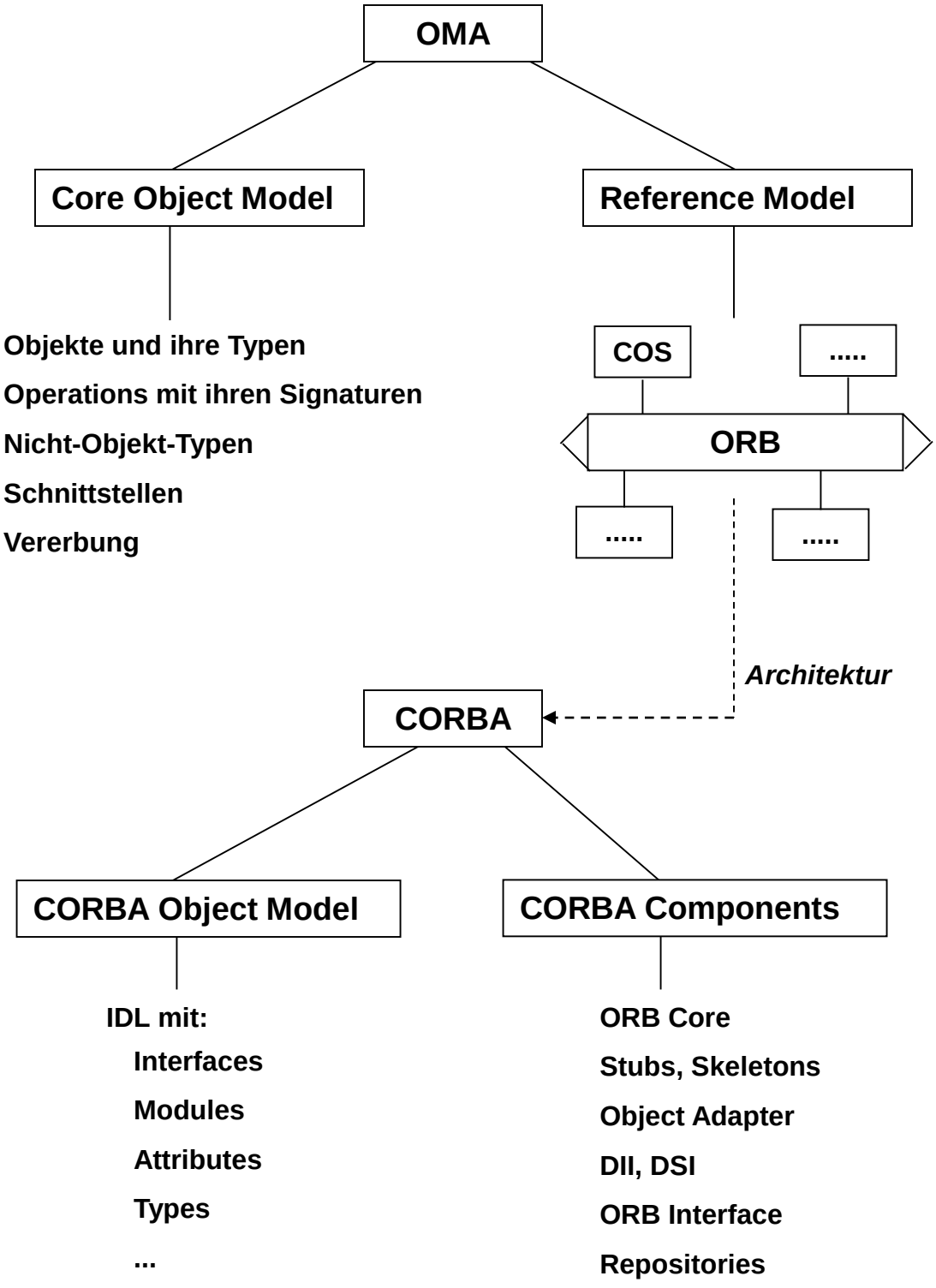
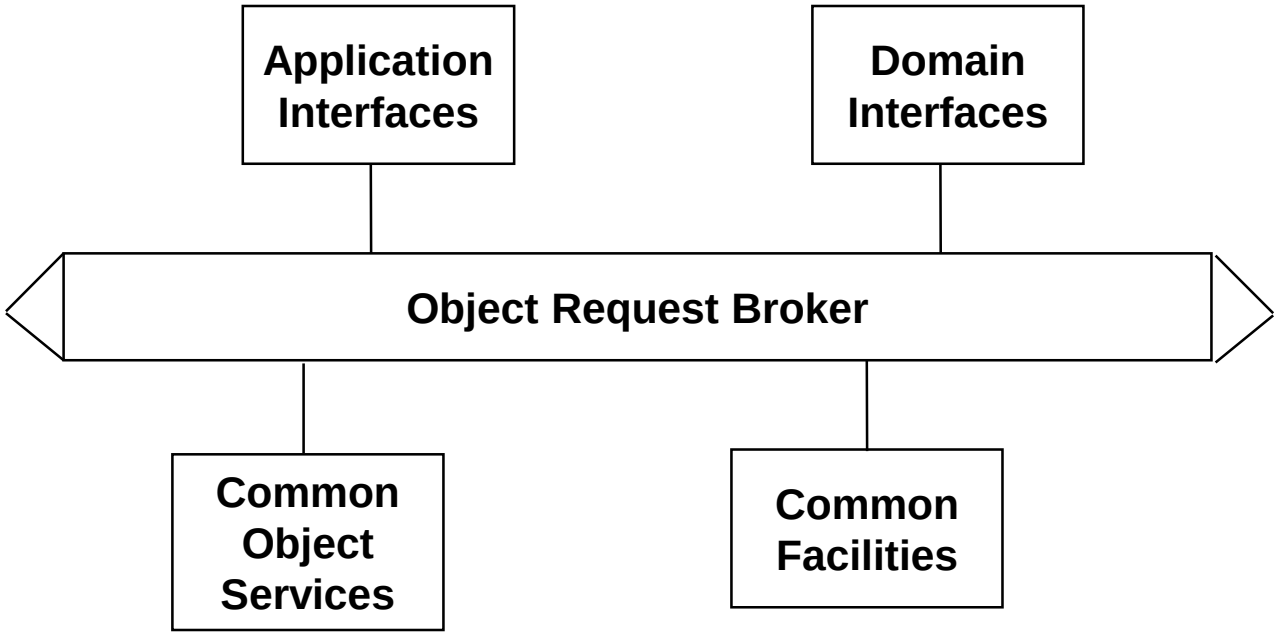


