

Objekt

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

Objekt-Klasse (Objekt-Typ)

spezifiziert Operationen und Datenstrukturen für gleichartige Objekte

Typ Abstrakte Spezifikation der Funktionalität

Klasse Implementierung dieser Funktionalität

Objekt-Instanz, Objekt-Exemplar

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

```

#include <iostream.h>
const double pi = 3.14159;

// Definition der Klasse box_car
class box_car {
public: double height, width, length; // Parameter Definition
double volume () {
return height * width * length; // Methode
}
};

// Definition der Klasse tank_car
class tank_car {
public: double radius, length; // Parameter Definition
double volume () {
return pi * radius * radius * length; // Methode
}
};

int main() {

// Erstellung von je einer Instanz der beiden Klassen
box_car x; x.height = 10.5; x.width = 9.5; x.length = 40.0;
tank_car y; y.radius = 3.5; y.length = 40.0;

// Aufrufe der Methoden der beiden Klassen
cout << "the volume of the boxcar is " << x.volume () << endl
<< "the volume of the tankcar is " << y.volume () << endl ;
return 0;
}

```

Objekte

Objekteigenschaften

Vererbung (Inheritance)
Abstract Data Typing
Polymorphismus

Charakteristika eines Objektes

Name (Identity)
Zustand (State, representation of attributes by internal data structures)
Verhalten (Behaviour, set of allowed operations, functions, methods)

Klasse (Class)

Template für die Konstruktion eines Objektes
Objekte sind Ausprägungen (instances) einer Klasse

Data Abstraction

Kapselung des Objektzustandes und -verhaltens

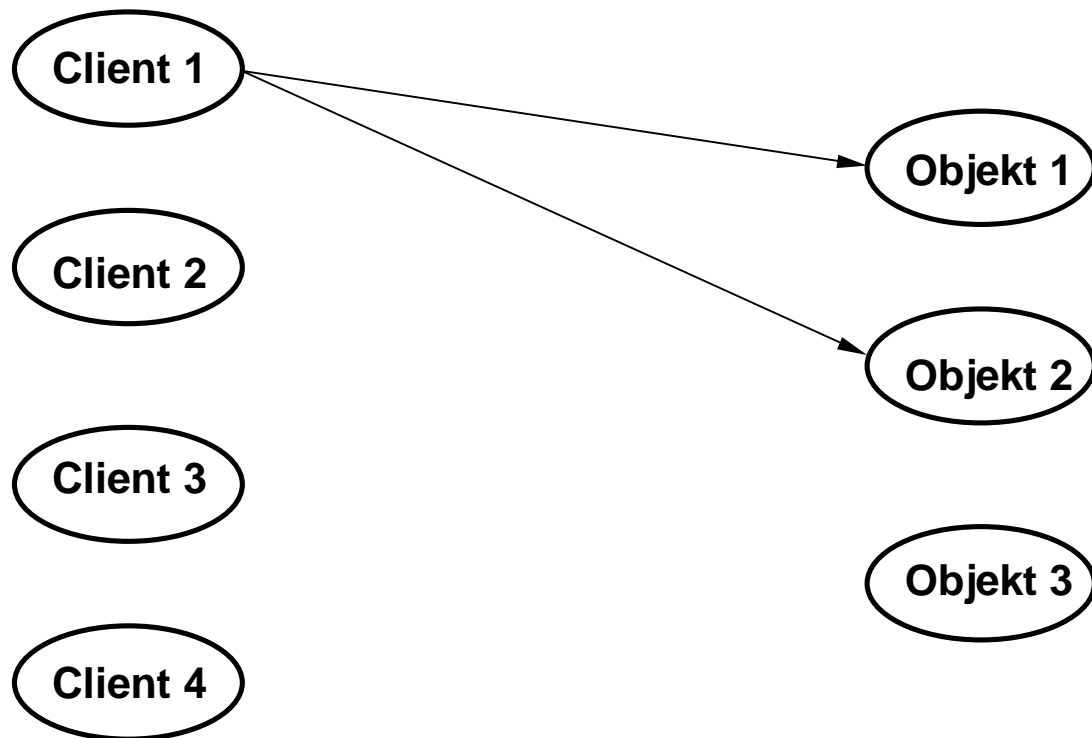
Vererbung

erlaubt es, neue Klassen von existierenden abzuleiten

Polymorphismus

Unterschiedliche abgeleitete Objekte können die gleiche Methode auf ihre eigene Art behandeln

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

Wiederverwendbarkeit von Code (1)

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.

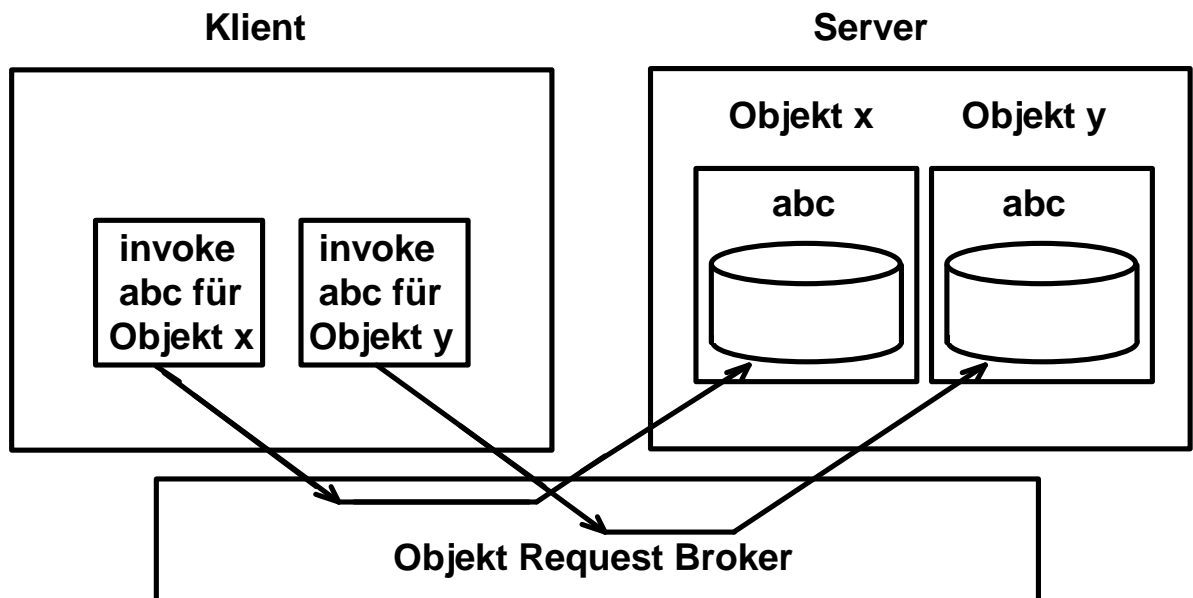
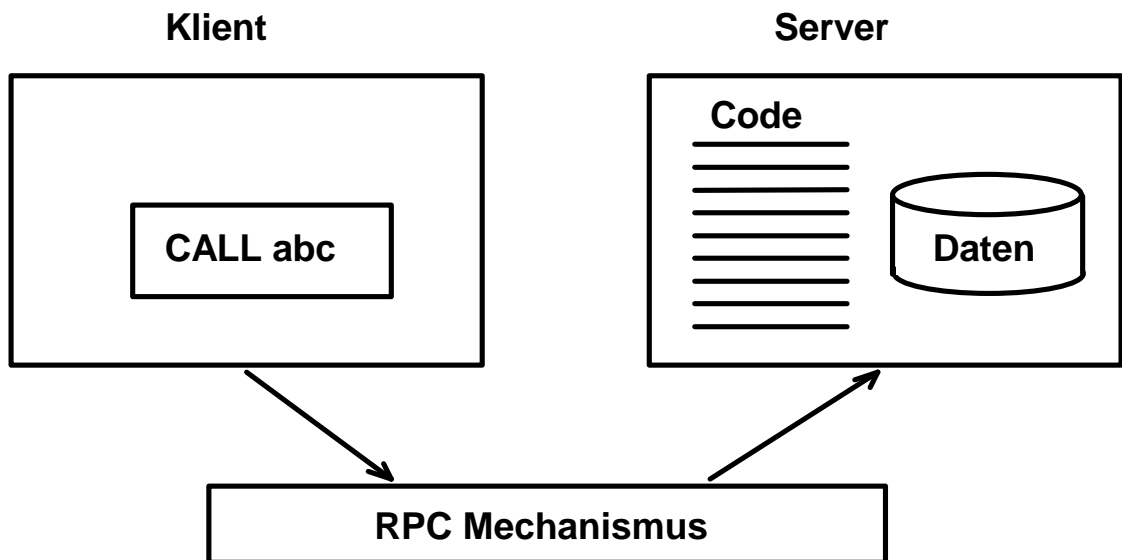
Wird nur in einer einzigen Sprache mit identischem Compiler entwickelt ist die Wiederverwendbarkeit von Klassen am leichtesten erreichbar. Beim Einsatz mehrerer Programmiersprachen muß die Objektphilosophie der Programmiersprache beachtet werden. Z.B. unterstützt C++ die Mehrfachvererbung, Java aber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

