer RMI Remote Class

RMI benötigt wie Corba einen RMI Server, unter dem die RMI Implementation läuft.

Zu kodieren sind:

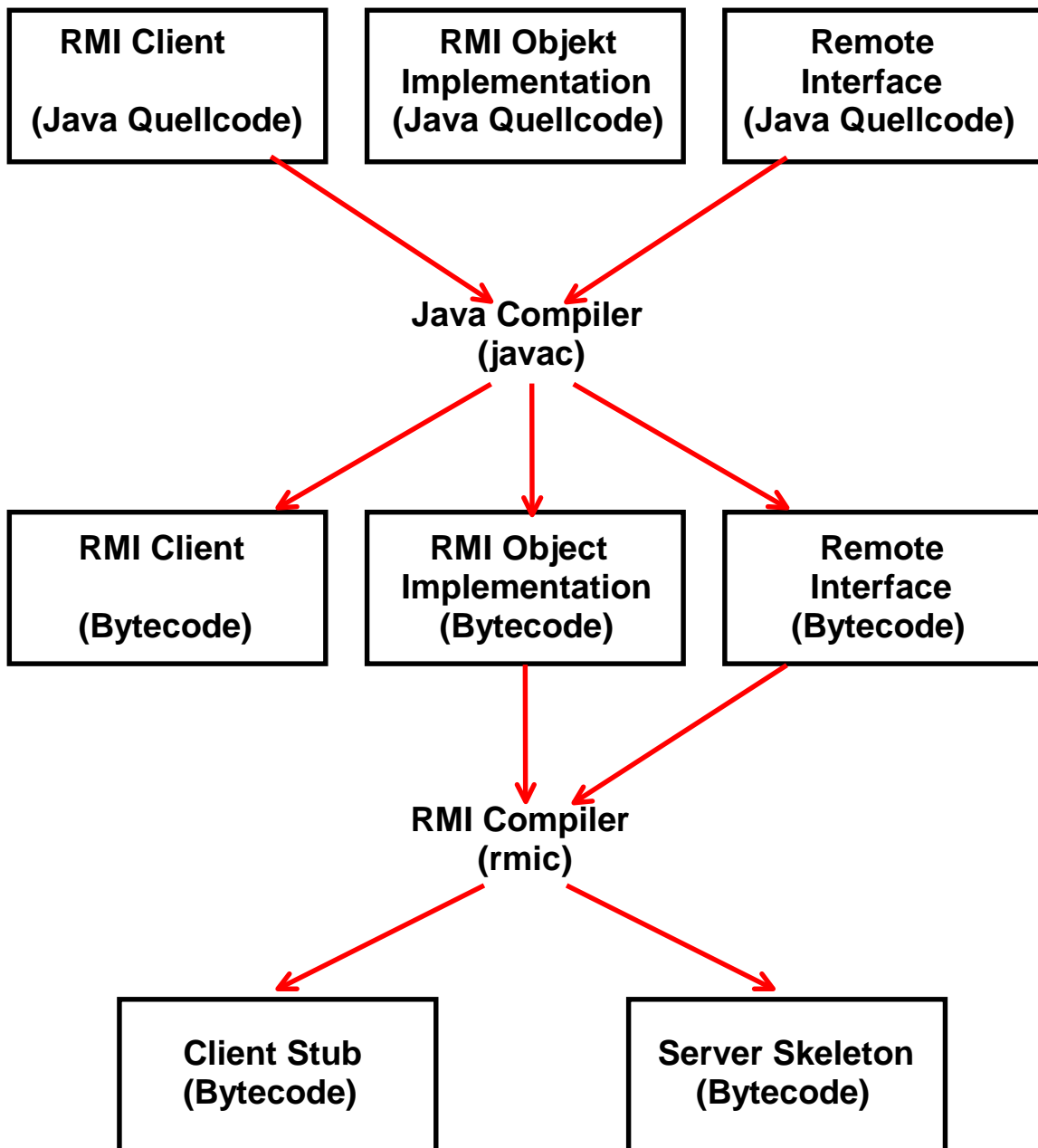
- Interface
- Client
- Implementation
- Server

Die Remote Interface enthält die Namen aller Methodenaufrufe und die dazugehörigen Parameter. Beispiel für die Interface einer Methode Addition, die zwei Zahlen a und b addiert:

```
public interface Addition extends java.rmi.Remote {  
    public long add(long a, long b)  
    throws java.rmi.RemoteException;  
}
```