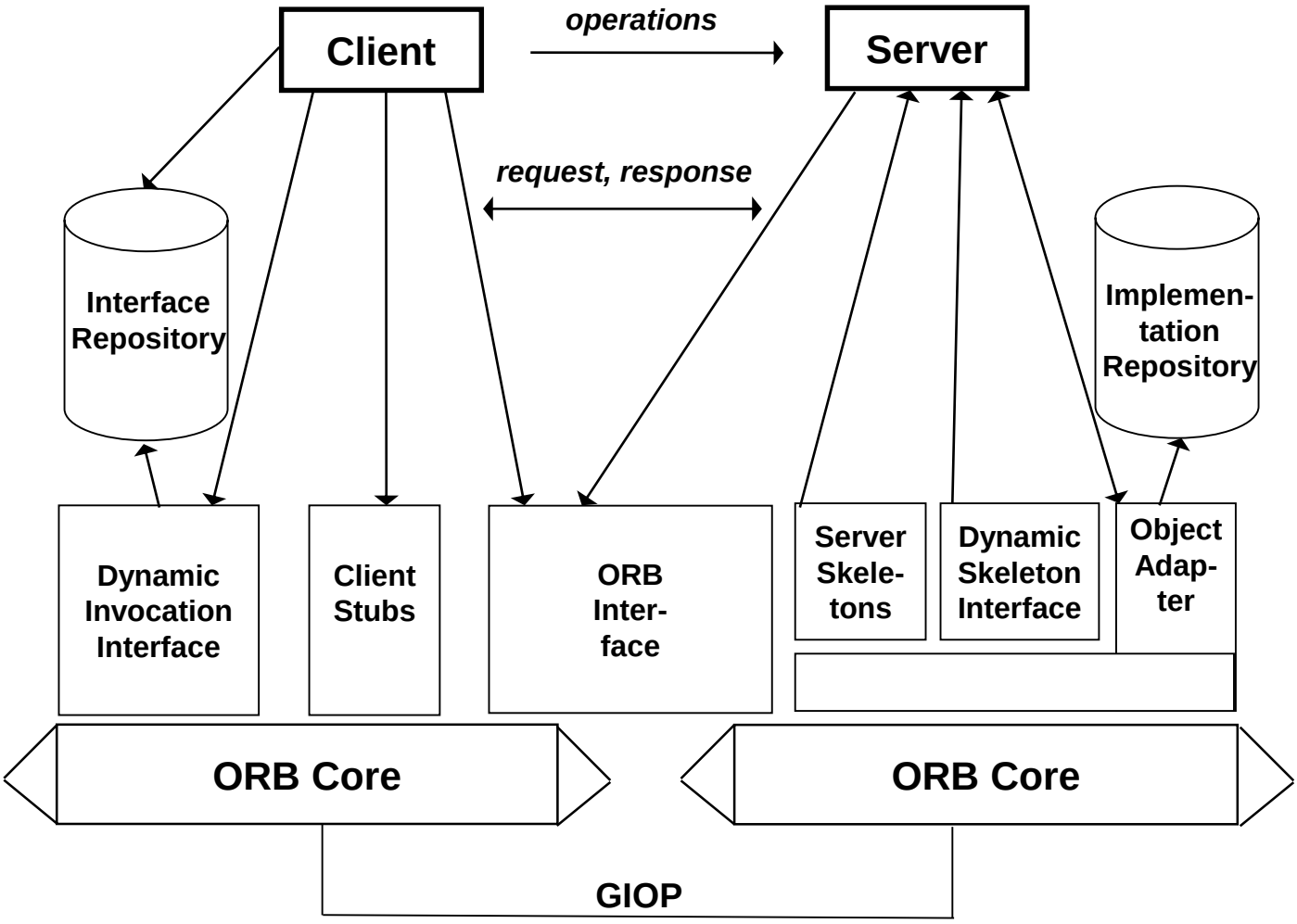
OMA = Object Management Architecture, ORB = Object Request Broker,
COS = Common Object Services, DII = Dynamic Invocation Interface,
DSI = Dynamic Skeleton Interface, IDL = Interface Description Language

Object Management Architecture OMA

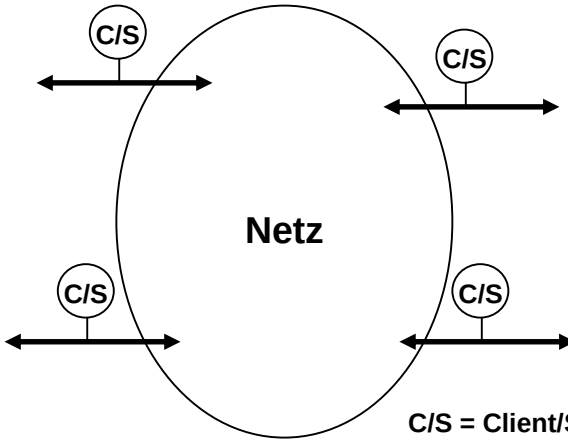


Common Object Services	Anwendungsunabhängiger Systemdienst. Beispiele: Namensdienst, Tradingdienst, Transaktions-dienst, Ereignisdienst.
Common Facility	Endanwenderdienst für verschiedene Anwendungen, kann auf Common Object Services aufbauen. Beispiel: Druckdienst.
Domain Interface	Dienst für speziellen Anwendungsbereich. Beispiele: Medizin-Dienst CORBAmed, Finanzdienst CORBAfinancials.
Application Interface	Nichtstandardisierter verteilter Anwendungsdienst.

OMA Referenz-Modell



GIOP = General Inter ORB Protocol



C/S = Client/Server

CORBA-Komponenten

ORB	**Software zur Unterstützung transparenter Interaktionen von Client-Objekten mit CORBA-Objekten, unabhängig von Hardware-Architekturen, Betriebssystemen, Programmiersprachen. ORBs kommunizieren über IIOP (Internet Inter-ORB Protocol).**
ORB Interface	**Bereitstellung von Grundfunktionen des ORB für Client und Server. Beispiele für Grundfunktionen: init(), object_to_string(), string_to_object(), connect(), run().**
Client Stub	**Bereitstellung eines Proxys mit derselben Schnittstelle wie ein CORBA-Objekt auf Client-Seite. Marshalling / Unmarshalling. Unterstützung einer statischen Schnittstelle. Bindung zur Übersetzungszeit.**
Server Skeleton	**Bereitstellung der Schnittstelle eines CORBA-Objektes für den entfernten Aufruf. Marshalling / Unmarshalling. Unterstützung einer statischen Schnittstelle. Bindung zur Übersetzungszeit.**
Object Adapter	**Erzeugung, Zerstörung, Lokalisierung, Aktivierung, Deaktivierung, Starten von CORBA-Objekten. Steuerung von Methodenaufrufen: Bestimmung des Zielobjektes und des Skeletons. Synchronisierung von Objektaufrufen. Verpflichtend für CORBA-Implementierungen: Portable Object Adapter (POA).**
Dynamic Invocation Interface	**Erzeugung von Aufrufen, bei denen die aufgerufene Operation erst zur Laufzeit, noch nicht zur Übersetzungszeit, feststeht.**
Dynamic Skeleton Interface	**Bereitstellung einer Schnittstelle für Aufrufe eines CORBA-Objektes ohne Kenntnisse der Schnittstellendefinition.**
Interface Repository	**Bereitstellung von Informationen über IDL-Definitionen zur Laufzeit, u.A. Informationen über Schnittstellen und Operationen.**
Implementation Repository	**Bereitstellung von Implementierungs- und Ausführungsdetails von Aufrufen von CORBA-Objekten, z.B. über Adressräume, Nebenläufigkeit, Portnummern, verwendete Datenstrukturen, ...**

CORBA-Komponenten: Aufgaben

Types	**Attributes**
Constants	**Interfaces**
Exceptions	**Modules**
Operations	

Objektreferenzen

Basistypen:	**(unsigned) short**
	(unsigned) long
	float, double
	char, string
	boolean
	octet
	any
komplexe Typen:	**enum**
	struct
	union
	sequence
	array
	interface

(Auszug)

IDL-Grundkonstrukte und -Typen

```
short i;
boolean schalter;
string error_msg;

enum Color { red, yellow, green};

struct  small_struct {short i; char c; string s;};

union u switch (boolean) {
        case FALSE: long count;
        case TRUE: string message;
};

typedef float mylist [50];   //Array
typedef float mytable [50] [20];   //Array
typedef sequence <string> myseq1;   //Sequenz
typedef sequence <small_struct, 50> myseq2;
typedef sequence <sequence < long, 100> > myseq3;
```

***keine* Variablen:**

tauchen in IDL-Spezifikationen nur innerhalb anderer Konstrukte (z.B. Konstanten, Attribute, struct-Komponenten, Argumentlisten von Operationen) auf

Arrays sind Folgen von Elementen desselben Typs mit fester Länge, Sequenzen sind Folgen von Elementen desselben Typs mit variabler Länge (maximale Länge spezifizierbar).

