Mainframe Internet Integration

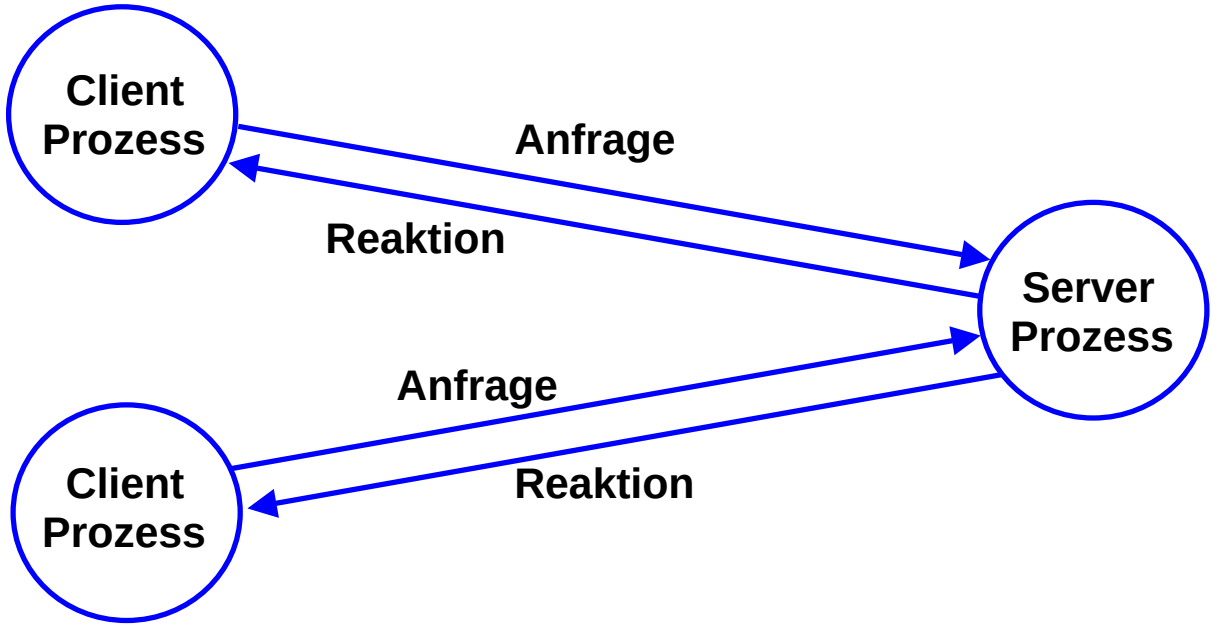
Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

SS2013

Java Remote Method Invocation Teil 1

Object Request Broker

Client/Server-Modell



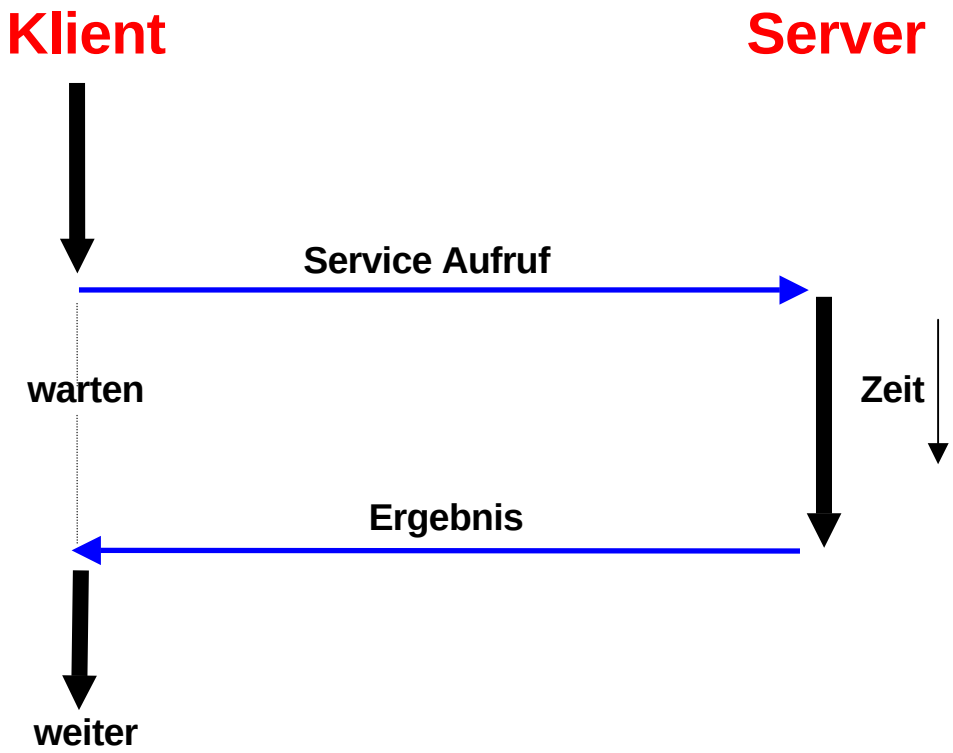
In einer Client/Server Konfiguration bietet ein Server seine Dienste (Service) einer Menge a priori unbekannter Klienten (Clients) an.

Dienst (Service)	**Software-Instanz, die auf einem oder mehreren Rechnern ausgeführt wird.**
Server:	**Rechner, der Dienst-Software (als Service bezeichnet) ausführt**
Klient:	**Nutzer eines Serverdienstes**
Interaktionsform:	**Anfrage/Reaktion (Request/Reply)**

Ein Server-Rechner kann gleichzeitig mehrere Serverdienste (Services) anbieten. Beispielsweise bietet ein CICS Server multiple CICS Servies (identifiziert durch unterschiedliche TRIDs) einer Vielzahl von CICS Klienten an. Der größere Teil aller Anwendungen in Wirtschaft und Verwaltung läuft auf Client/Server Systemen.