Schritte zur Erstellung und Ausführung einer Anwendung

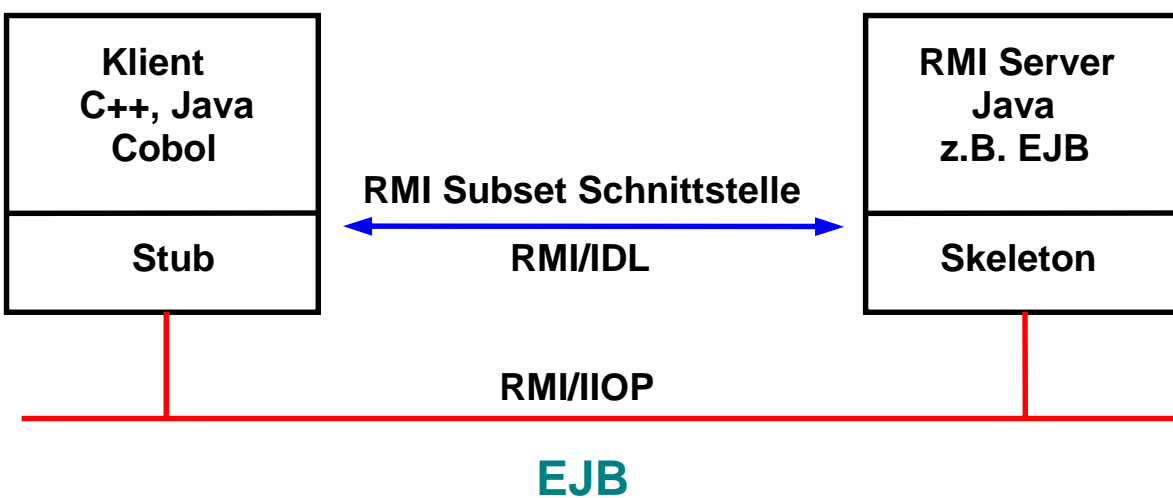
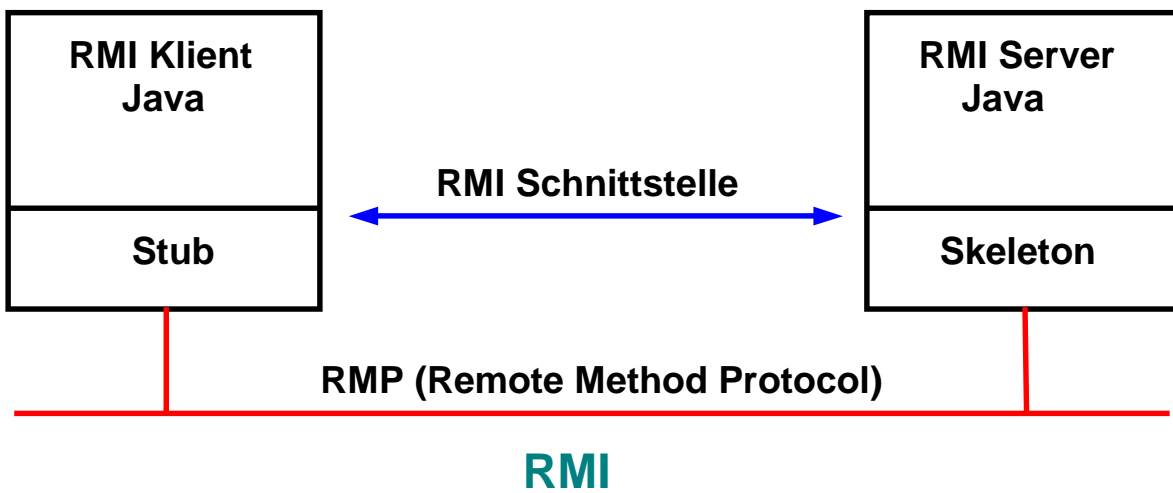
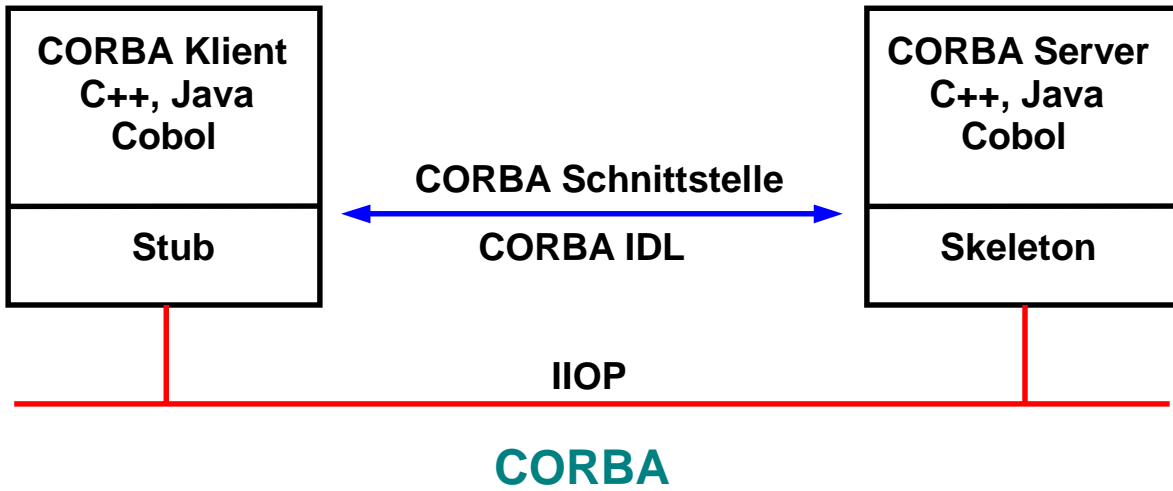
1. Interface definieren, mit der das Remote Object aufgerufen wird.
> extends interface java.rmi.Remote .
2. Die Implementierung der Server Anwendung schreiben. Muss die Remote Interface implementieren. .
> extends java.rmi.server.UnicastRemoteObject
3. Klassen kompilieren.
4. Mit dem Java RMI Compiler Client Stubs und Server Skeletons erstellen. > rmic xyzServerImpl
5. Klient implementieren und übersetzen.
6. Start Registry, Server starten, Klienten starten.