***Für die Definition von Array- und Sequenz-Typen muss typedef benutzt werden*.**

IDL-Typen: Beispiele

```
const string STR = "hello":

typedef short TempType;
const TempType MIN_TEMP = 5;

const float DOPPELPI = 3.14 * 2;
```

```
exception ex1 {short code; string info;};
```

```
short op1 (in string a1, out long a2, inout short a3);
TempType op2 () raises (ex1);
oneway void op3 (in short a1);
```

attribute short zahl;	⇔	short zahl(); void zahl (in short z);
readonly attribute string meldung;	⇔	string meldung();

IDL-Grundkonstrukte: Konstanten, Exceptions, Operations, Attributes

```
interface Konto {
        const short NLENGTH= 80;
        const short REASON = 240;

        enum Kto_Art {Giro, Spar, Festgeld};

        typedef char Name[NLENGTH];
        typedef unsigned long Tan;
        typedef sequence<Tan> Used_Tans;

        readonly attribute Kto_Art kto_art;
        readonly attribute unsigned long kto_nr;
        attribute Name kto_inhaber;
        attribute float kto_stand;
        attribute Used_Tans used_tans;

        exception ungueltigeAbhebung {
                long error_code;
                char reason [REASON];
        };

        oneway void Einzahlung (in float betrag);

        void Auszahlung (in float betrag, in Tan t)
                raises (ungueltigeAbhebung);

        float Kontostand_Feststellung ();
};
```

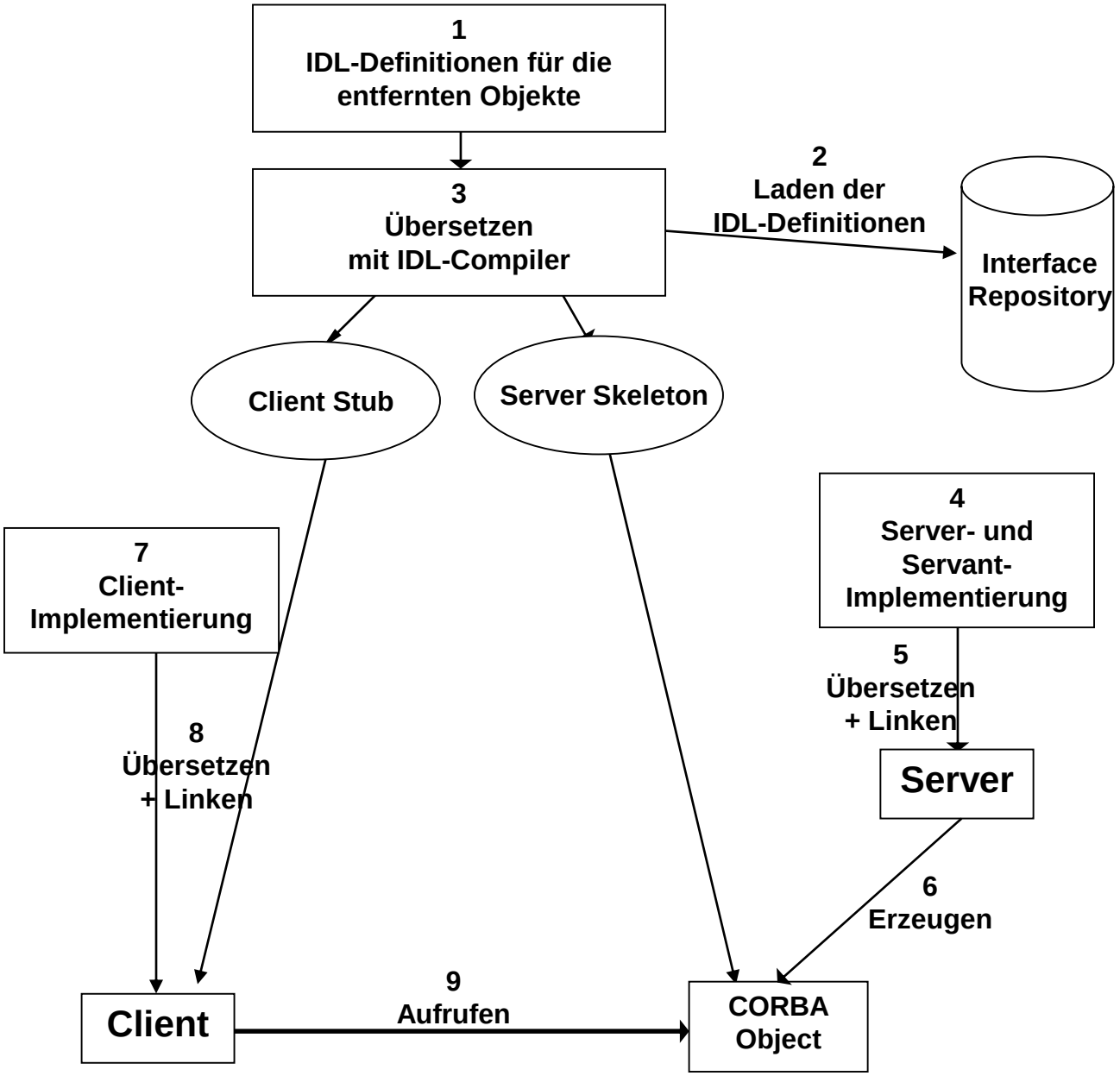
IDL-Interface: Beispiel

```
module mod1 {
        typedef ...
        enum ...
        interface if1 { ...};
        interface if2 { ...};
        interface if3 { ...};
};


module Finanzen {
        interface Konto{
            readonly attribute string inhaber;
            void einzahlen (in float betrag);
            void abheben (in float betrag);
            float kontostand_zeigen();
        };
};
```

IDL-Typ oder IDL-Konstrukt	C++-Typ	Java-Typ
(unsigned) short	CORBA::(U)Short	short
(unsigned) long	CORBA::(U)Long	int
float	CORBA::Float	float
double	CORBA::Double	double
char	CORBA::Char	char
string	char * (*)	java.lang.String
boolean	CORBA::Boolean	boolean
octet	CORBA::Octet	byte
any	CORBA::Any	org.omg.CORBA.Any
enum	enum	class (!)
struct	struct oder class	class
union	class (!)	class
(array)	(array)	(array)
sequence	class mit operator[]	array
interface	class + Referenztypen (..._ptr, ..._var)	interface
module	namespace	package
exception	class ... : ... Exception	class ... extends Exception
typedef	typedef	*(class in Helper-Datei)*
const (IDL-Typ)	const (C++-Typ)	static final ... (Java-Typ)
operation	member function	method
attribute	getter, setter memb. func's	getter, setter methods

(*) Speicherplatz-Management für CORBA-Strings:

```
namespace CORBA {
    static char*   string_alloc (ULong len);      // Speicherplatzzuweisung
    static char*   string_dup (const char *);     // Kopieren von Strings
    static void    string_free (char *);          // Speicherplatzfreigabe
    ...
}
```

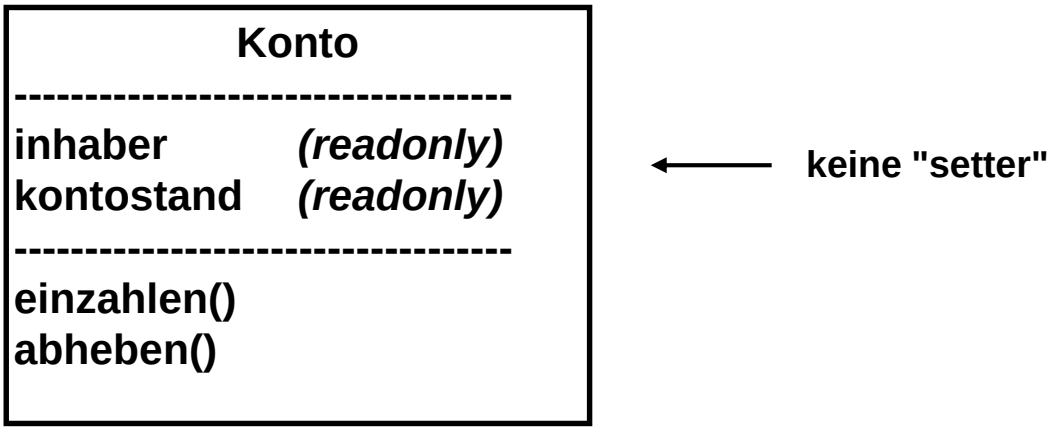


Server	Anwendung, die entfernt zu benutzende Objekte erzeugt und anschließend auf eintreffende Operationsaufrufe wartet
Servant	Objekt, das vom Server erzeugt wird, um vom Client angesprochen zu werden (programmiersprachliche Implementierung eines entfernten Objektes, d.h. z.B. Java- oder C++-Objekt)
CORBA-Objekt	Objekt, das aus einem Servant durch Zuweisung einer Objektreferenz und Registrierung an einem Objekt-Adapter hervorgeht