Synchroner Methodenaufruf

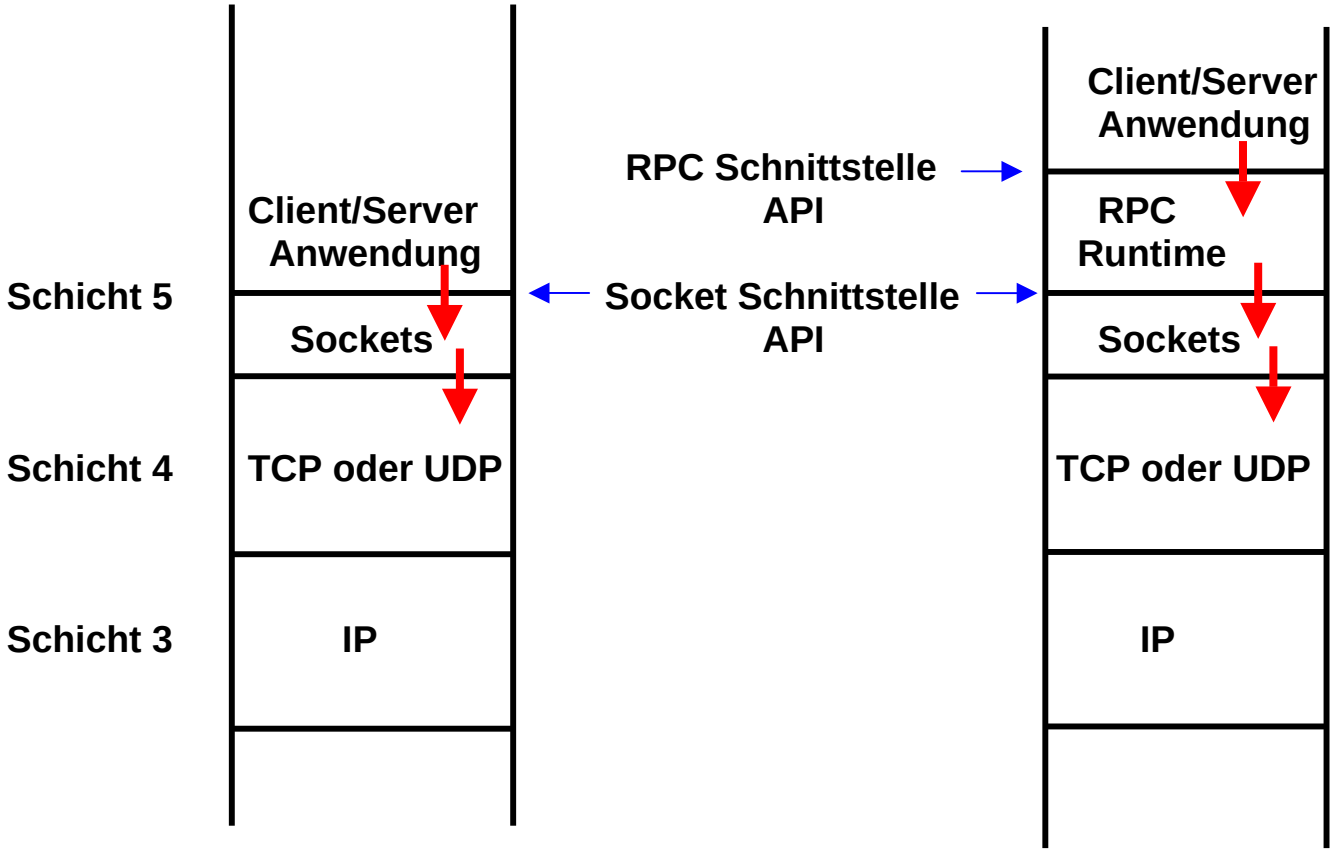
Client/Server Systeme arbeiten synchron.



Der Klient sendet eine Nachricht an den Server. Das Eintreffen der Nachricht bewirkt den Aufruf eines von mehreren Services, die der Server bereitstellt. Der Klient wartet (blockiert), bis das Ergebnis des Service Aufrufes verfügbar ist.

Neben dem Synchronen Methodenaufruf existiert der asynchrone Methodenaufruf. MQSeries und MDB (Message Driven Beans) sind Beispiele für einen asynchronen Service Aufruf: Der Klient wartet nicht auf eine Antwort des Servers.

Diese Folien sind teilweise eine Wiederholung von WebSpphere MQ Teil 1 aus dem Vorlesungsscript „Einführung in z/OS“. Siehe http://jedi.informatik.uni-leipzig.de/de/Vorles/Einfuehrung/Mq/MQ01.pdf#page=02



De Facto alle Client/Server Verbindungen benutzen heute TCP oder UDP in der Schicht 4 des OSI Modelles. In der darüberliegenden Schicht 5 wird meistens das Socket Protokoll eingesetzt. Sockets sind leistungsfähig und vielseitig einsetzbar. Das Anwendungsprogramm des Klienten benutzt die Socket Schnittstelle, um mit dem Anwendungs-programm (Service) des Servers zu kommunizieren. Sockets sind jedoch schwierig zu programmieren.