In verschiedenen Dialekten einer Sprache implementierte Klassen sind nur als Quelltext kompatibel, nicht in binärer Form als EXE oder DLL Dateien.

Wiederverwendbarkeit von Code (2)

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

- **keine Sprachunabhängigkeit: Smalltalk- und C++-Objekte verstehen sich nicht**
- **keine 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. Der Grund: C++-Compiler benutzen Konstanten beim Zugriff auf Daten und Methoden**
- **Beschränkung auf einen Prozeßraum (Objekte können nicht über Prozeßgrenzen hinweg kommunizieren)**



Der Klient sendet eine Anforderung (request) an ein Objekt. Die Methode des Objektes befriedigt die Anforderung.

Der Objekt Request Broker (ORB) ermöglicht über definierte Schnittstellen den Methodenaufruf von entfernten Objekten. Er findet das Objekt und übermittelt die Daten, welche die Anforderung darstellen.

Die Schnittstelle zwischen Klient und Objekt ist unabhängig von

- wo das Objekt sich befindet
- in welcher Sprache es implementiert wurde

Unterschiedliche ORB Architekturen

CORBA Standard der OMG

DCOM/ActiveX der Fa. Microsoft

**Remote Method Invocation (RMI), Teil des JDK 1.1
der Fa. SUN**

**Common Gateway Interface (CGI), Erweiterung
des HTTP Protokolls**

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 Zusammenwirken von Anwendungen verschiedener Hersteller unabhängig von Betriebssystem, Programmiersprache und Hardware ermöglichen soll.

Object Request Broker (ORB)

Universelles Kommunikationsmedium für beliebig geartete Objekte in verteilten heterogenen Systemen, Kernstück der OMA, 1991 von OMG in Version 1.1 spezifiziert.

Common ORB Architecture (CORBA)

Von OMG erarbeiteter Standard, aktuelle Version 2.0 (Sommer 1995).

CORBA: Eigenschaften

- **Objektorientierung**

Grundlegende Einheiten sind Objekte als beliebige, eindeutig identifizierbare Einheiten (nicht unbedingt im Sinne einer Programmiersprache).

- **Verteilungstransparenz**

Gleiche Zugriffsmechanismen für lokale und entfernte Objekte, Aufenthaltsort der Objekte ist Client i.A. unbekannt.

- **Effizienz**

Von OMG festgelegte Architektur für ORB ermöglicht effiziente Implementierungen (z.B. bei lokalen Objekten kaum Nachteile gegenüber normalem Funktionsaufruf).

- **Hardware-, Betriebssystem- und Sprachunabhängigkeit**

Komponenten von CORBA-Programmen können auf verschiedenen Betriebssystemen, Hardwarearchitekturen und mit verschiedenen Programmiersprachen realisiert werden.

- **Offenheit**

Über ORB können Programme verschiedener Hersteller kooperieren.

CORBA Implementierungen

Verfügbare ORB's:

DEC	ObjectBroker
IBM	Component Broker (früher DSOM)
IONA	Orbix
HP	ORB Plus
SunSoft	Solaris NEO
Inprise	Visigenic Object Broker
andere	

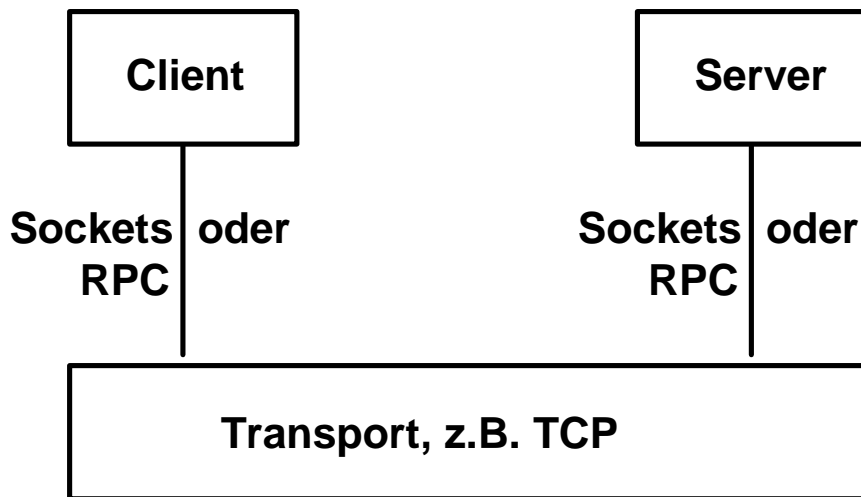
Unterstützte Sprachen:

C++
Smalltalk
COBOL
Java
andere

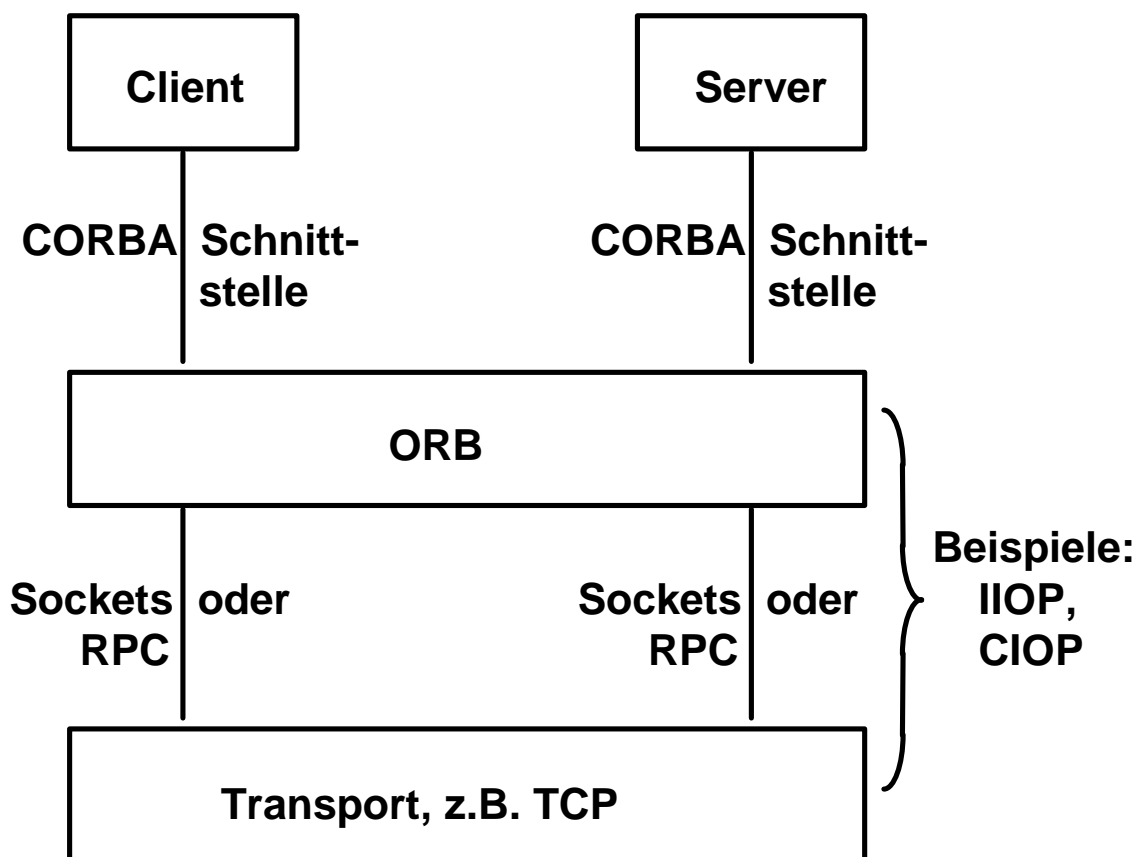
Sun JOE (Java Objects Everywhere) ist ein Standard, der Java Applets mit CORBA-Anwendungen kommunizieren läßt. Java Applets, die mit einem CORBA ORB zusammenarbeiten (ORBlets), werden mit einem OMG IDL Compiler übersetzt