CORBA-Entwicklungsprozess (statischer Fall) I

1	**Entwicklung der Schnittstellen-Definitionen der Server-Objekte mit IDL.**
2	**Laden der Schnittstellen-Definitionen ins Interface Repository (auch *Binden* genannt, das Interface Repository ermöglicht den Zugriff auf IDL-Definitionen zur Laufzeit).**
3	**Übersetzen der Schnittstellen-Definitionen. Es werden erzeugt: Client Stubs, Server Skeletons, (meist) Beispielklassen für die Server-Implementierung. Es ist möglich, dass die Schnittstellendefinitionen für Client und Server in verschiedene Programmiersprachen übersetzt wird.**
4	**Implementierung des Servers und der Klassen für die Server-Objekte (z.B. werden die Beispielklassen weiterentwickelt).**
5	**Übersetzen und Linken der Server- und Server-Objekt-Implementierungen (normaler C++- oder Java-Compiler).**
6	**Starten des Servers und Erzeugen von Server-Objekten durch den Server.**
7	**Implementierung des Clients.**
8	**Übersetzen und Linken des Clients.**
9	**Starten des Clients und Aufrufen von Server-Objekten durch den Client.**

CORBA-Entwicklungsprozess (statischer Fall) II




```
// IDL-Definition
interface Konto {
        readonly attribute string inhaber;
        // oder: string inhaber_zeigen ();
        float kontostand_zeigen ();
        // oder: readonly attribute float kontostand;
        void einzahlen (in float betrag);
        void abheben (in float betrag);
};
```

```
#include <iostream>
#include "konto.h"

class Konto_impl :  public virtual POA_Konto {

      private:  CORBA::Float   _kontostand;
                char * _inhaber;

      public:
            Konto_impl() {
                  _kontostand = 0;
                  _inhaber = (const char*) "Mustermann";
            };

            Konto_impl (int st, const char* inh) {
                  _kontostand = st;
                  _inhaber = CORBA::string_dup(inh);
            };

            void einzahlen (CORBA::Float betrag) {
                  _kontostand = _kontostand + betrag;
            };

            void abheben (CORBA::Float amount) {
                  _kontostand = _kontostand - betrag;
            };

            CORBA::Float kontostand_zeigen () {
                  return _kontostand;
            };

            char *inhaber_zeigen () {
                  return CORBA::string_dup (_inhaber);
            };

};
```

CORBA-Servant (Beispiel)

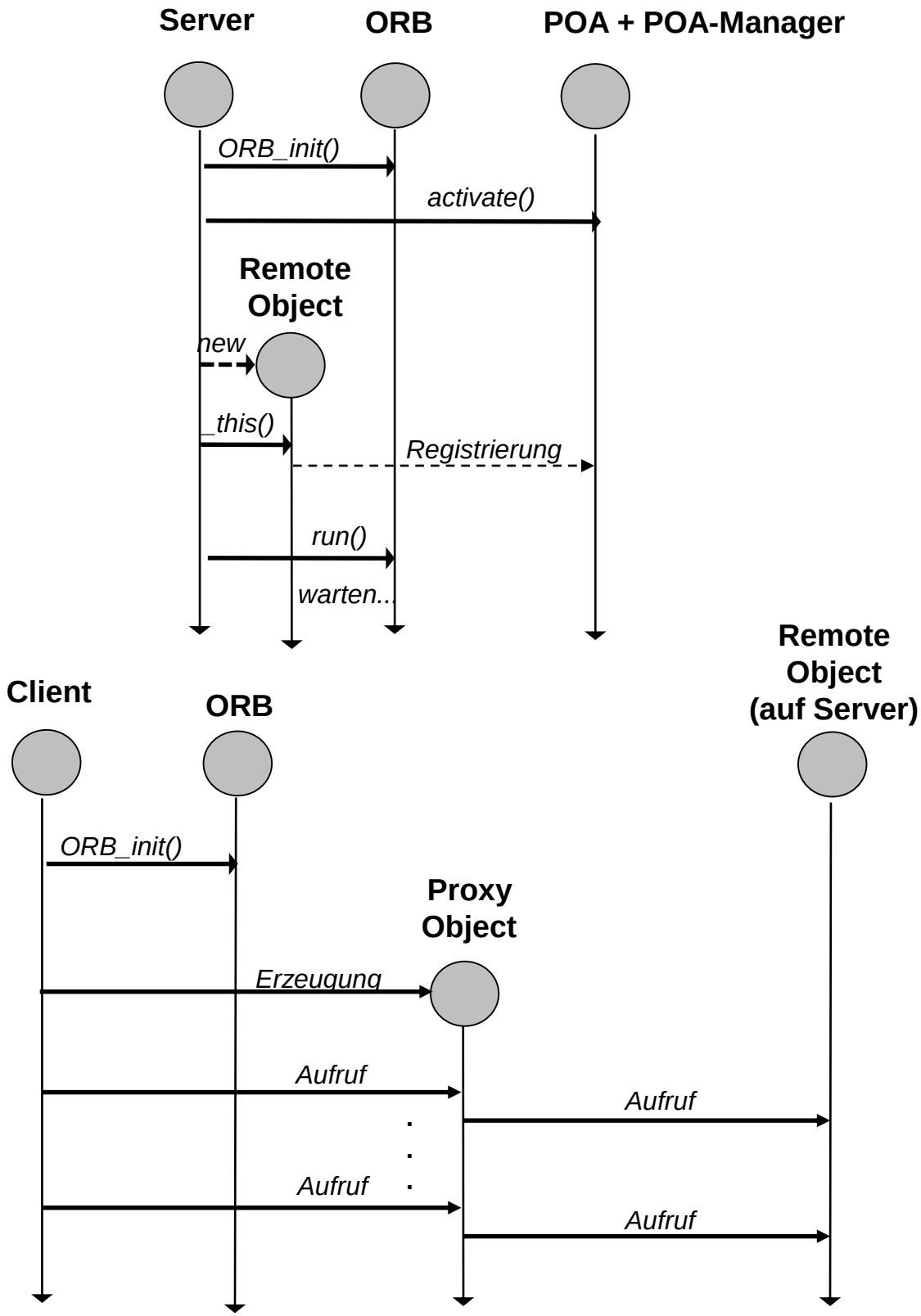
```
// Usage: <servername>
int main (int argc, char *argv[]) {
    try {
        CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);
        CORBA::Object_var obj = orb->resolve_initial_references ("RootPOA");
        PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);
        PortableServer::POAManager_var mgr = poa->the_POAManager();
        mgr->activate();

        Konto_impl kto_servant (0.0, "Musterperson");          // Servant
        Konto_var kto = kto_servant._this ();                  // CORBA-Objekt
        CORBA::String_var str = orb->object_to_string (kto);
        cout << str << endl;

        orb->run();                                            // Server wartet

    } catch (const CORBA::Exception& e) {
        cout << "CORBA exception: " << e << endl;
        return 1;
    }
}
```

CORBA-Server (Beispiel)