Deswegen wird universell der Remote Procedure Call (RPC) eingesetzt. Der RPC enthält bereits viele Funktionen, die bei der Nutzung von Sockets von Hand programmiert werden müssen. Das Anwendungsprogramm des Klienten benutzt die RPC Schnittstelle, um mit dem Anwendungsprogramm (Service, hier auch als Procedure bezeichnet) des Servers zu kommunizieren. Die RPC Runtime übersetzt die RPC Aufrufe in Socket Aufrufe.

Die RPC Runtime ist eine Sammlung von Routinen, welche die RPC Schnittstelle (API) implementieren.

RPC Implementierungen

Sockets für TCP/IP , sowie LU 6.2 und APPC für SNA, sind Protokolle auf der niedrigsten Verbindungsstufe (Schicht 5 des OSI Modells).

Darüberliegende Protokolle (Schicht 6 und 7) werden allgemein als Remote Procedure Call (RPC) bezeichnet. Leider existiert eine Vielzahl von RPCs, deren APIs (Application Programming Interface) in den meisten Fällen inkompatibel sind:

Klassische Remote Procedure Calls:

- **SUN RPC (Standard in Solaris und vielen Linux Versionen),**
- **DCE RPC (unterstützt von Microsoft und IBM, besonders auch von z/OS),**
- **CPI-C (von IBM, ursprünglich für SNA entwickelt).**

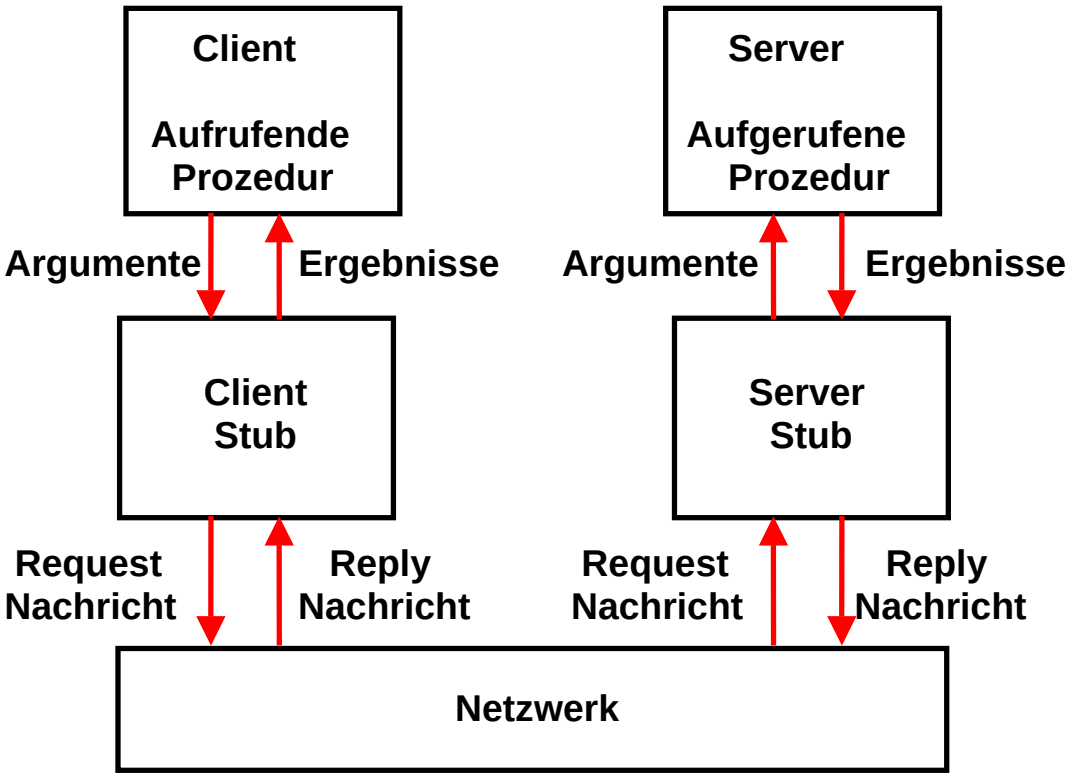
Objekt-orientierte Remote Procedure Calls:

- **Corba,**
- **RMI (Remote Method Invocation)**
- **DotNet der Fa. Microsoft, auch als .Net / COM+ / DCOM / ActiveX / DNA / ASP.NET bezeichnet. Die Abgrenzung der Begriffe ändert sich häufig. Wir benutzen durchgängig den Begriff DotNet. DotNet verwendet eine Erweiterung des DCE RPC.**

Anwendungsspezifische Remote Procedure Calls:

- **CICS Distributed Program Link (DPL)**
- **Web Services verwenden den SOAP RPC**

Für den RPC sind Prozedur, Unterprogramm, Subroutine und Methode (objektorientierte Programmierung) austauschbare Begriffe. Eine Procedur implementiert z.B. einen serverseitigen Service. Ein Prozedur-Aufruf (request) besteht aus einer Nachricht, welche den Namen der aufgerufenen Prozedur sowie eine Liste der übergebenen Parameter enthält.



Remote Procedure Call

Client und Server laufen als zwei getrennte Prozesse.

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

Ein Server-seitiges RPC Programm stellt den Klienten seine Dienste über eine Interface (Schnittstelle) zur Verfügung. es definiert seine Interface unter Benutzung einer Interface Definition Language (IDL).

Ein Stub Compiler liest die IDL Schnittstellen-beschreibung und produziert zwei Stub Proceduren für jede Server Procedure (Client Side Stub und Server Side Stub).

Der klassische RPC benutzt die Bezeichnung Server Stub. Modernere RPCs verwenden statt dessen die Bezeichnung Skeleton. Die Bezeichnungen Server Stub und Skeleton sind austauschbar.

