Kapitel 8

Verteilte Objektarchitekturen

Inhalt

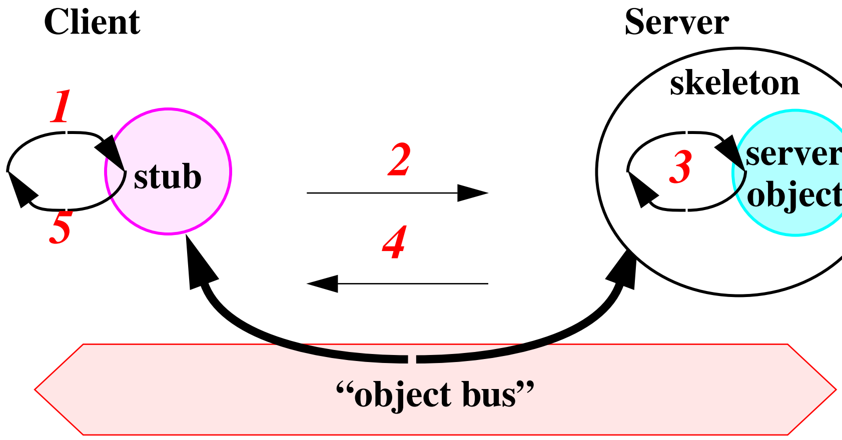
- ❑ Grundlagen
- ❑ **CORBA**
- ❑ DCOM/COM+
- ❑ Java RMI
- ❑ Zusammenfassung

Grundlagen (1) - Motivation

- Grundidee: Weiterentwicklung des RPC-Konzepts für Objekte.
  - Anwendung besteht aus interoperablen Objektkomponenten, potentiell über Netzwerk verteilt
  - Verteilungsaspekte sind für die Anwendung nicht sichtbar
  - Objektdienste sind über Remote Method Invocation (RMI) nutzbar
- Vorteile der Objektorientierung nutzbar
  - Objektidentität
  - Verkapselung: Dienste/Zustandänderung nur über Methodenaufruf
  - Vererbung, Polymorphismus
  - Trennung von Schnittstelle und Implementierung
  - Wiederverwendbarkeit von Objekten
- Objekte können (für den Programmierer transparent) interagieren
  - über verschiedene Programmiersprachen hinweg
  - zwischen unterschiedlichen Betriebssystem- und Hardwareplattformen
  - über Rechnerknoten mit Hilfe verschiedener Netzwerkprotokolle

Grundlagen (2) - Remote Method Invocation

❑ Grundprinzip



- **Server Object**: Objektinstanz, die den Zustand eines "business objects" einkapselt, und sein Verhalten implementiert (durch *public interface* beschrieben)
- **Stub** und **Skeleton** realisieren Ortstransparenz. Stub implementiert das gleiche interface wie das server object, agiert and seiner Stelle
- Client-Objekt ruft Objektmethode des Stub-Objekts (***1***)
- Stub kodiert Aufruf(-parameter) als Botschaft ("marshalling") und schickt sie zum Server (***2***)
- Skeleton dekodiert die Botschaft ("unmarshalling") und ruft die enstprechende Serverobjektmethode (***3***)
- Das Resultat des Aufrufs wird vom Skeleton an den Client zurückgeschickt (***4***)
- Stub liefert das Resultat an Clientanwendung ab (***5***)