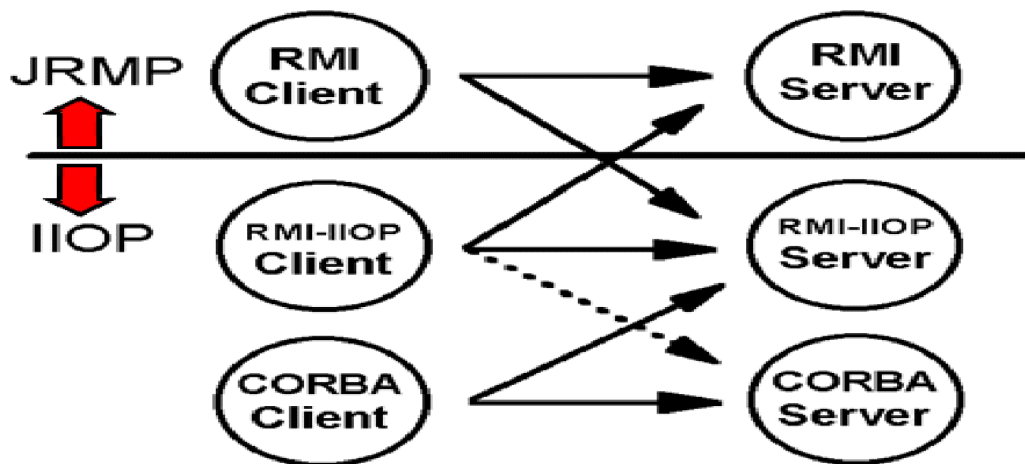


Entwicklungsprozess für RMI Objekte

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 schließt neben Java auch C/C++, Cobol, PL/1, Ada, Fortran, XML, Lisp, Smalltalk und Python ein. RMI ist auf Java beschränkt.
- RMI Anwendungen sind einfacher zu erstellen als Corba Anwendungen, weil die Notwendigkeit der IDL Definition in einer getrennten Sprache entfällt. Java Interface Definitionen sind Teil der Java Sprache.-
- 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).