Stubs bzw. Skeletons implementieren eine RPC Runtime. Sie übersetzen Aufrufe der Client API in Sockets und dann in Nachrichten, die über das Netz übertragen werden.

Beim CICS Distributed Program Link (DPL) benötigt der Aufruf eines entfernten Systems lediglich dessen Name (TRID) und die Bezeichnung der COMMAREA. Eine IDL ist nicht erforderlich.

IDL (Interface Definition Language)

Die Prozedur eines Servers kann vom Klienten über eine Schnittstelle (Interface) in Anspruch genommen werden. Wenn man eine Server seitige Anwendung entwickelt, erstellt man zwei Komponenten:

- **Die Anwendung selbst, oft als „Implementation“ bezeichnet,**
- **Die Schnittstelle (Interface), welche beschreibt, wie die Anwendung von einem Klienten aufgerufen werden kann, einschließlich der Beschreibung der übergebenen Parameter.**

Die Anwendung kann in einer beliebigen Programmiersprache geschrieben werden. Die Schnittstelle wird in einer spezifischen Sprache, der IDL (Interface Definition Language) beschrieben.

Mit Hilfe eines „IDL Compilers“ wird die in der IDL beschriebene Schnittstelle übersetzt. Das Ergebnis ist ausführbarer binary Code für Skeleton und Stub. Skeleton und Stub werden somit mit Hilfe des IDL Compilers automatisch generiert. Sie müssen die Eigentümlickeiten der Schnittstelle der Server Prozedur kennen, nicht aber die Art, in der die Server Prozedur implementiert wurde. Spezifisch brauchen sie nicht zu wissen, in welcher Sprache (Cobol, C++, Java) die Server Prozedur implementiert wurde.

Die unterschiedlichen RPC Implementierungen verwenden unterschiedliche IDL Sprachen und damit unterschiedliche IDL Compiler.

Java nimmt eine Sonderstellung ein. Die Schnittstelle einer Java Klasse, z. B. die Remote Interface einer EJB, wird mit Hilfe eines Java Sprachelements, der Java Interface, beschrieben. Eine zusätzliche Java IDL ist nicht erforderlich.

Business Objekt

Business Objekte sind Repräsentationen von Dingen des täglichen Lebens.

Beispiel: Das Business Objekt „Fritz Meier“ ist eine Instanz (Ausprägung) der Objektklasse „Bankkonto“. Franz Müller, Barbara Schneider und Else Schmidt sind weitere Instanzen der Objektklasse Bankkonto.

Services der Objektklasse Bankkonto können in Anspruch genommen werden, indem man ihre Methoden aufruft. Derartige Methoden der Objektklasse Bankkonto können z.B. sein:

- **Kontostand abfragen**
- **Geld einzahlen**
- **Geld abheben**
- **Postanschrift ändern**
- **Vollmacht für Ehepartner erteilen**

usw.

Eine Methode wird aufgerufen, indem man an ihre Interface eine Nachricht schickt. Die Nachricht enthält einzelne Parameter, z.B.

- **Name/Adresse der Objektklasse, z.B. Bankkonto Fritz Meier**
- **Name der Methode, z.B. Kontostand abfragen**
- **übergebene Parameter, z.B. stand vom 1.Januar letzten Jahres**
- **Parameter der Antwort, z.B. € 638,15 .**

Beispiel
Business Objekt “Kunde”

Komponente			
	1	Name, Vorname	
	2	Titel, Anrede	
	3	Straße	
	4	PLZ	Firmenanschrift 1
	5	Ort	
	6	Straße	
	7	PLZ	Firmenanschrift 2
	8	Ort	
	9	Straße	
	10	PLZ	Privatanschrift
	11	Ort	
	12	Straße	
	13	Kunden-Nr. = f(Objekt Nr., Objekt Identifikation, ...	
	14		
	•		
	•		
	•		
	•		

ca. 500 weitere Attribute, z.B.:

- Fax Nr.
- Tel. Nr.
- e-mail Adresse
- Info über Partner
- Info über Kinder
- Zeiger auf Kinder
- Arbeitgeber
- Stellung im Betrieb
- Mitglied im Tennisclub xyz
- Rentendaten
- •
- •
- •
- •

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 Einfachvererbung.

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

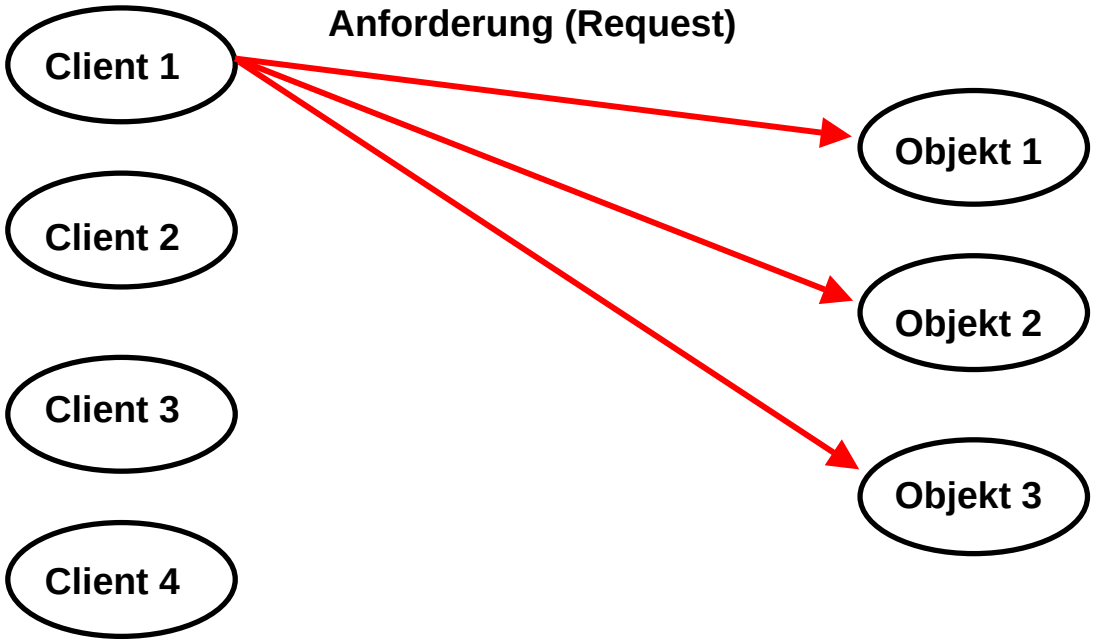
- **Sprachunabhängigkeit: Smalltalk- und C++-Objekte verstehen einander nicht.**
- **Compilerunabhängigkeit: Objekte zweier Compiler (z.B. *Watcom* C++ und *GNU* C++) können nicht miteinander kommunizieren, weil die Verwaltung interner Informationen nicht standardisiert ist.**
- **starre Kopplung zwischen 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, XP, z/OS) 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.