Architektur des ORB wird beschrieben im OMG-Dokument

***CORBA: Common Object Request Broker
Architecture and Specification
(Version 2.0, Juli 1995).***



Mit Prozeduren arbeitendes Client/Server-System



Objektorientiertes Client/Server-System

ORB benutzt einen RPC oder Socket Service, z.B.

SunSoft NEO

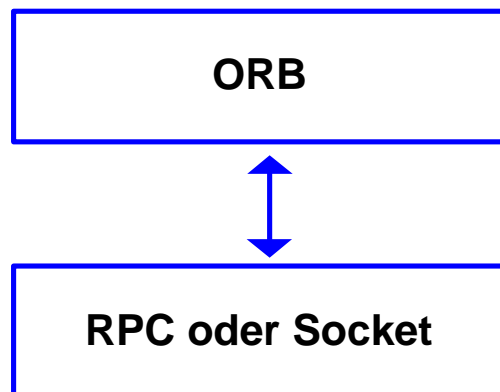
ONC RPC

HP ORB Plus

DCE RPC

IBM DSOM

**Sockets über TCP/IP,
NetBIOS oder IPX**



Corba beinhaltet Funktionen der OSI-Schichten 7 und 6

Object Request Broker:

- **Wird von Objekten zur Kommunikation mit anderen Objekten (im selben oder einem anderen, auch entfernten Programm) genutzt.**
- **Ermöglicht transparentes Weiterleiten des Operationsaufrufs zum Zielrechner und zum aufgerufenen Objekt, unabhängig vom dort verwendeten Betriebssystem und von der Sprache, in der das Objekt implementiert ist.**
- **Übergabe des Operationsaufrufs an den ORB wird durch das Language-Mapping der verwendeten Programmiersprache geregelt.**
- **Architektur des ORB wird beschrieben im OMG-Dokument**

***CORBA: Common Object Request Broker
Architecture and Specification
(Version 2.0, Juli 1995).***

GIOP, IIOP

General Inter-ORB Protocol, Internet Inter-ORB Protocol

GIOP spezifiziert:

- **8 Nachrichtentypen für den Datenaustausch Client/Server**
- **Common Data Representation (CDR)**

CDR spezifiziert (mit Hilfe der IDL):

- **Primitive und zusammengesetzte Datentypen (optimiertes XDR Verfahren)**
- **„Pseudo Objects“, u.a. „Interoperable Object References (IOR)“.**

Die IOR enthält eine eindeutige Bezeichnung eines Objektes innerhalb eines verteilten ORB Systems. Sie enthält eine Datenstruktur mit:

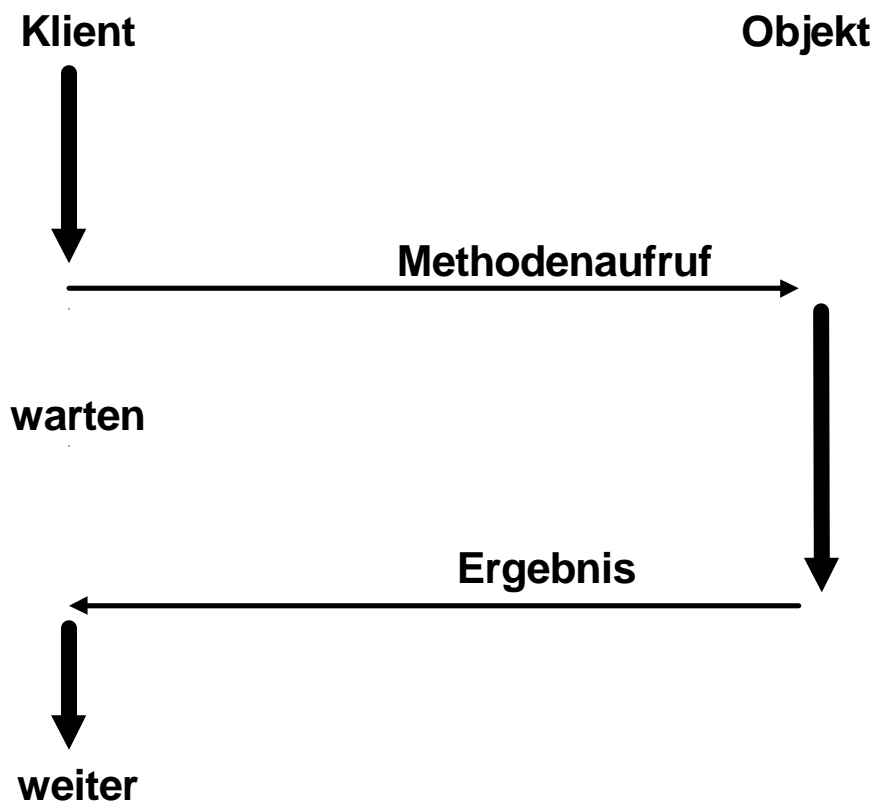
- **Referenz auf das Objekt selbst**
- **Transport Protokoll abhängige Information, welche die Art und Weise der Verbindungsaufnahme beschreibt.**

IIOP ist die TCP/IP spezifische Implementierung des GIOP Protokolls. Verwendet Sockets für den Nachrichtentransport.

Synchroner Methodenaufruf

CORBA arbeitet synchron.

Der Klient wartet, bis das Ergebnis des Methodenaufrufes verfügbar ist.