RMI-over-IIOP

Corba verwendet das IIOP Protokoll für die Client-Server Kommunikation

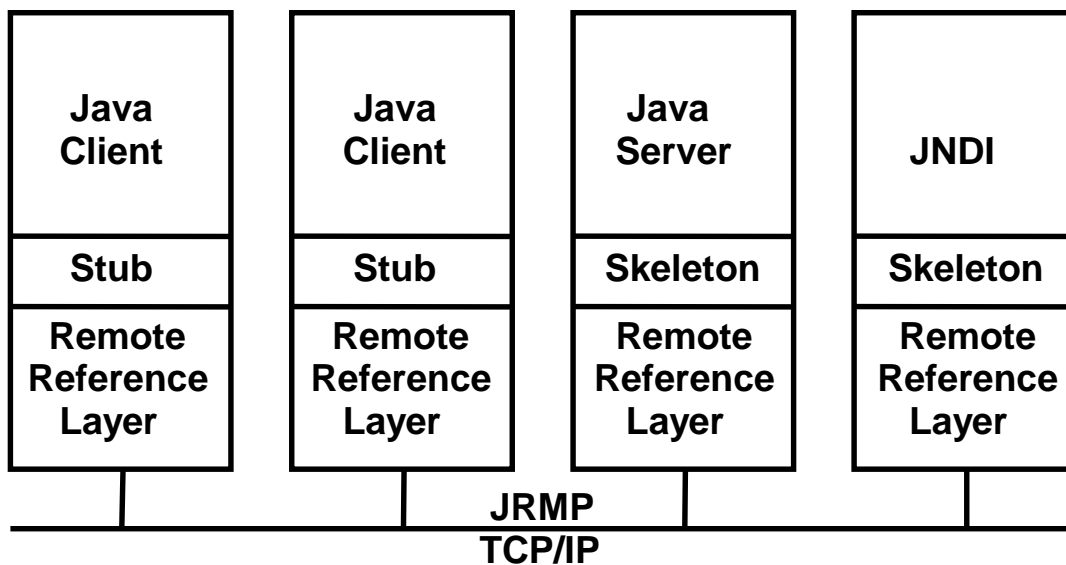
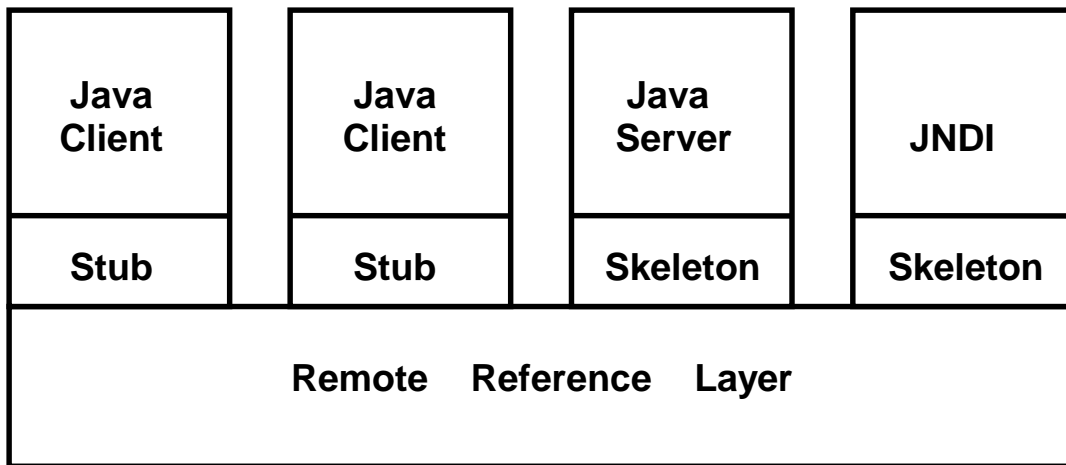
RMI verwendet das JRMP Protokoll für die Client-Server Kommunikation

Der Java J2EE Standard und sein JDK unterstützt "RMI-over-IIOP" als zusätzliche Alternative zu dem bisherigen "RMI-over-JRMP". Dies bedeutet, es wird IIOP an Stelle von JRMP als Übertragungsprotokoll eingesetzt.

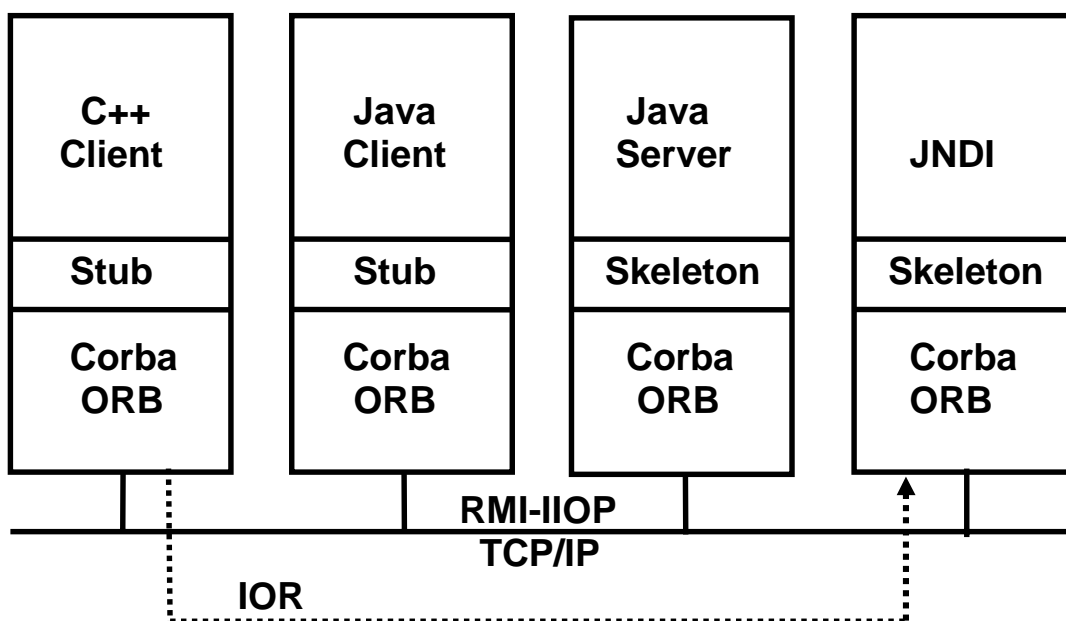
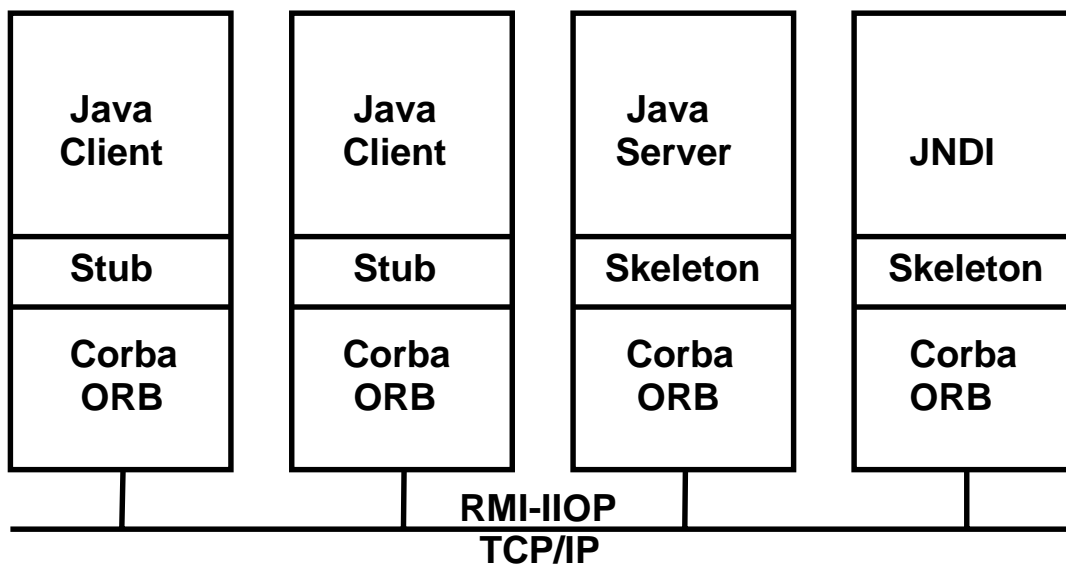
Es wird der `rmic-iiop` Compiler an Stelle des bisherigen `rmic` Compilers eingesetzt, um die RMI Stubs und Skeletons zu erzeugen. Davon abgesehen ändert sich wenig an der Erzeugung und dem Betrieb von RMI - Java Anwendungen. Einige Hersteller, z.B. IBM, wollen deshalb RMI-over-JRMP ganz durch RMI-over-IIOP ablösen.