Objekt orientierter RPC

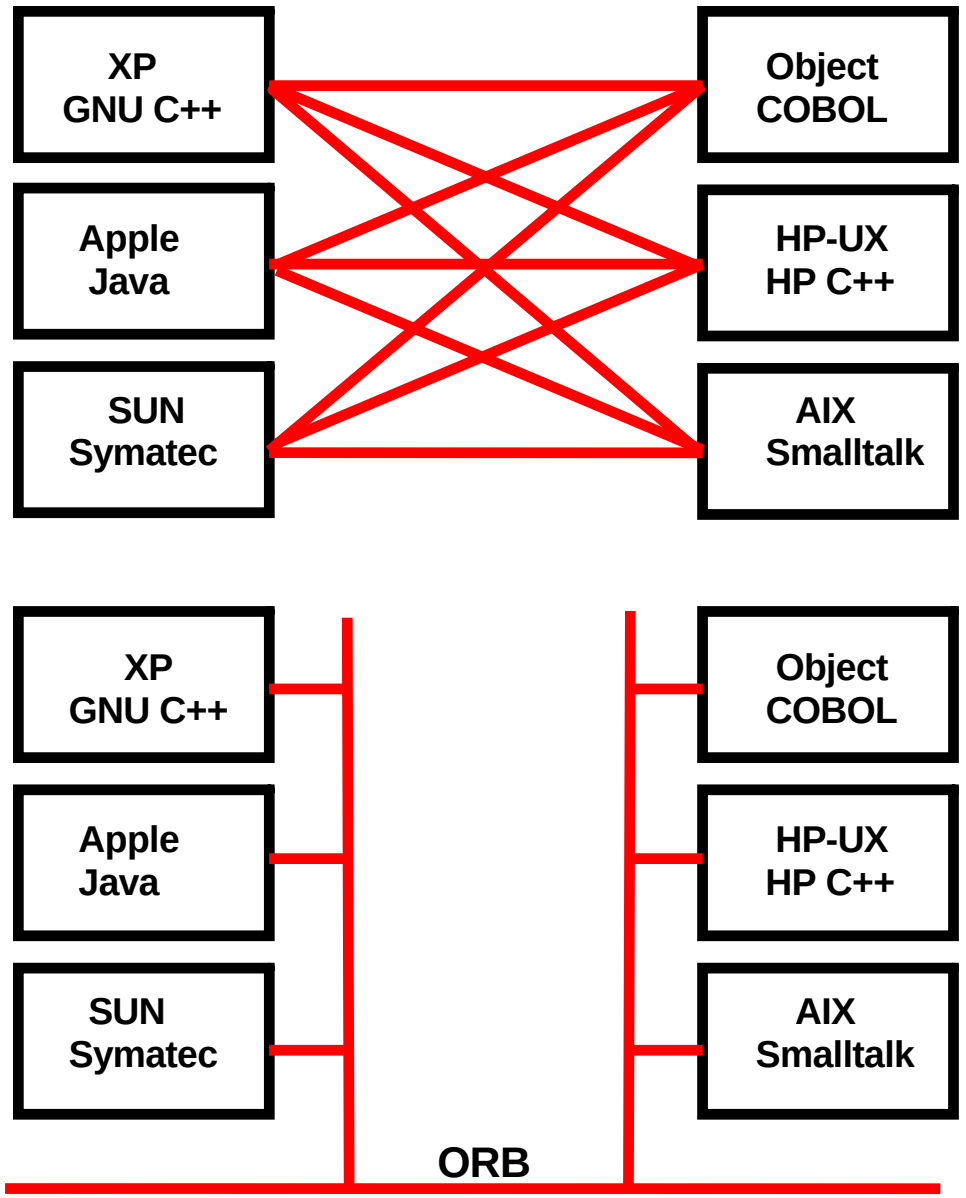


Objekte
bieten Operationen (Methoden) an, durch deren Aufruf sie zu bestimmten Aktionen veranlasst werden können

Klient
Eine Software-Einheit, die eine Operation (eine Methode) eines Objektes aufruft

In einem Objekt-orientierten RPC ist der Service (Dienstleistung eines Servers) als Objekt implementiert. Der Service wird in Anspruch genommen, indem die Methoden des Objekts aufgerufen werden.