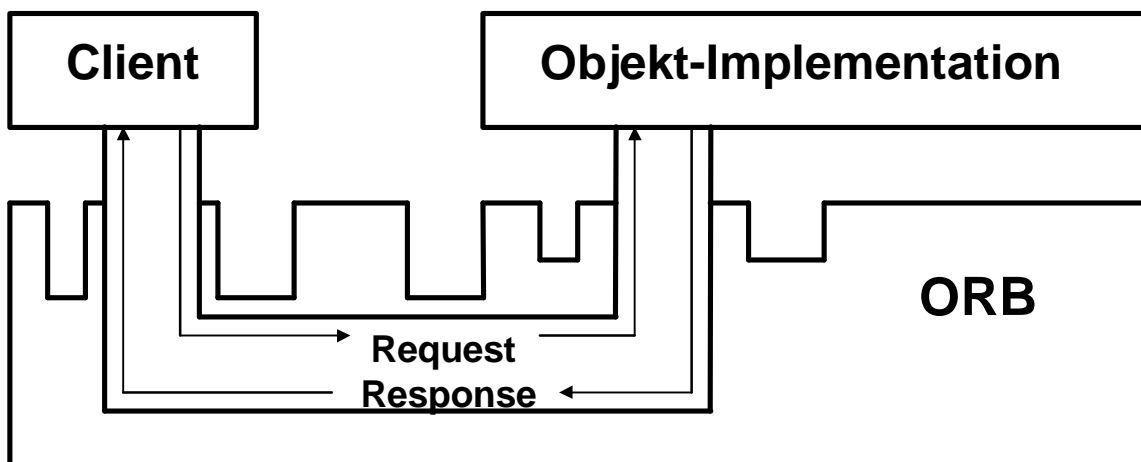
CORBA 2.0

Architektur:

Erhält ORB Client-Request, so

- lokalisiert er die Objekt-Implementation im Netz (anhand der Objekt-Referenz),
- überträgt Daten zum Zielrechner,
- übergibt Request an Objekt-Implementation,
- überträgt Response an Client.

Bei unterschiedlichen Datendarstellungsformaten auf Sender- und Empfängerseite übernimmt ORB auch die Konvertierung.



Object Reference

Zeiger auf ein Corba-Objekt, das irgendwo im Netzwerk existiert

Corba Client

**Programm, das ein Corba-Objekt aufruft
Softwareeinheit, die von einem Objekt eine Operation aufrufen
möchte. Ein Client kann (gleichzeitig oder nacheinander) mehrere
Objekte nutzen.**

Objekt

**Im Rechner repräsentiert durch Objekt-Implementation und private
Daten der jeweiligen Objekt-Instanz. Ein Objekt kann gleichzeitig von
mehreren Clients genutzt werden. Ein Objekt kann selbst Client
eines anderen Objekts sein. Ein Objekt wird über seine Objekt-
Referenz identifiziert.**

Objekt-Implementation:

Programmcode, der das Verhalten des Objekts definiert.

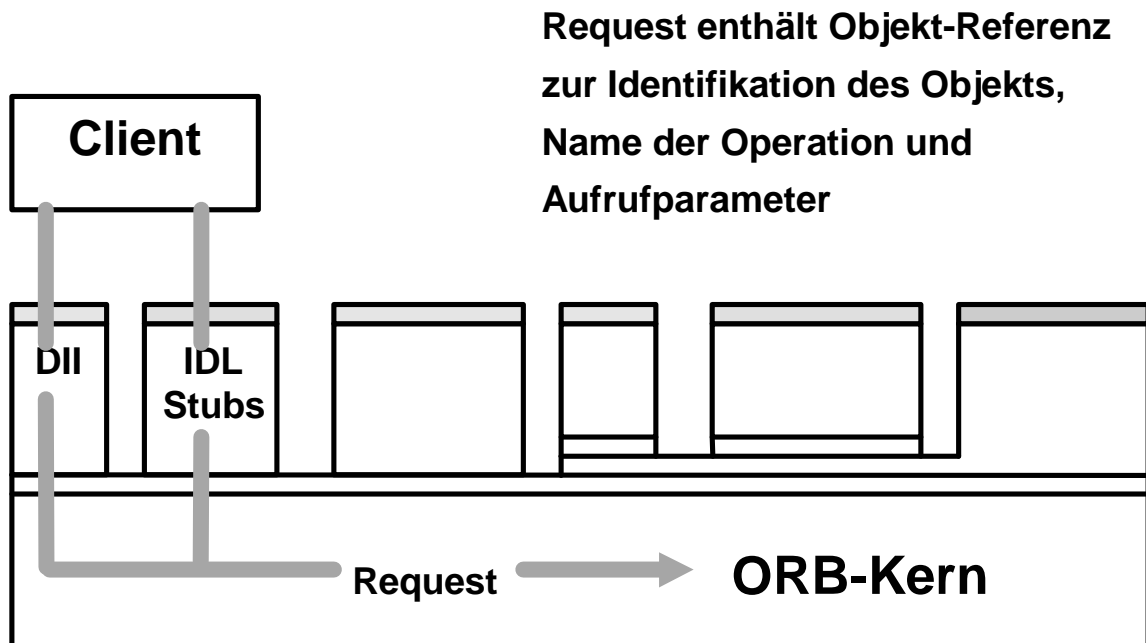
Corba Server

**Implementiert eine Anzahl von Corba-Objekten
Programm, das eine oder mehrere Objekt-Implementationen
bereitstellt.**

CORBA 2.0

Architektur:

Versenden eines Requests durch einen Client



Es gibt zwei Wege, einen Request abzuschicken:

1. **Über einen IDL-Stub**
Dazu muß das Interface des angeforderten Objekts zum Zeitpunkt der Entwicklung des Client-Programms bekannt sein. Der Operationsaufruf gleicht dann formal einem lokalen Funktionsaufruf
2. **Über das Dynamic Invocation Interface**
Dabei wird dem Client zuerst die nötige Information über das Interface bereitgestellt. Danach wird eine Operation ausgewählt und aufgerufen.

Für den ORB sind beide Arten äquivalent.

Stubs

IDL Stubs dienen als lokale Proxies für entfernte Methoden

Der Stub marshals Parameter

Die IDL Schnittstellen Definition beschreibt Methoden und Attribute einer Klasse

IDL Stubs stellen programmiersprachenabhängige Header Dateien zur Verfügung, die zur Übersetzungszeit gebunden werden, um Methodenaufrufe in der Programmiersprache des Klienten zu ermöglichen

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

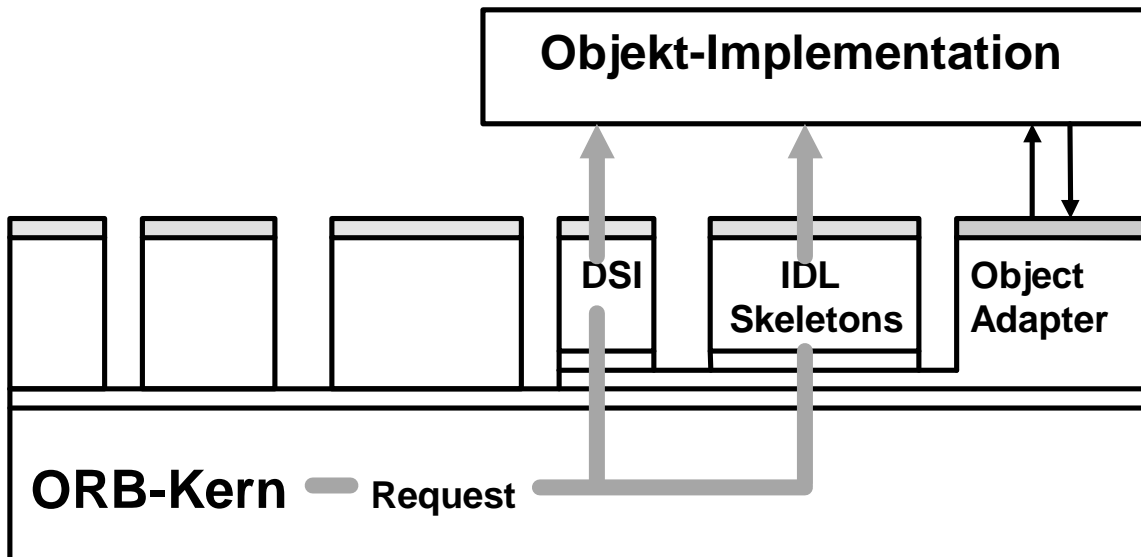
- **Objektreferenz**
- **Methodennamen**
- **Parameter**
- **Kontext**