Soll eine RMI Komponente (in Java) mit einer Corba Komponente (z.B. in C++) kommunizieren, so ist es erforderlich, aus der Java Interface Definition mit Hilfe des `rmic-idl` Compilers eine IDL File zu erstellen. Letztere erzeugt dann mit Hilfe des normalen IDL Compilers Stubs und Skeletons.



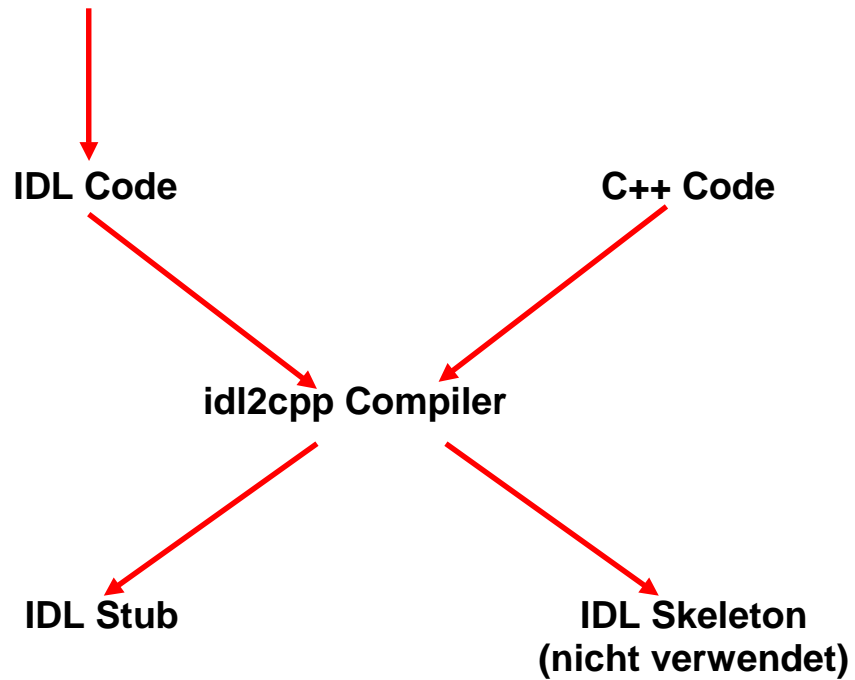


Java Remote Methode Invokation (RMI)