Problematisch ist, dass Objekte in unterschiedlichen Sprachen implementiert werden, aber in binärer Form auf dem Server installiert sind. Die binäre Implementierung eines Client Objects ist ohne Anpassung nicht in der Lage mit der binären Implementierung eines Server Objects zu kommunizieren.



Ohne zusätzliche Vorrichtungen muss die Kommunikation jedes Klienten an jedes Server Objekt je nach verwendeter Sprache und darunterliegender Plattform getrennt angepasst werden.

Unter Benutzung eines Object Request Brokers (ORB) können in unterschiedlichen Programmiersprachen entwickelte binäre Objekte plattform-unabhängig miteinander kommunizieren.

Kommunikations-Alternativen für den Objekt-orientierten RPC

Object Request Broker
ORB

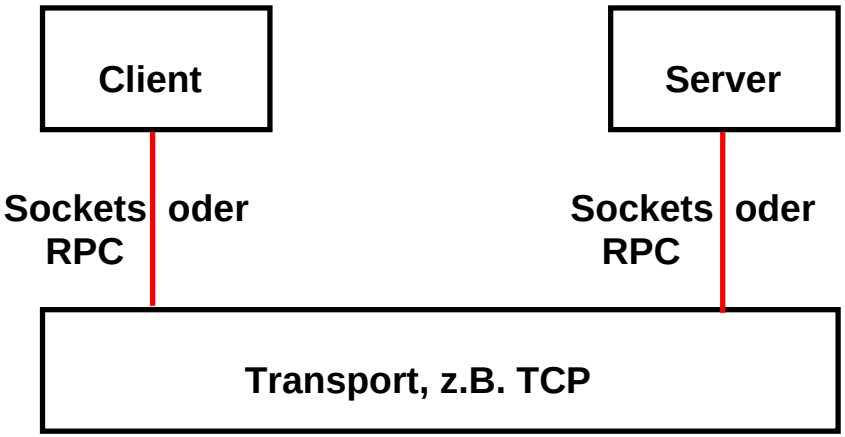
Ein ORB (Object Request Broker) ist eine Komponente eines Betriebssystems. Er ermöglicht es, daß Objekte, die in unterschiedlichen Programmiersprachen entwickelt wurden, in binärer Form miteinander zusammenarbeiten. Dies kann über Prozeß- und physische Rechnergrenzen hinweg geschehen.

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

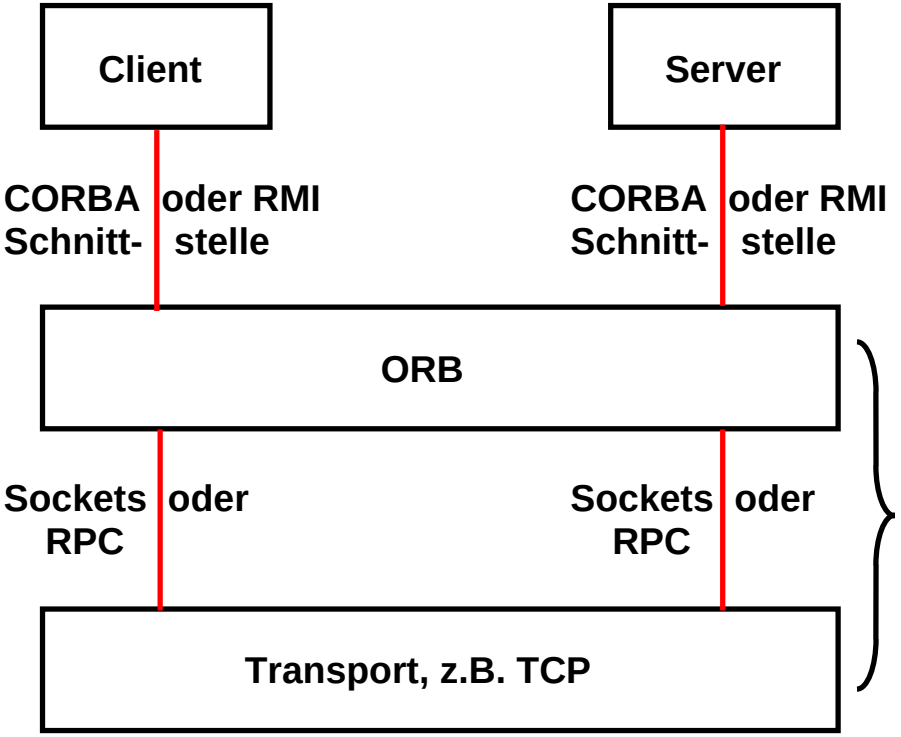
Es existieren mehrere Alternativen einen ORB zu implementieren:

- **CORBA Standard der OMG (Open Management Group),**
- **Remote Method Invocation (RMI), Teil des Java JDK der Fa. SUN**
- **DotNet der Fa. Microsoft.**