Der Kontext enthält Information über die Position des Aufrufers und nimmt Fehlermeldungen entgegen

Die Übermittlung der Objektreferenz an den Klienten ist die Aufgabe des Benutzers

CORBA 2.0

Weiterleitung eines Requests zur Objekt-Implementation



Auch hier zwei Möglichkeiten:

1. Übergabe des Requests an ein IDL-Skeleton

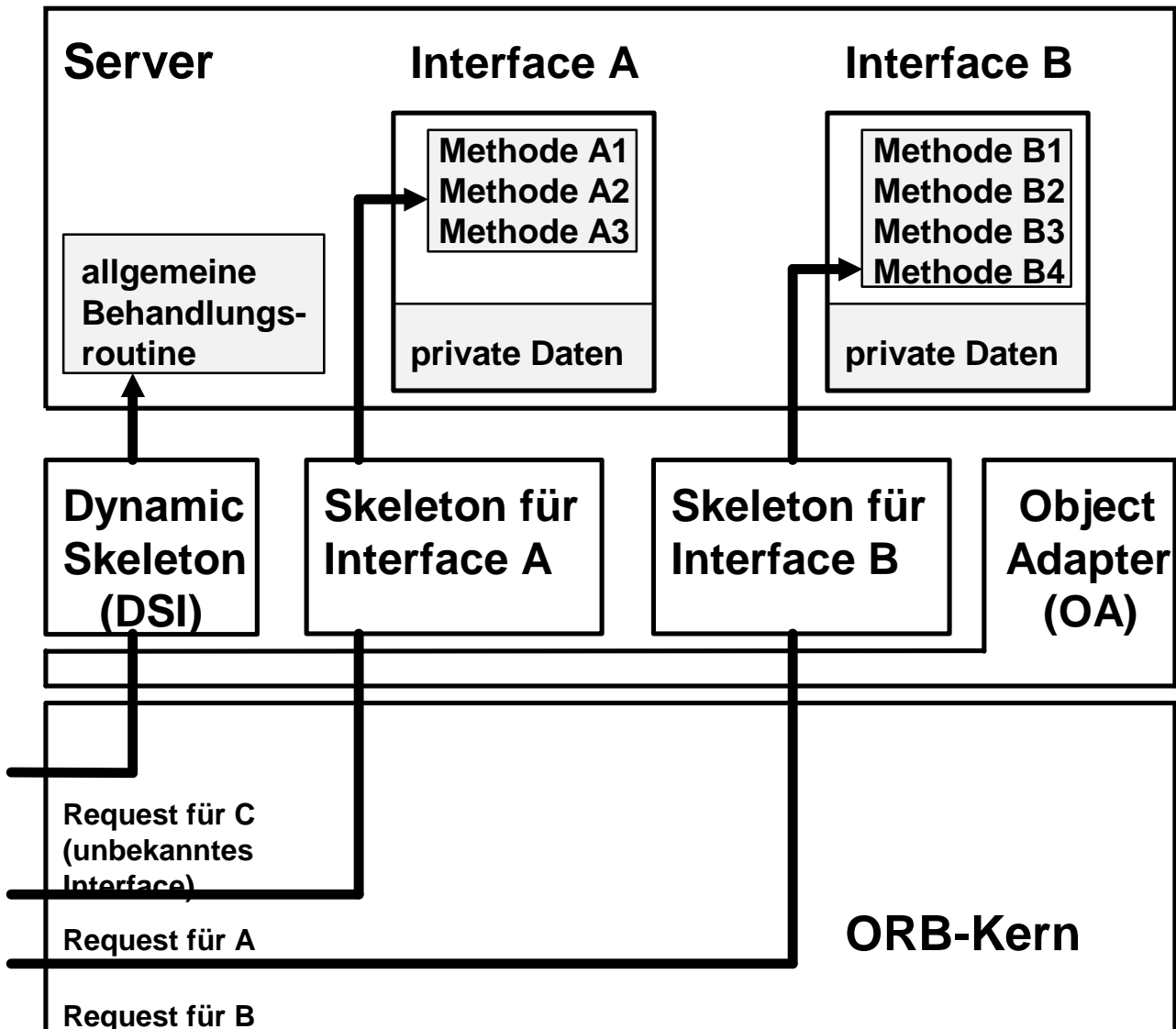
Normalfall: Der durch die Objekt-Referenz identifizierte Skeleton ruft die im Request angeforderte Operation auf

2. Übergabe an Dynamic Skeleton Interface

Wird genutzt, wenn für angefordertes Objekt kein Skeleton existiert. DSI ermittelt erst Interface des Objekts, ordnet die im Request mitgelieferten Parameter der angeforderten Operation zu und ruft die Operation auf (Nicht auf allen CORBA-Plattformen implementiert).

CORBA 2.0

Aufbau eines Servers:



Zuweisung der Requests an Skeletons bzw. DSI erfolgt durch ORB und Object Adapter.

Allgemeine Behandlungsroutine muß durch Server bereitgestellt werden.

CORBA Stubs und Skeletons

Klienten- und Serverobjekte sind mit dem ORB über die IDL-Schnittstelle verbunden (Interface Definition Language).

Die IDL-Schnittstelle erlaubt es einem Objekt, seine Methoden und Parameter an andere Objekte mit Hilfe des ORB zu übergeben.

Im einfachsten Fall erfolgt die Kommunikation über Stubs unter Verwendung eines darunterliegenden RPC- oder Socketmechanismus. Der Serverstub wird auch als „Skeleton“ bezeichnet.

Stubs werden mit Hilfe des IDL-Compilers für die entsprechenden Objekte generiert. Stubs bewirken den Nachrichtentransport, Behandlung von Ausnahmesituationen usw.

Zusätzlich zu Stubs kennt CORBA ein „Dynamic Invocation Interface“ (DII, auf der Client-Seite), sowie ein korrespondierendes „Dynamic Skeleton Interface“ (DSI, auf der Server-Seite). Hierbei wird einem Objekt erst zur Laufzeit Information über ein anderes Objekt mitgeteilt.

Auf der Server-Seite besteht kein Unterschied zwischen statischen und dynamischen Methodenaufrufen.

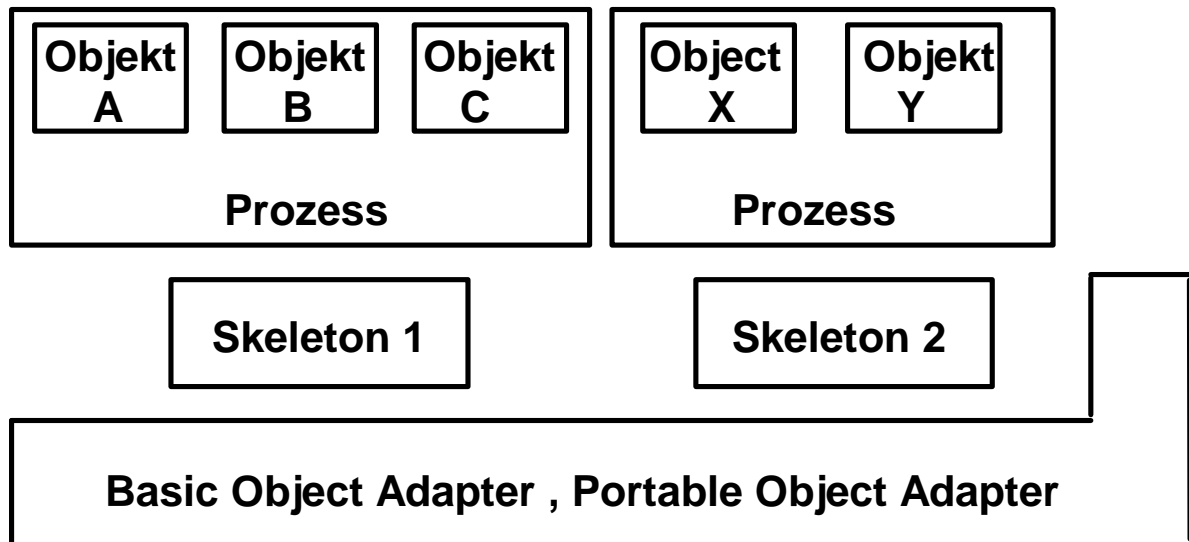
IDL-Skeletons stellen für jeden Dienst, den ein Objekt bereitstellt, eine statische Schnittstelle zur Verfügung.