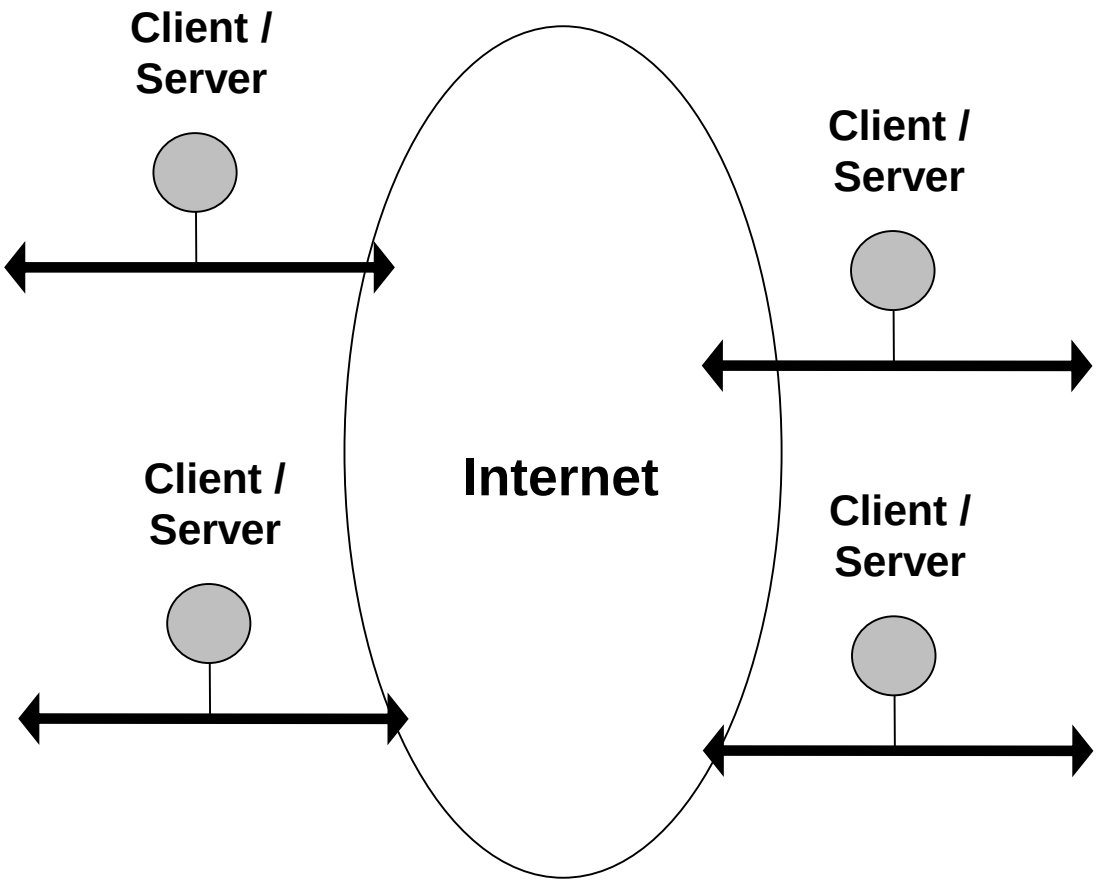
```
// Usage: <clientname> <IOR>
#include <iostream>
#include "konto.h"

int main (int argc, char *argv[] ) {
      CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

      CORBA:: Object_var obj = orb->string_to_object (argv[1]);
      Konto_var client = Konto::_narrow (obj);

      client->einzahlen (700.00);
      client->abheben (250.00);
      cout << "Kontostand ist " <<
                              client->kontostand_zeigen() << endl;
      cout << "Inhaber ist " client->inhaber_zeigen() << endl;

      return 0;
}
```

IOR = Interoperable Object Reference

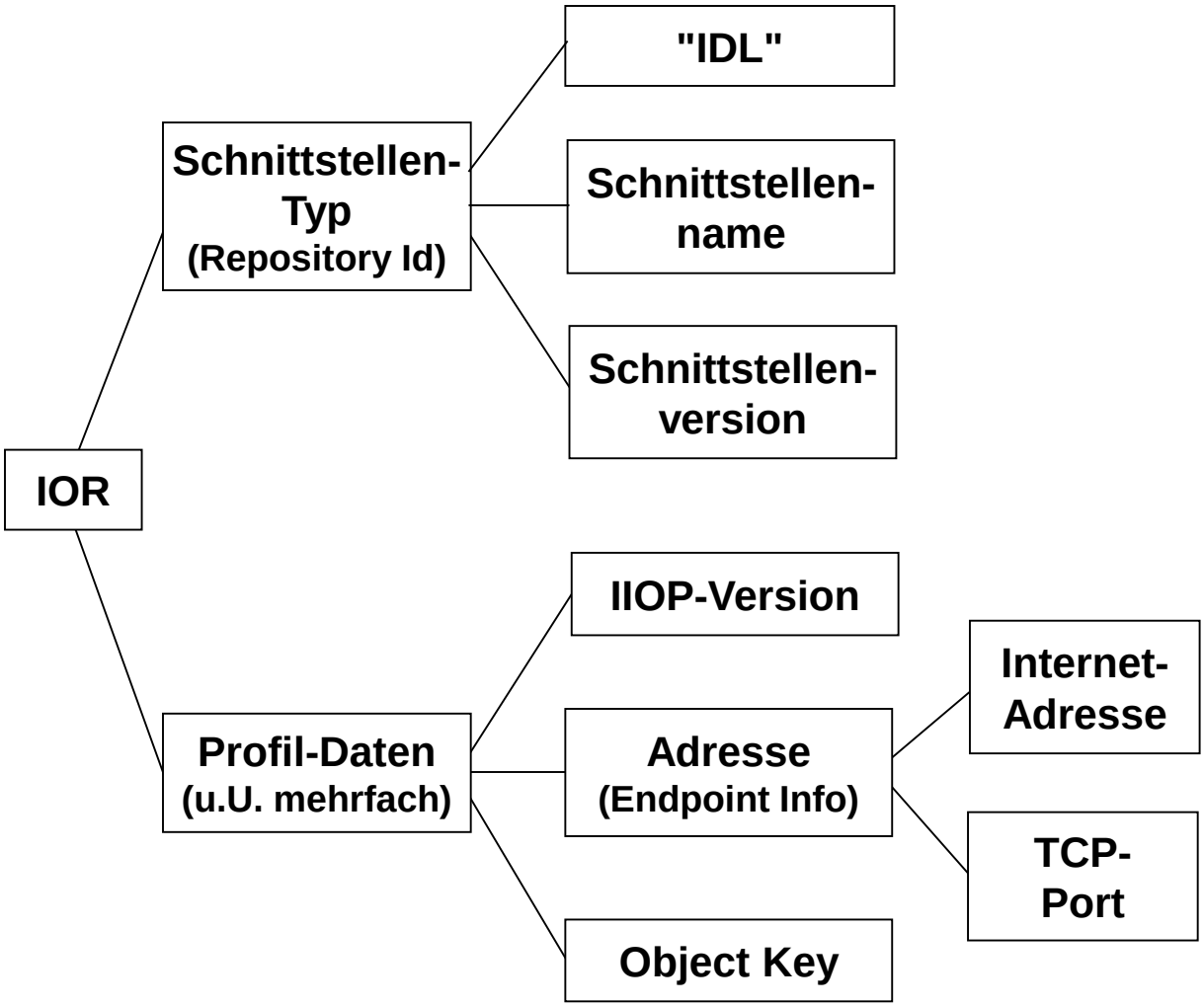
CORBA-Client (Beispiel)