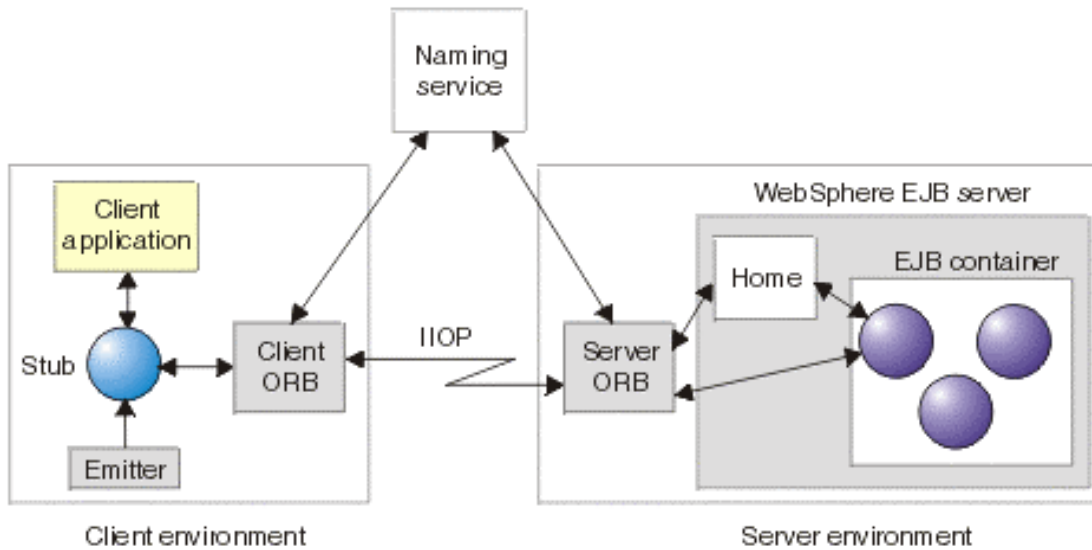


RMI over IIOP

Interface des Java Server Objects
`public interface xxx extends java.rmi.Remote {`