Dynamic Skeleton Interface

Aufruf von Objekten, die keine Skeletons besitzen.

Einsatz vor allem für Methodenaufrufe zwischen ORB's verschiedener Hersteller.

CORBA Server



CORBA Unterscheidet zwischen Server und Objekt

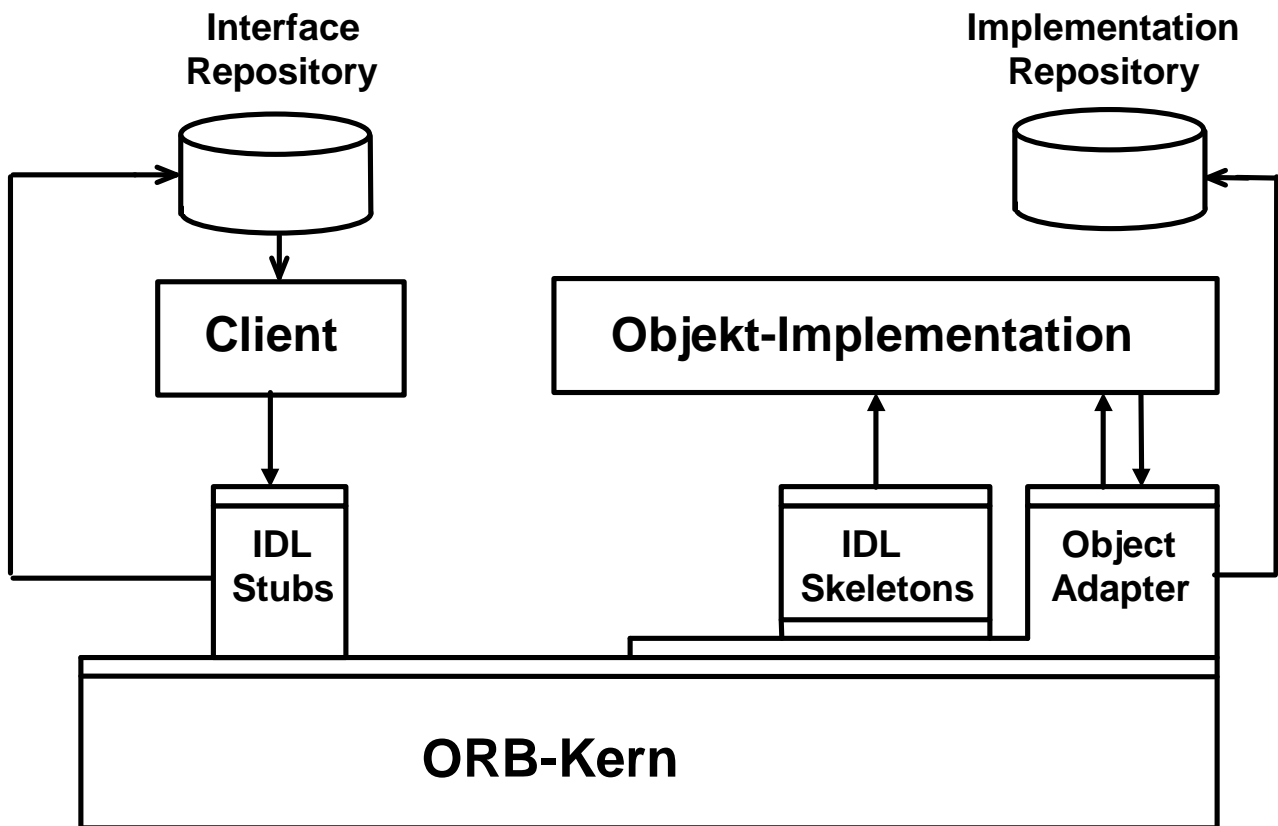
Objekt implementiert Schnittstelle


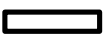


Server beinhaltet typischerweise mehrfache Objekte (shared server), z.B. alle von der gleichen Klasse

Der Server wird erstmalig aktiviert, wenn ein Request für eines seiner Objekte eintrifft. Weitere Requests werden auf einer „first come, first serve“ Basis abgearbeitet. Typischerweise 1 Thread pro Objekt.

(Ein „Persistent Server“ ist ein shared Server, der die ganze Zeit aktiv ist, z.B. für Transaktionsverarbeitung.)

Wichtige Schnittstellen des ORB zum Client und zur Objekt-Implementation

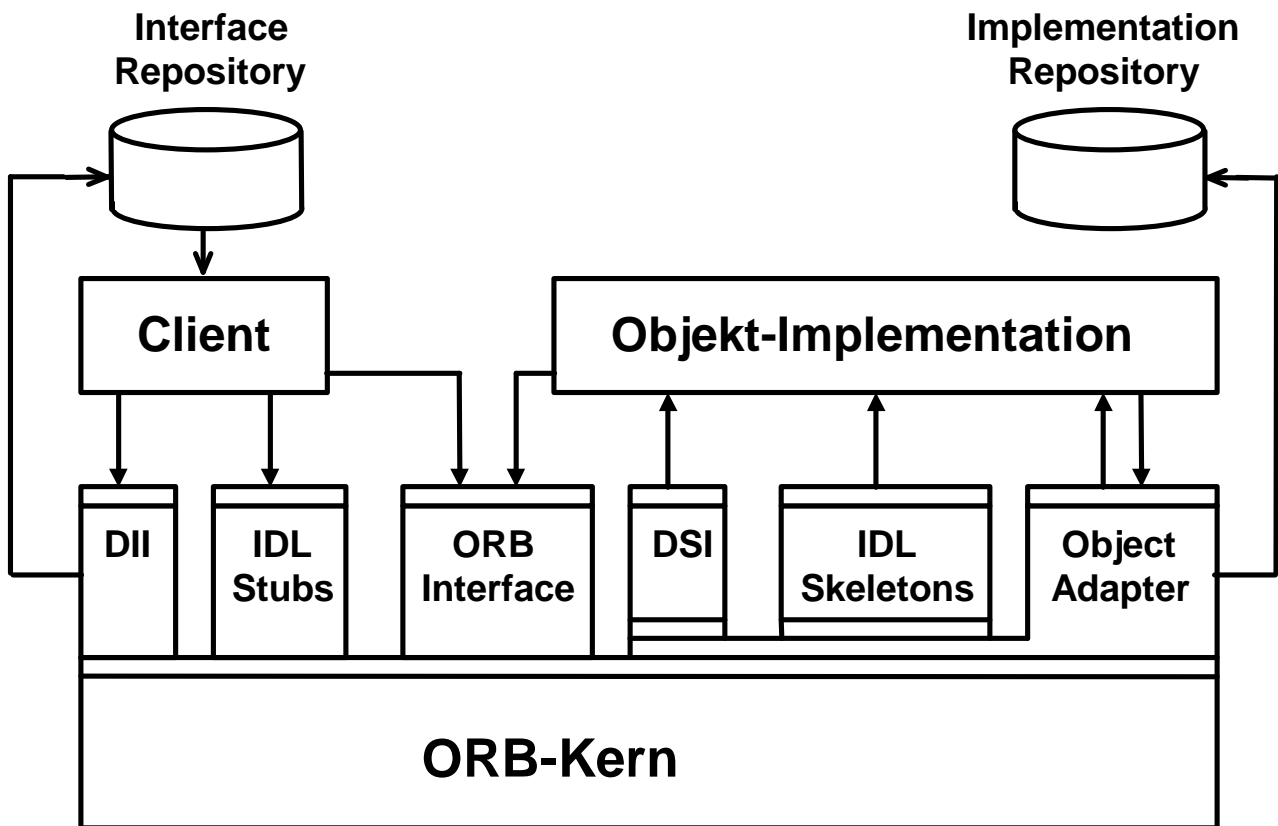


-  Standardisierte Schnittstelle (identisch für alle ORBs)
-  Mehrere, jeweils auf einen Objekttyp spezialisierte Schnittstellen
-  Es kann mehrere Object Adapter geben
-  ORB-spezifische Schnittstelle