CORBA: Server- und Client-Verhalten



Inter-ORB-Kommunikaton



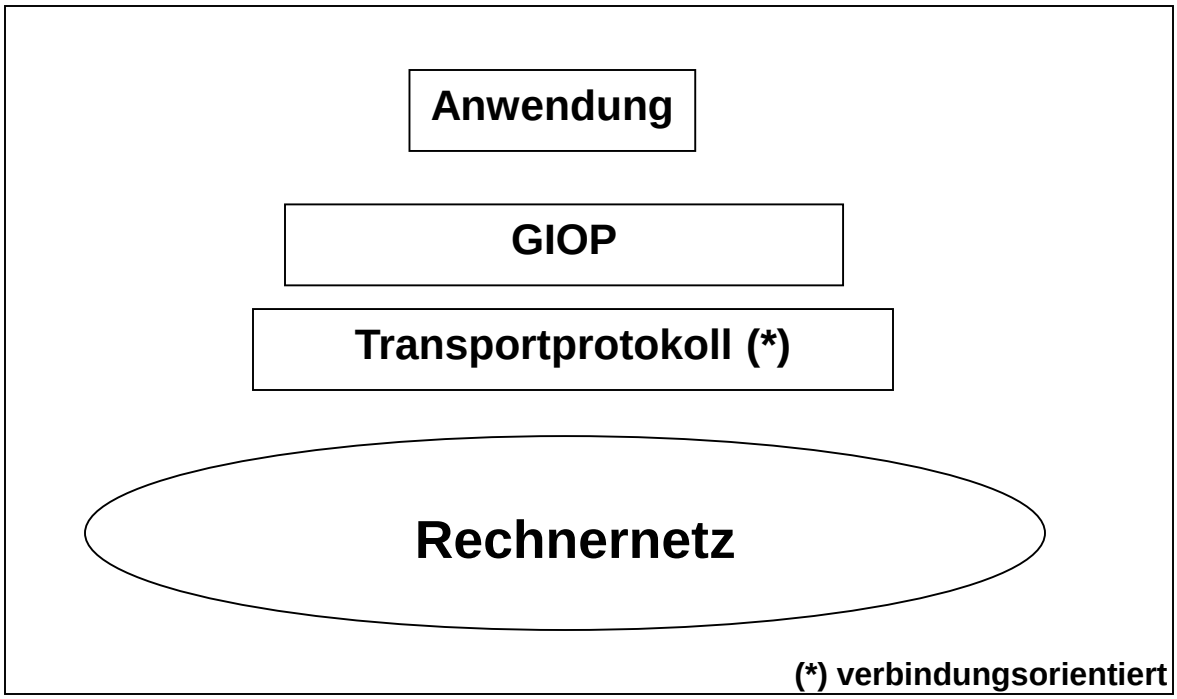
IOR = Interoperable Object Reference

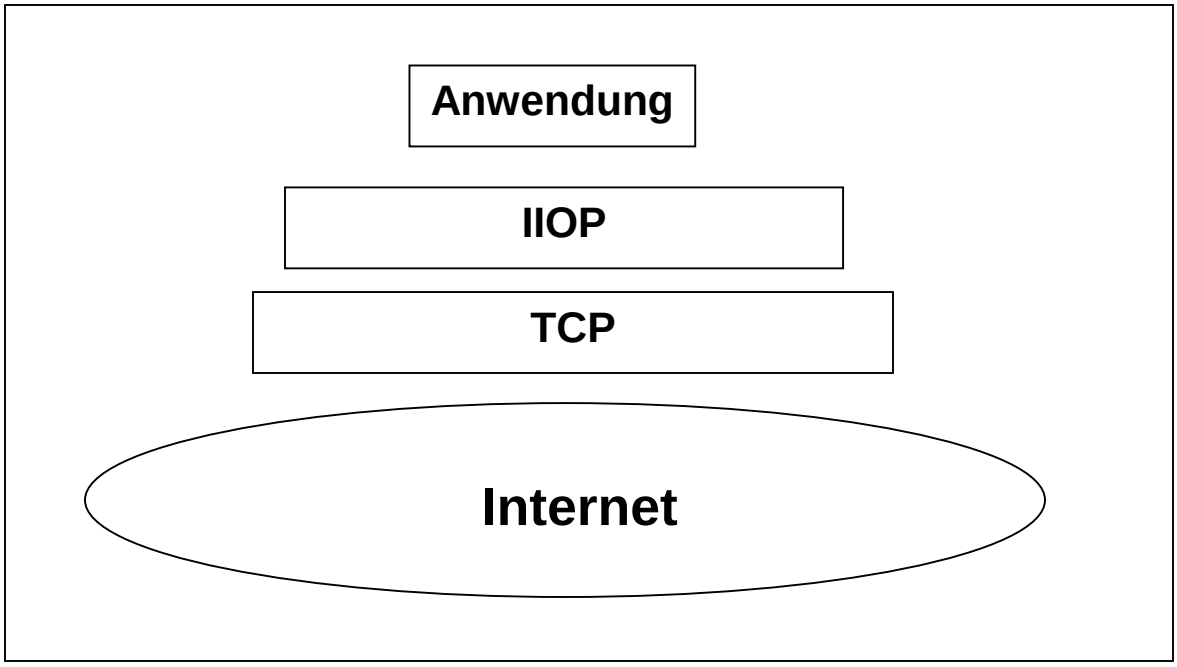
IOR: Aufbau

IOR:010000000E00000049444C3A496F6E746F3A312E30000000
0100000000000002E0000000101000000D0000003139332E
36382E32352E3633000069121200000006A33F33C4E32373630
AA9B0497E8FA1245DC

hex-Code	Beschreibung
01	Little-Endian-Codierung: niedrigststellige Ziffer bei Zahlen am Anfang (linksbündig)
00 00 00	Padding
0E 00 00 00	Länge des folgenden Schnittstellen-Typs = 14 Oktetts.
49 44 4C 3A 49 6F 6E 74 6F 3A 31 2E 30 00	"IDL:Konto:1.0\0"
00 00	Padding
01 00 00 00	Es folgt genau ein Satz von Profil-Daten.
00 00 00 00	ProfileId mit Wert TAG_INTERNET_IOP = 0, d.h. das durch die folgenden Profil-Daten charakterisierte Server-Objekt ist über das Internet-Protokoll TCP erreichbar.
2E 00 00 00	Länge der folgenden Profil-Daten = 46 Oktetts.
01	Little-Endian-Codierung
01 00	IIOP Version 1.0
00	Padding
0D 00 00 00	Länge der folgenden Internet-Adresse = 13 Oktetts.
31 39 33 2E 36 38 2E 32 35 2E 36 33 00	Internet-Adresse = "193.68.25.63\0"
00	Padding
69 12	TCP-Port = 4713 (12_{16}*16^2+69_{16} = 4608 + 105)
12 00 00 00	Länge des folgenden Object Key = 18 Oktetts.
6A 33 F3 3C 4E 32 37 36 30 AA 9B 04 97 E8 FA 12 45 DC	Object Key

IOR: Beispiel





CORBA-Protokollarchitektur

GIOP Message (allgemein):

12-Byte GIOP Message Header	Variable-length GIOP Message Body

(nachrichtenspez. Header + Message Body)

Header: Bytes 0-3: "GIOP"
Bytes 4-5: GIOP-Version, z.B. 1.2
Byte 6: Flags bzgl. Encoding (big/little endian), Fragmentation
Byte 7: Message-Typ, z.B. 0 für Request
Bytes 8-11: Message-Länge ohne Headers

Message-Typ	Richtung	Erläuterung
Request	C→S	Anfrage an ein entferntes Objekt
Reply	S→C	Antwort eines entfernten Objektes
CancelRequest	C→S	Annullieren eines laufenden Requests
LocateRequest	C→S	Einholen von Adressierungs-Info über ein entferntes Objekt
LocateReply	S→C	Antwort auf ***LocateRequest***
CloseConnection	C↔S	Abbruchsignalisierung
MessageError	C↔S	Signalisierung, dass eine empfangene Nachricht fehlerhaft war
Fragment	C↔S	Fragmentierte Nachricht

C=Client, S=Server

GIOP Request Message (eingebettet in eine allgemeine GIOP-Message):

12-byte GIOP Message Header	Variable-length GIOP Request Header	Variable-length GIOP Request Body