**Erstellen des Stubs für einen C++ Client
in einer Java RMI-IIOP Umgebung**

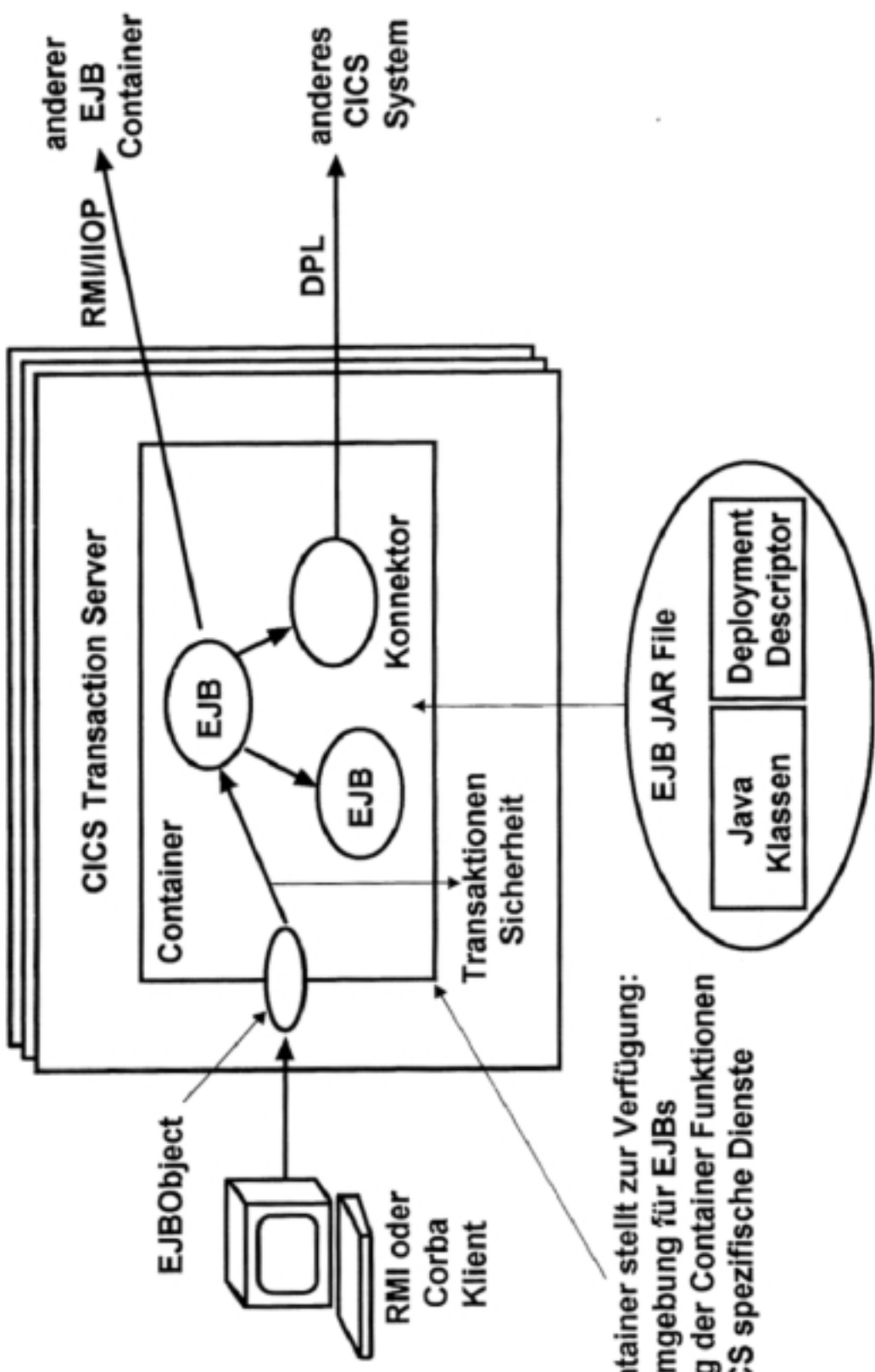


CORBA Klient greift auf EJBs eines WebSphere EJB Servers zu.

The EJB server provides the server implementation objects (Enterprise JavaBeans) that client applications need to access and implements the services that support those objects. The EJB class file is used to generate IDL for the class and its home (this is reverse of the typical CORBA model where IDL is used to generate the object).

When the client wants to call a method on a server object (an Enterprise JavaBean), the following sequence of events occur:

- I. When the client environment is started, the client ORB is initialized and the ORB bootstrap process gets access to the naming service (with CORBA CosNaming bindings).
- II. When a client application needs to access an Enterprise JavaBean, the client environment uses the naming service to find the home for the bean.
- III. The home locates or creates the Enterprise JavaBean then passes the interoperable object reference (IOR) of the bean back to the client.
- IV. The client's ORB creates a stub object (local to the client) for the bean and stores the IOR in the stub object.
- V. The client uses the stub object to communicate with the remote bean as though it was in the local address space.



CICS Container stellt zur Verfügung:
 Laufzeitumgebung für EJBs
 Abbildung der Container Funktionen
 auf CICS spezifische Dienste

Microsoft DCOM

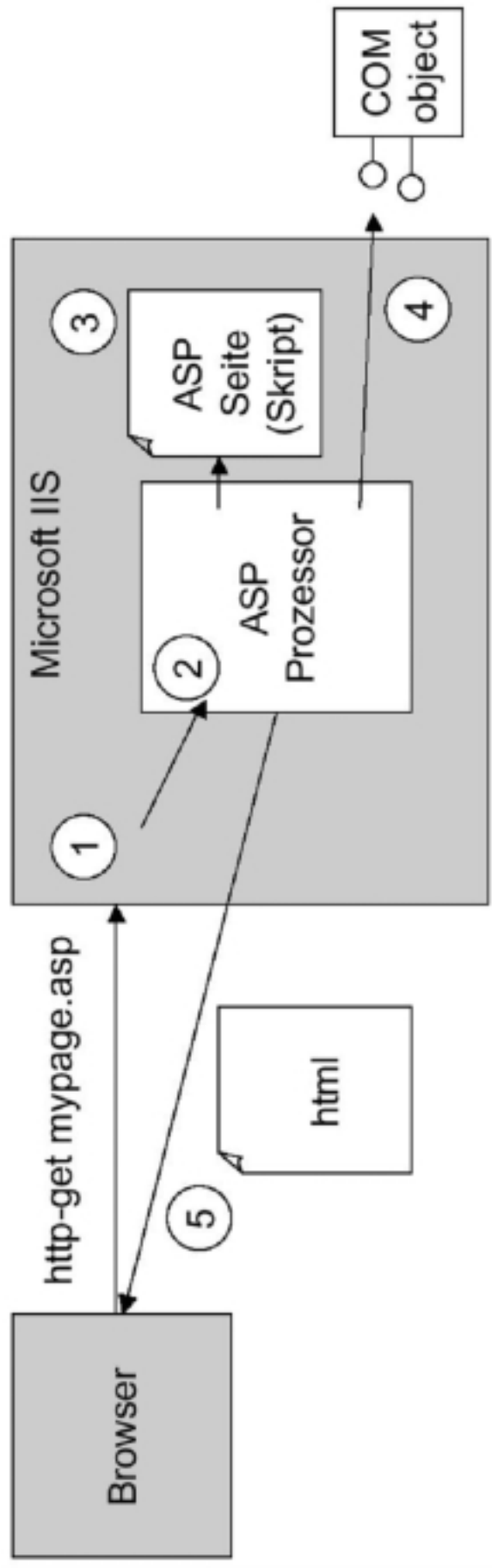
Das DotNet Objektmodell ist Microsoft's proprietäre Alternative zu CORBA.