IDL Stubs
IDL Skeletons }
}

Objektspezifische Schnittstellen, werden von IDL-Compiler generiert

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

Interface Repository

On-Line Datenbank, enthält Beschreibungen der Schnittstellen aller Objekte:

- **Welche Methoden sind verfügbar**
- **Methoden Parameter**
- **Welche Attribute hat das Objekt**

Beschreibt Funktionen, über die ein Server Objekt verfügt. Überprüft Parameter beim Aufruf einer Methode.

Typisch 1 Eintrag pro Objekt Klasse.

Implementation Repository

Enthält Information über alle implementierten Objekte

Jedes Objekt wird durch eine eindeutige ID (Objekt Referenz) identifiziert.

Typisch 1 Eintrag pro Objekt

Interface Repository

Bestandteil des ORB, Run-Time verteilte Datenbank.

**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

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

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

Implementation Repository

Bestandteil des ORB.

Enthält Run-Time Informationen, die der Auffindung eines (anhand einer Objekt-Referenz identifizierten) Objekts im Netz ermöglichen. Macht damit Objekt-Lokationen für Clienten transparent.

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).

Object Adapter Schnittstelle zu ORB-Dienstleistungen, die der Server nutzen kann

ORB Interface Schnittstelle zu ORB-Dienstleistungen, die

- von Client und Server benutzt werden können
-
- für alle ORB-Implementierungen identisch sind

Interface Repository verwaltet IDL-Definitionen, die zur Laufzeit verfügbar sind und vom (Client-) DII benutzt werden. Wird außerdem vom ORB benutzt, um Anforderungen weiterzuleiten

Implementation Repository verwaltet Informationen, die der ORB benötigt, um den Server zu lokalisieren

Auf einem Corba-Rechner läuft ständig ein ORB-Prozess (Daemon). Dieser unterhält ein Schnittstellen-Repository, welches die IDL-Schnittstellen aller Objekte, die der ORB kennt, abspeichert.

Corba-Objekte müssen keine laufenden Prozesse sein. Der ORB ist in der Lage, ein Objekt bei Bedarf zu aktivieren. Hierzu unterhält der ORB ein Implementation Repository. Dieses ermöglicht die Aktivierung eines angesprochenen Objektes und die Übergabe des ursprünglichen Requests.

Bei statischen Aufrufen (static invocation; über Stubs und Skeletons) erfolgt Type-Checking durch den Compiler. Die Schnittstellendefinitionen der aufgerufenen Objekte müssen zur Compile-Zeit vorhanden sein.

Bei dynamischen Aufrufen (dynamic invocation) greift der Client auf Objekte zu, die er zur Laufzeit entdeckt.