mit Object Key, Operation Name
mit *in*- und *inout*-Argumenten der Operation

GIOP Messages

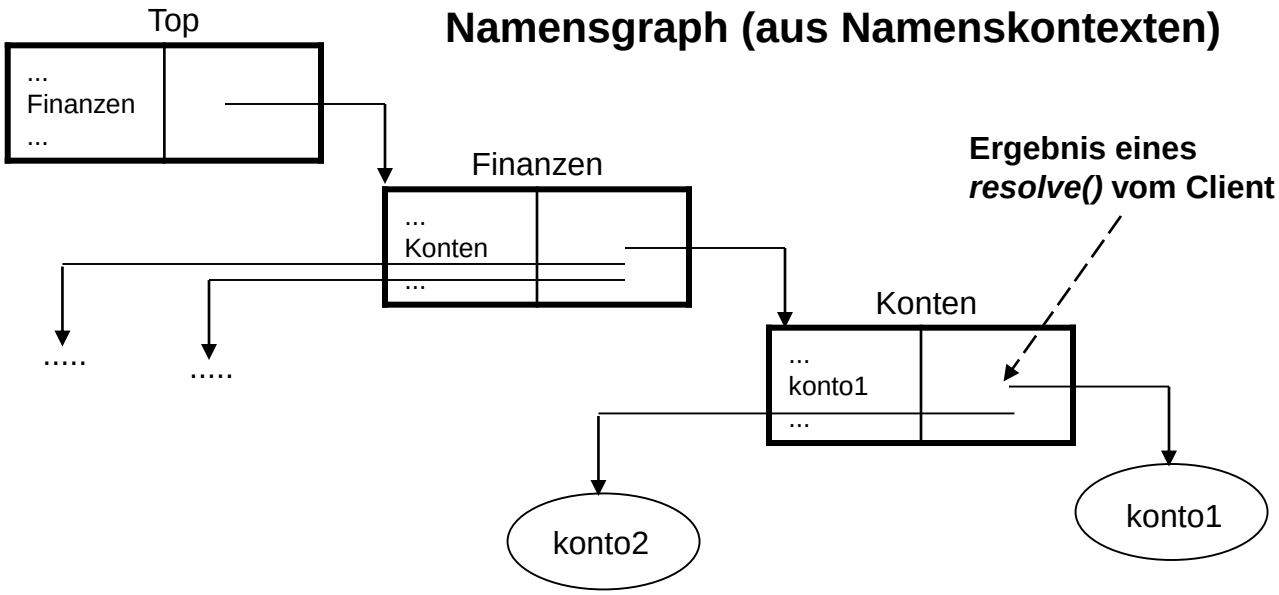
CORBA Service	Aufgabe
Naming	Verwaltung von Namen, Zuordnung von Namen und Objektreferenzen
Life Cycle	Verwaltung von Objekten: Erzeugen, Kopieren, Verschieben, Löschen
Event	spontane Benachrichtigung über Ereignisse
Trading	Angebot und Inanspruchnahme von Diensten mit vorgegebenen Eigenschaften, Zuordnung von Eigenschaften und Objektreferenzen
Transaction	Unterstützung von Transaktionen über Two-Phase Commit
Concurrency Control	Unterstützung der Nebenläufigkeit von Transaktionen durch einen Lock Manager, der Sperren implementiert
Security	Unterstützung von Zugriffskontrolllisten, Vertraulichkeit, Non-Repudiation
Persistence	Dauerhaftes Speichern von Objekten
Externalization	Unterstützung von Streams für Ein/Ausgabe von Daten in/aus Objekten
Query	Bereitstellung von Abfrageoperationen an Objekte auf Basis von SQL3 und OQL
Collections	Unterstützung von Collections (Sets, Bags, Lists, Arrays, Dictionaries) von Objekten
Relationship	Verwaltung von Beziehungen zwischen Objekten
Time	Uhrzeitsynchronisation zwischen verteilten Objekten, Unterstützung zeitgetriggerter Ereignisse
Licensing	Nutzungskontrolle von Ereignissen
Properties	Verwaltung (insbesondere Hinzufügen, Löschen) von Eigenschaften zu/von Objekten

Namenskontext

Namensbindung →

Namenskomponente	**Objektreferenz**
.	.
.	.

Namensgraph (aus Namenskontexten)



Name (aus Namenskomponenten: Top/Finanzen/Konten/konto1

Top	Finanzen	Konten	konto1
Namens-komponente: ***Kontextname***	*Namens-komponente:* ***Kontextname***	*Namens-komponente:* ***Kontextname***	*Namens-komponente:* ***elementarer Name***

Identifier	**Kind**

(häufig leerer String,
Kind = deutsch.: Art)

Naming Service: Grundbegriffe

```
module CosNaming {                    // Cos = Common Object Services
      typedef string Istring;

      struct NameComponent {
            Istring id;
            Istring kind;
      };

      typedef sequence <NameComponent> Name;

      interface NamingContext {
            ...
            void bind (in Name n, in Object obj) raises ....;
            void rebind (in Name n, in Object obj) raises ....;
            Object resolve (in Name n) raises ....;
            NamingContext bind_new_context(in Name n) ...;
            ...
      };

      // interface BindingIterator mit Operationen next_one() ...
      ...
};
```

NamingContext
resolve(), list(), destroy(), new_context(), unbind(), bind(), rebind(), bind_context(), rebind_context(), bind_new_context()

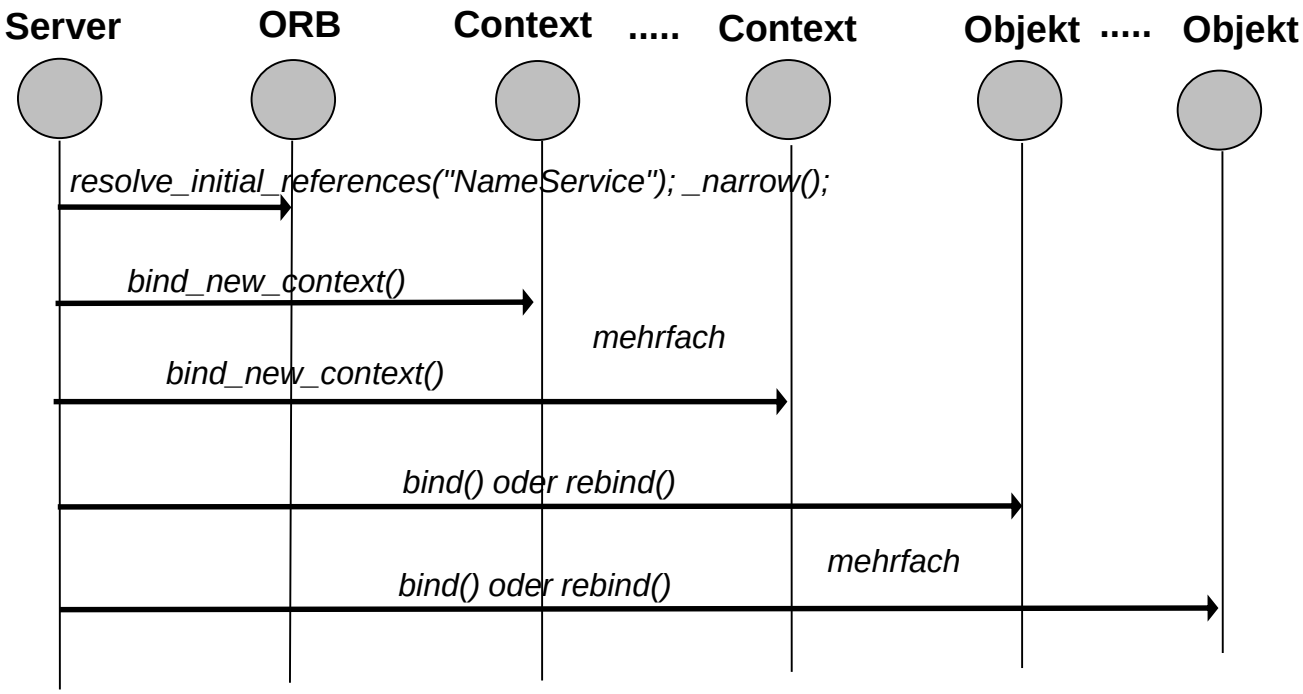
BindingIterator
next_one(), next_n(), destroy()

Objektreferenz für den Naming Service:

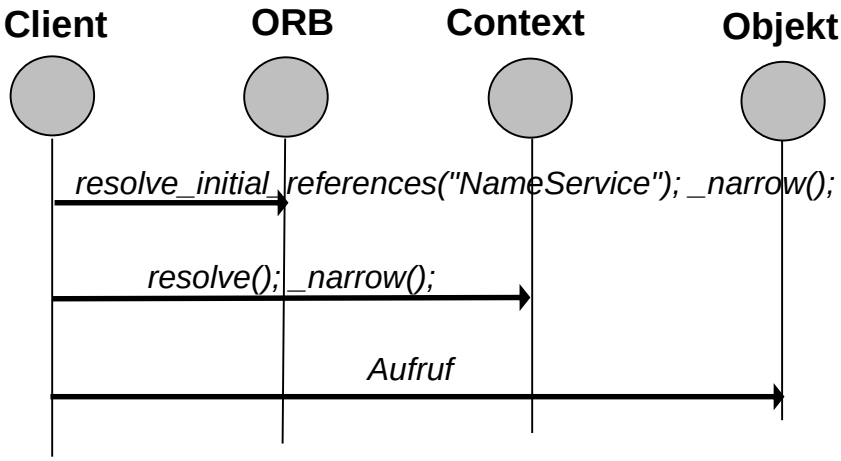
```
... = .... resolve_initial_references("NameService");
```