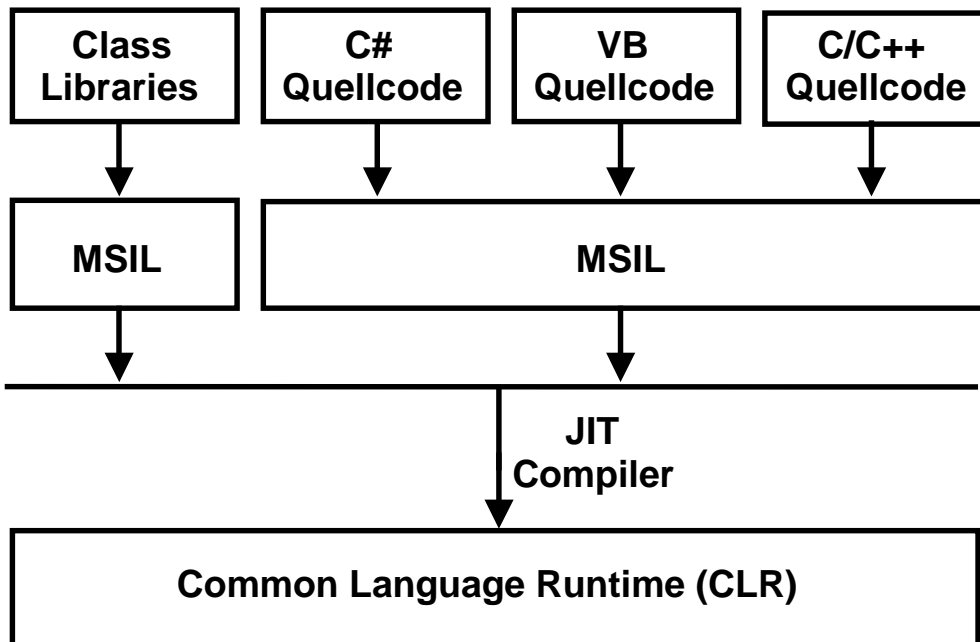
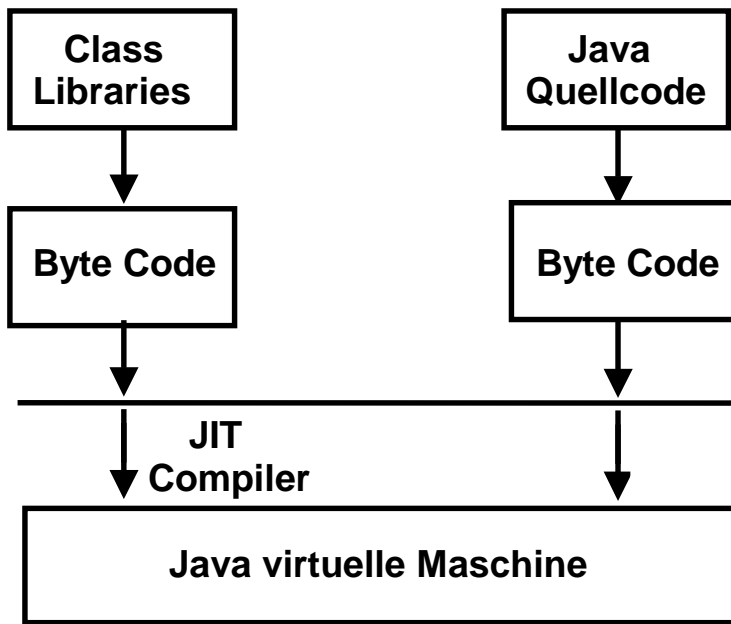
Verwendet (leicht modifizierten) DCE RPC.

Integriert (wie Corba) Komponenten unterschiedlicher Hersteller in binärer Form, die in unterschiedlichen Sprachen geschrieben sein können. Verwendet hierzu eine IDL.

Keine Vererbung.

Verfügbar unter z/OS und den meisten Unix Dialekten (incl. LINUX), jedoch bisher kaum Einsatz außerhalb der Windows Betriebssystemfamilie.





Microsoft Common Language Runtime (CLR)

Vergleich CORBA - DCOM

Beide Technologien werden vermutlich für längere Zeit in Wettbewerb miteinander stehen.

In reinrassigen Microsoft Umgebungen (z.B. mittelständische Unternehmen) gilt DCOM als ein relativ leicht einsetzbares, zuverlässiges, ausgereiftes und performantes Verfahren.

In Zusammenarbeit mit Microsoft stellt die Firma Software AG seit 1999 DCOM Implementierungen für Linux, Sun Solaris, HP UX, IBM AIX und OS/390 zur Verfügung. Diese werden jedoch nur selten eingesetzt. In heterogenen Umgebungen ist DCOM bisher wenig vertreten.

CORBA ist besser geeignet, bestehende Altanwendungen einzubinden.

Die Java Browser Sicherheit (Sandbox) ist besser als DCOM (vertrauensbasierte Verfahren).