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,**
- **...**

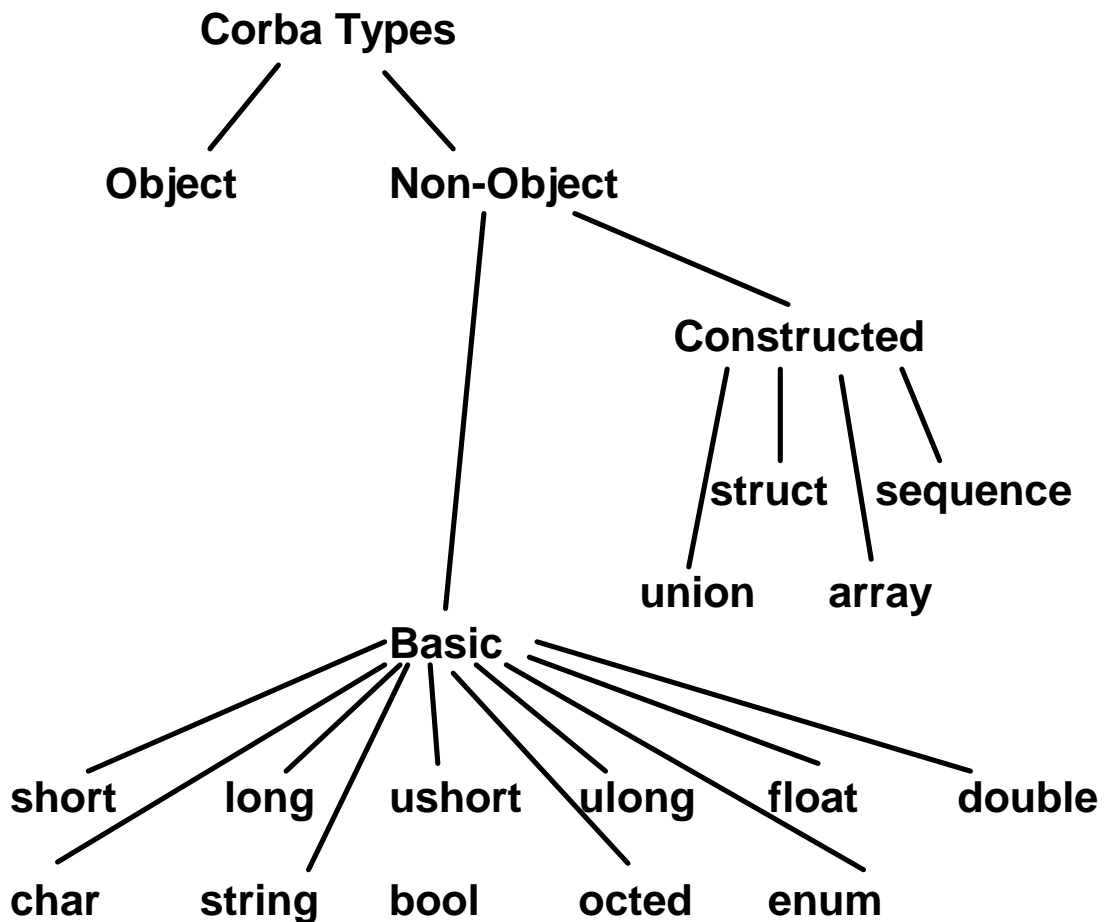
CORBA 2.0

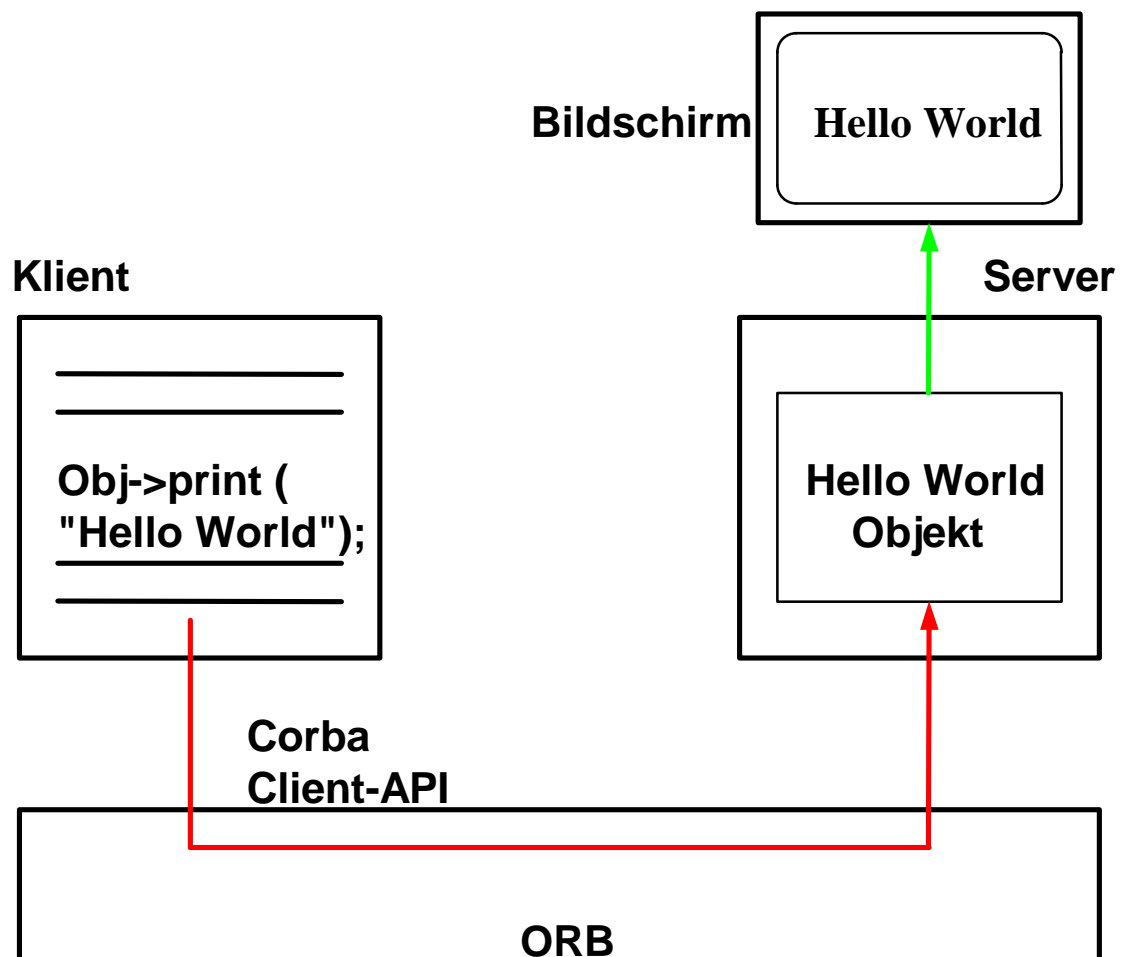
Language Mappings:

- 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.
- Sollen die Zugriffsmöglichkeit auf Objekte unabhängig von der bei ihrer Entwicklung bzw. der Entwicklung der Client- und Server-Programme benutzten Programmiersprache garantieren.
- Sind zum Teil von OMG standardisiert (für C, C++, Java und Smalltalk). Weitere Standards sind z.Z. in Arbeit (für COBOL, Ada, Fortran und PL/1).

CORBA Datentypen

beschreiben die Werte von CORBA Parametern, Attributen, Exceptions und Return Werten



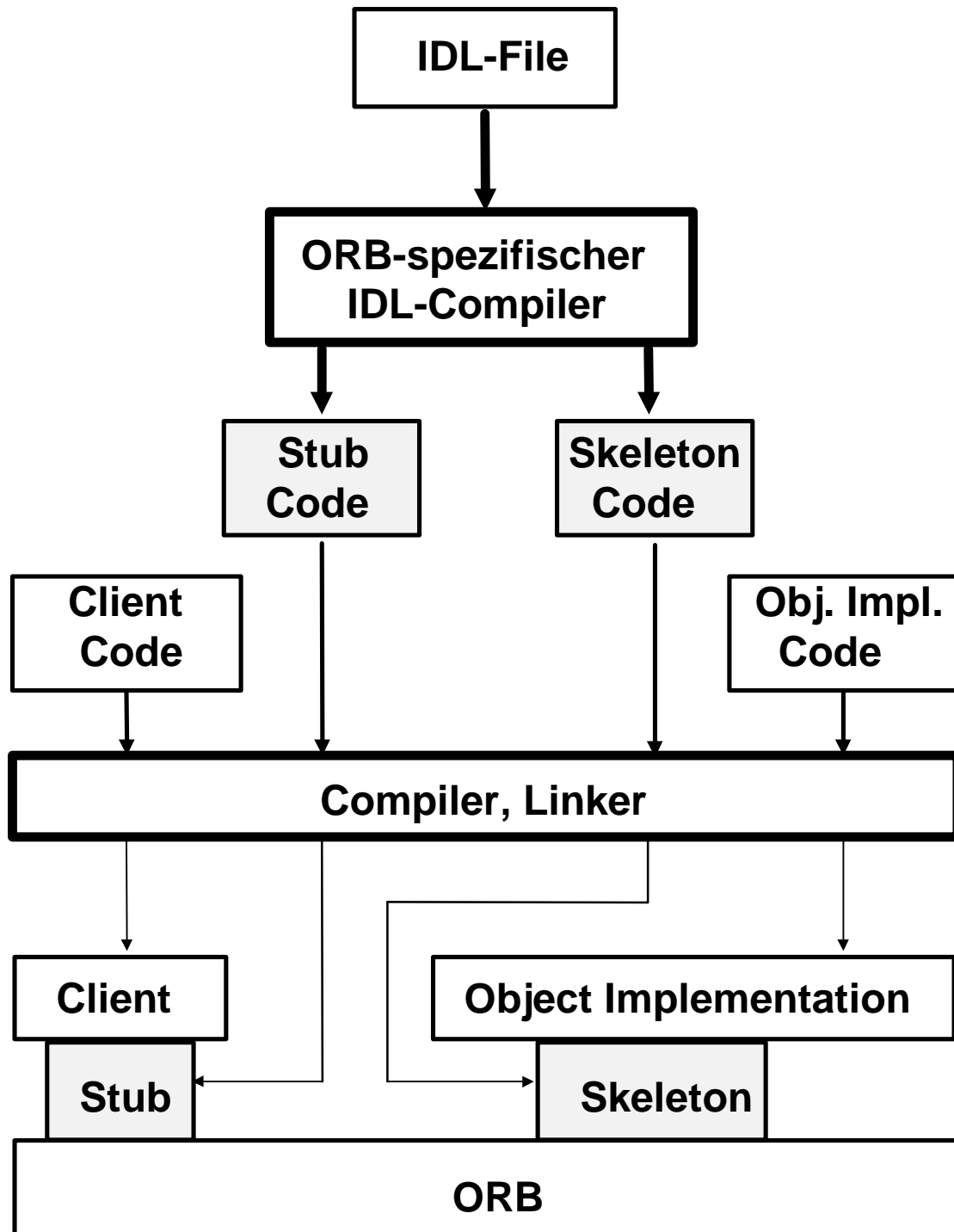


Arbeitsschritte zur Erstellung einer „Hello World“-Anwendung

1. Erstellung der Schnittstellenbeschreibung für das „Hello World“-Objekt
2. Übersetzung der Schnittstellenbeschreibung
3. Implementierung des Servers
4. Registrieren des Servers
5. Implementierung des Klienten
6. Aktivieren aller Komponenten (Ausführen der Anwendung)

OMG IDL

Erstellung von Client und Objekt-Implementation
mittels IDL-Compiler



Wiederverwendbarkeit von Code (2)

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

- **keine Sprachunabhängigkeit: Smalltalk- und C++-Objekte verstehen sich nicht**
- **keine 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. Der Grund: C++-Compiler benutzen Konstanten beim Zugriff auf Daten und Methoden**
- **Beschränkung auf einen Prozeßraum (Objekte können nicht über Prozeßgrenzen hinweg kommunizieren)**