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



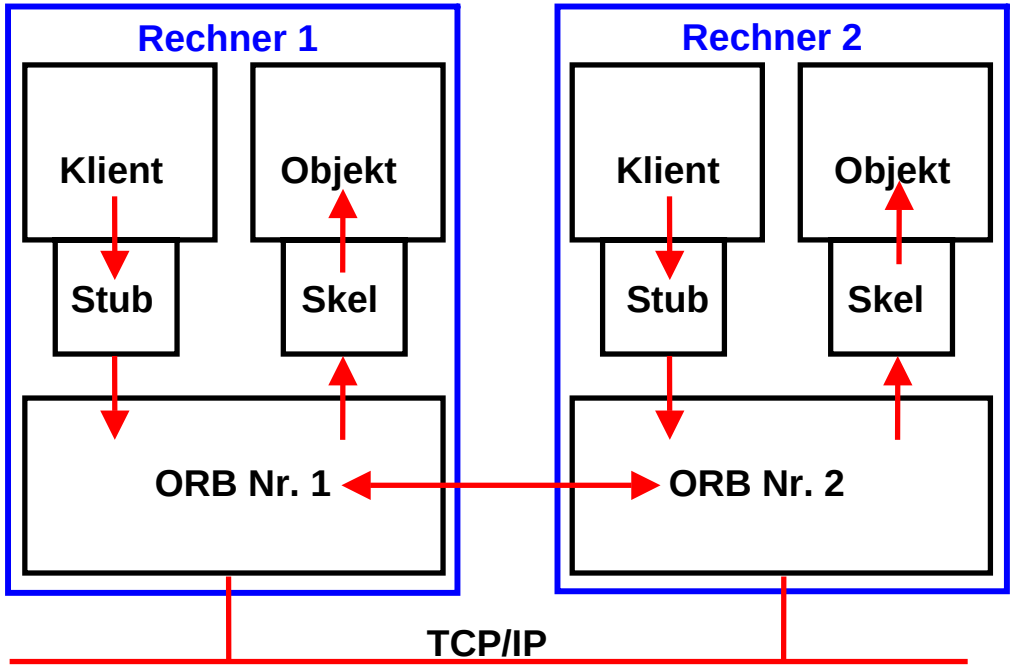
Mit Prozeduren arbeitendes Client/Server-System



Beispiele: IIOP, JRMP Dotnet

Objektorientiertes Client/Server-System

In einem Objekt orientierten Client/Server System wird ein „Objeckt Request Broker“ (ORB) zwischen Transport, Client und Server zwischen geschaltet.



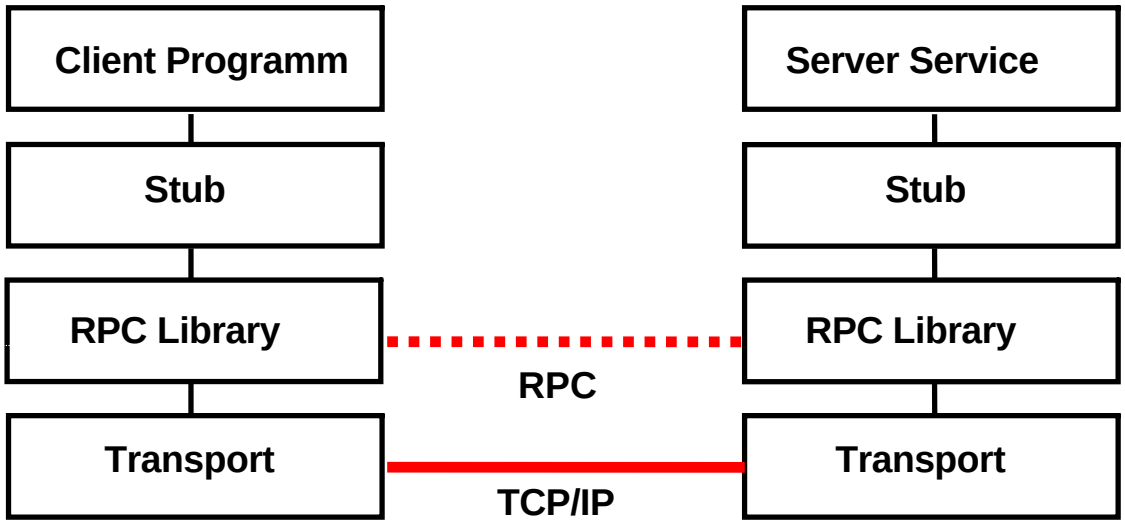
Der ORB stellt eine Schnittstelle zwischen Client und Server dar, die ähnlich der Socket- oder RPC-Schnittstelle arbeitet, aber

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

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

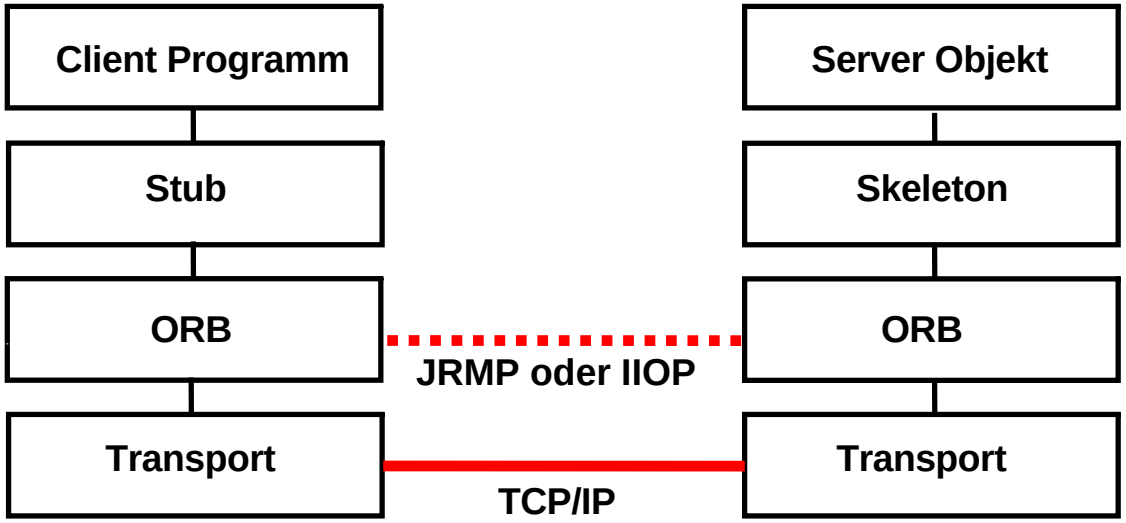
Eine eindeutige Objekt Bezeichnung (Interoperable Object Reference, IOR) wird dem Objekt bei der Entstehung zugeordnet. Der Client nimmt die Dienste eines Objektes in Anspruch, indem er es zunächst mit Hilfe seiner IOR lokalisiert, und dann einen Methodenaufruf ausführt. Der Methodenaufruf enthält:

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



Klassischer RPC

Klient ruft eine von mehreren Prozeduren eines Servers auf



Corba oder RMI

Klient ruft eine von mehreren Methoden eines Server Objektes auf.