Naming Service: IDL und Operationen

Server :



Client:

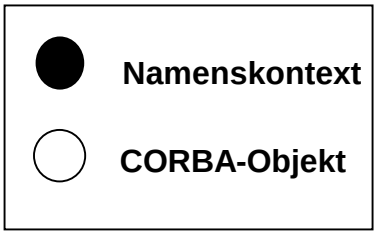


Naming Service: Benutzung

```
module KontoApp {
        interface Konto {
                void einzahlen (in float betrag);
                void abheben (in float betrag);
                float kontostand_zeigen ();
        };
};


class KontoServant extends _KontoImplBase {

        private float kontostand;

        KontoServant () {
             super();
             System.out.println ("KontoServant created.");
             kontostand = 0.0;
        }

        public void einzahlen (float betrag) {
             kontostand = kontostand + betrag;
        }

        public void abheben (float betrag) {
             kontostand = kontostand - betrag;
        }

        public float kontostand_zeigen() {
             return kontostand;
        }

}
```

Naming Service Beispiel 1 + 2: IDL und Java-Servant

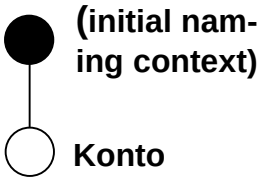
```
public class KontoServer {

    public static void main (String args[] ) {

        try {
            ORB orb = ORB.init (args, null);
            KontoServant ktoRef = new KontoServant ();                  // 1
            orb.connect (ktoRef);                                       // 2
            org.omg.CORBA.Object objRef
                = orb.resolve_initial_references ("NameService");       // 3
            NamingContext ncRef
                = NamingContextHelper.narrow (objRef);                  // 4
            NameComponent nc
                = new NameComponent ("Konto", "");                      // 5
            NameComponent nm[] = {nc};                                  // 6
            ncRef.rebind (nm, ktoRef);                                  // 7
            java.lang.Object sync = new java.lang.Object ();            // 8
            synchronized (sync) {
                sync.wait();                                            // 9
            }
        } catch (Exception e) {
            System.out.println ("Error: " + e);
            e.printStackTrace (System.out);
        }
    }
}
```



1: Objektreferenz für Anwendungsobjekt erzeugen
2. Referenz mit ORB verbinden, jetzt können entfernte Aufrufe durch das Objekt empfangen werden
3: Objektreferenz für Naming Service holen
4: Auf Typ NamingContext downcasten
5: Namenskomponente erzeugen
6: Namenskomponente in Namen einfügen (Name = Array von Namenskomponenten)
7: Namen an Referenz binden (Namensbindung erzeugen)
8: Monitor-Objekt für Server-Warteschleife erzeugen
9: Server warten lassen

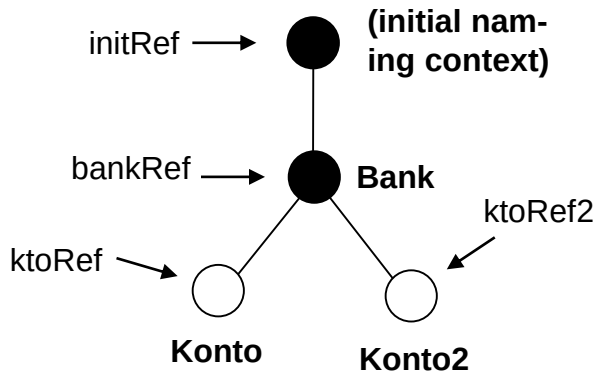
Naming Service Beispiel 1: Java-Server

```
import ....;

public class KontoClient {

    public static void main (String args[] ) {

        try {
            ORB orb = ORB.init (args, null);
            org.omg.CORBA.Object objRef                          //1
                = orb.resolve_initial_references ("NameService");
            NamingContext ncRef                                  //2
                = NamingContextHelper.narrow (objRef);
            NameComponent comp1                                  //3
                = new NameComponent ("Konto", "");
            NameComponent name[] = {comp1};                      //4
            org.omg.CORBA.Object obj = ncRef.resolve (name);     //5
            Konto ktoRef = KontoHelper.narrow (obj);             //6
            ktoRef.einzahlen (600.00);                           //7
            ktoRef.abheben (222.00);                             //7
            System.out.println ("Kontostand: " +
                                ktoRef.kontostand_zeigen() );    //7
        } catch (Exception e) {
            System.out.println ("Error: " + e);
            e.printStackTrace (System.out);
        }
    }
}
```

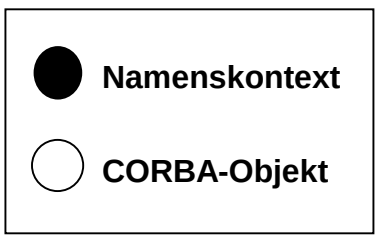


1: Objektreferenz für Naming Service holen
2: Auf Typ NamingContext downcasten
3: Namenskomponente erzeugen
4: Namenskomponente in Namen einfügen
5: Objektreferenz zum Namen finden
6: Objektreferenz auf Anwendungstyp downcasten
7: Objektreferenz verwenden

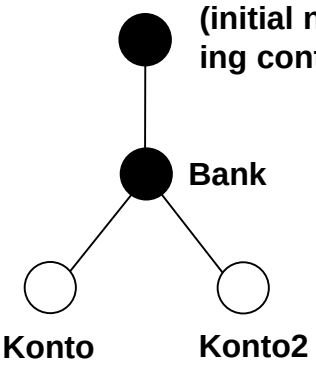
Naming Service Beispiel 1: Java-Client




```
import ...;

public class KontoServer {

    public static void main (String args[] ) {

        try {

            ORB orb = ORB.init (args, null);
            org.omg.CORBA.Object objRef
                = orb.resolve_initial_references ("NameService");
            NamingContext initRef = NamingContextHelper.narrow (objRef);
            NameComponent comp1
                    = new NameComponent ("Bank", "");
            NameComponent name[] = {comp1};
            NamingContext bankRef = initRef.bind_new_context (name);

            KontoServant ktoRef = new KontoServant ();
            orb.connect (ktoRef);

            KontoServant ktoRef2 = new KontoServant ();
            orb.connect (ktoRef2);

            name[0]= new NameComponent ("Konto", "");
            bankRef.rebind (name, ktoRef);

            name[0]= new NameComponent ("Konto2", "");
            bankRef.rebind (name, ktoRef2);

            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) {
                sync.wait();
            }

        } catch (Exception e) {
            System.out.println ("Error: " + e);
            e.printStackTrace (System.out);
        }
    }
}
```



Naming Service Beispiel 2: Java-Server




```
import ....;

public class KontoClient {

    public static void main (String args[] ) {

        try {
            ORB orb = ORB.init (args, null);
            org.omg.CORBA.Object objRef
                = orb.resolve_initial_references ("NameService");
            NamingContext rootRef = NamingContextHelper.narrow (objRef);


            NameComponent comp1
                = new NameComponent ("Bank", "");
            NameComponent comp2
                = new NameComponent ("Konto", "");
            NameComponent name[] = {comp1, comp2};
            org.omg.CORBA.Object obj = rootRef.resolve (name);
            Konto ktoRef = KontoHelper.narrow (obj);
            ktoRef.einzahlen (600.00);
            ktoRef.abheben(222.00);
            System.out.println ("Kontostand von Konto: " +
                                        ktoRef.kontostand_zeigen() );


            comp1 = new NameComponent ("Bank", "");
            comp2 = new NameComponent ("Konto2", "");
            NameComponent name1[] = {comp1, comp2};
            obj = rootRef.resolve (name1);
            ktoRef = KontoHelper.narrow (obj);
            ktoRef.einzahlen (700.00);
            ktoRef.abheben (333.00);
            System.out.println ("Kontostand von Konto2: " +
                                        (ktoRef.kontostand_zeigen() );

        } catch (Exception e) {
            System.out.println ("Error: " + e);
            e.printStackTrace (System.out);
        }
    }
}
```

Naming Service Beispiel 2: Java-Client