IIOP (Internet Inter-ORB Protokoll) und JRMP (Java Remote Method Protokoll) sind auf der gleichen Ebene einzuordnen wie die Http, SOAP, 3270 oder FTP Protokolle.

Integration von unterschiedlichen Object orientierten Client/Server Protokollen

Es existieren drei bedeutende objektorientierte Client/Server Standards:

- **CORBA**
- **JEE Remote Method Invocation (RMI)**
- **DotNet von Microsoft**

Der RMI-Standard bietet zwei alternative Protokolle: JRMP und RMI/IIOP. RMI/IIOP wurde entwickelt, um CORBA und RMI miteinander zu verbinden. Einige Hersteller von JEE Software bieten beide Alternativen in ihren Produkten an. IBM hat sich entschieden, ausschließlich RMI/IIOP einzusetzen, was konform mit dem RMI Standard ist.

Die Koexistenz Charakteristika von Corba und RMI sind sehr gut.

Auf der anderen Seite sind die Microsoft DotNet Produkte sehr unvereinbar mit der Corba / RMI Welt. Es ist möglich, eine Kommunikation zwischen beiden Alternativen mittels ORB Brücken zu erreichen. Allerdings ist die Leistung nicht optimal und die Integration erfordert Expertenwissen. Aus diesem Grund ist DotNet bei größeren Unternehmen, die Java häufig verwenden, nicht sehr beliebt. Es wird vor allem in kleineren Unternehmen eingesetzt, die hauptsächlich Microsoft Server Produkte einsetzen.

Web Services und ihr SOAP RPC sind eine moderne Alternative zu RMI und Corba.

CORBA
Common Object Broker Architecture

CORBA ist ein Standard der OMG (Open Management Group). Die OMG ist ein 1989 gegründeter, internationaler, nicht profit-orientierter Zusammenschluß von zunächst 8 (Anfang 1996: Ca. 600) Softwareentwicklern, Netzbetreibern, Hardwareproduzenten und kommerziellen Anwendern von Computersystemen (ohne Microsoft).

CORBA unterstützt viele Sprachen, darunter C/C++, COBOL, PLI, ADA und Java. CORBA Produkte werden von zahlreichen Herstellern implementiert. Für alle von CORBA unterstützten Sprachen existiert eine einheitliche Interface Definition Language (IDL). IDL ist der OMG Standard um Sprach-neutrale APIs zu definieren. IDL ermöglicht eine Platform-unabhängige Beschreibung der Interfaces von verteilten (distributed) Objekten.

Literatur

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

http://www.cs.wustl.edu/~schmidt/corba-overview.html
http://www.blackmagic.com/people/gabe/iiop.html

RMI
Remote Method Invocation

Remote Method Invocation (RMI, deutsch etwa „Aufruf entfernter Methoden“), ist der Aufruf einer Methode eines entfernten Java Objekts und realisiert die Java-eigene Art des Remote Procedure Calls. „Remote“ bedeutet dabei, dass sich das Objekt in einer anderen Java Virtual Machine befinden kann, die ihrerseits auf einem entfernten Rechner oder auf dem gleichen lokalen Rechner laufen kann. Dabei sieht der Aufruf für das aufrufende Objekt (bzw. dessen Programmierer) genauso wie ein lokaler Methoden Aufruf aus; es müssen jedoch besondere Ausnahmen abgefangen werden, die zum Beispiel einen Verbindungsabbruch signalisieren können.

Auf der Client-Seite kümmert sich der Stub um den Netzwerktransport. Vor dem Erscheinen der Java Standard Edition (JSE) in Version 1.5.0 war der Stub eine mit dem RMI-Compiler `rmic` erzeugte Klasse. Seit der Version 1.5 ist es nicht mehr notwendig, den RMI-Compiler aufzurufen. Das Erstellen des Stubs wird von der Java Virtual Machine übernommen.

Für die erste Verbindungsaufnahme werden aber die Adresse des Servers und ein Bezeichner (z.B. eine RMI-URL) benötigt. Für den Bezeichner liefert ein Namensdienst auf einem Server eine Referenz auf das entfernte Objekt zurück. Damit dies funktioniert, muss sich das entfernte Objekt im Server zuvor unter diesem Namen (IOR oder String Name) beim Namensdienst registriert haben. Der RMI Namensdienst wird über statische Methoden der Klasse java.rmi.Naming angesprochen. Der Namensdienst ist als eigenständiges Programm implementiert. Es existieren zwei Arten des Namensdienstes, der RMI Registry für lokale Netze und der „Java Naming and Directory Service“ (JNDI) für weltweite Netze.

Eine Client/Server Anwendung, bei der sowohl der Client als auch der Server in Java programmiert sind, kann sowohl RMI als auch eine andere RPC Art verwenden (z.B. CORBA oder DCE). RMI ist aber ein besonders einfaches Verfahren, Client/Server Anwendungen zu erstellen.