❑ Client und Server sind nur Rollen

CORBA (1) - Einführung

- CORBA:
  Common Object Request Broker Architecture

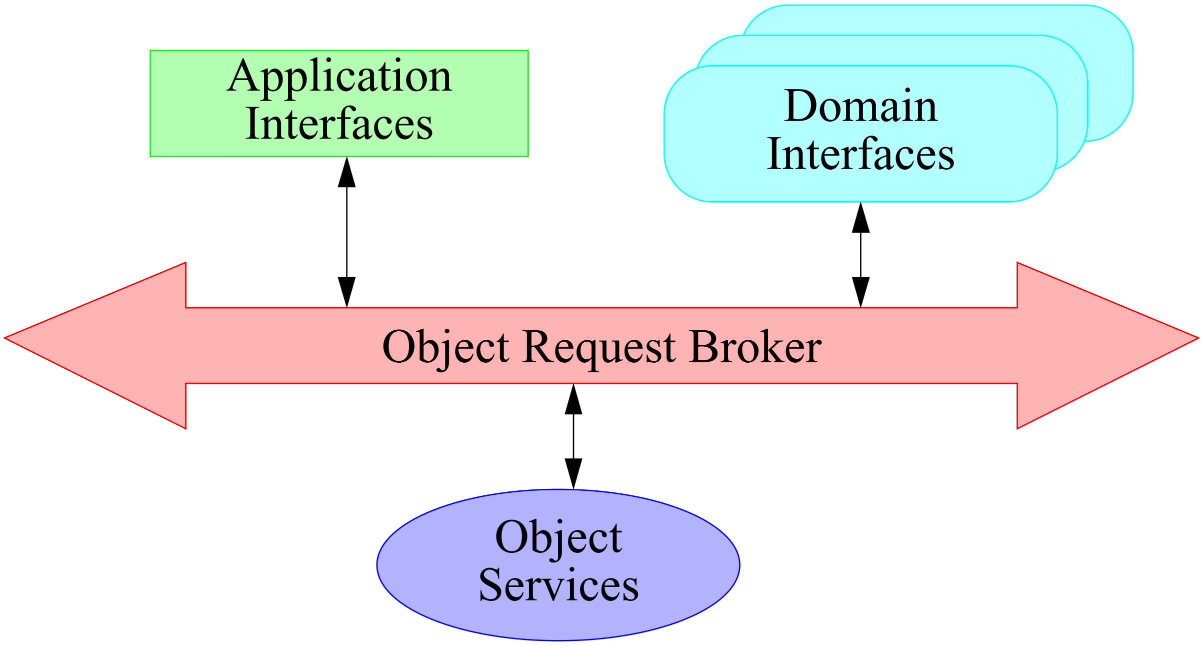
- objektorientierte, universelle Middleware-Plattform
  - Objektbus- und Komponenten-Architektur
  - erweitert RPC zu einem objektorientierten bzw. komponentenorientierten Programmiermodell
  - sprachunabhängig
  - plattformunabhängig

- OMG
  - Industriekonsortium (gegründet 1989, 11 Mitglieder)
  - heute über 1000 Mitglieder
  - keine Standards, keine Referenzimplementierungen, Herstellerunabhängigkeit
  - Produkt: Object Management Architecture (OMA)
    - Objektmodell definiert die Beschreibung von Objektschnittstellen
    - Referenzmodell definiert Interaktionen zwischen Objekten

- erste Produktentwicklungen Anfang der 90er, z. B. 1993 *Orbix* (erste, kommerziell verfügbare CORBA-Implementierung, für C und C++) von *IONA*

CORBA (2) - Referenzmodell

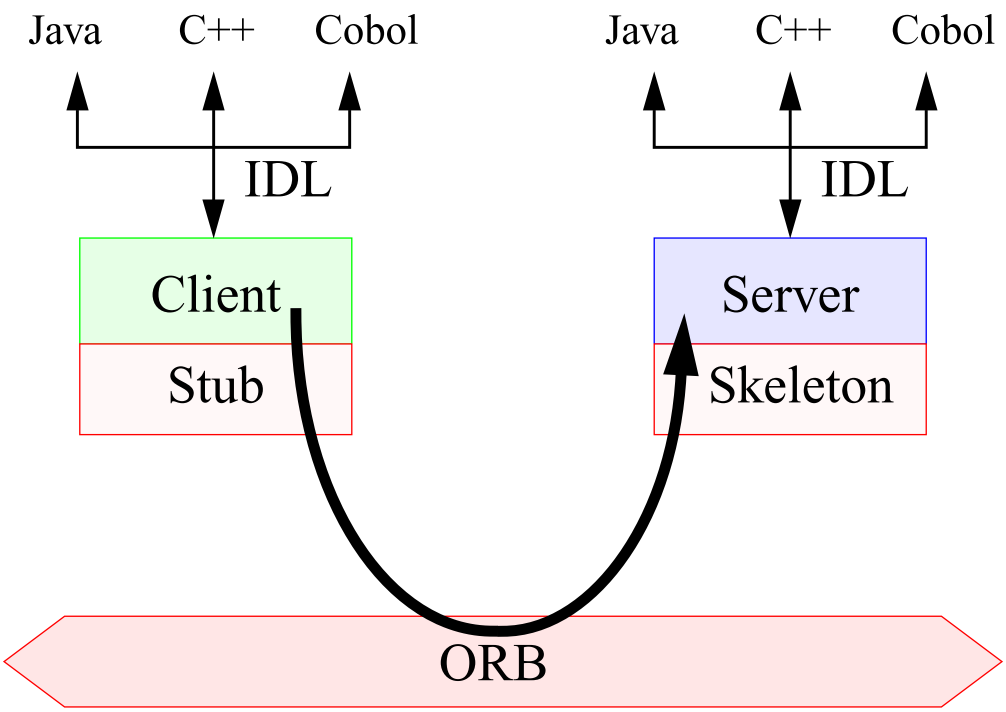
- Object Management Architecture (OMA)



- Interfaces in unterschiedlichen Kategorien
  - Object Services (horizontal)
  - Domain Interfaces (vertikal)
    - Telekommunikation
    - Finanzdienstleistungen
    - E-Commerce
    - Medizin
    - ...
  - Application Interfaces

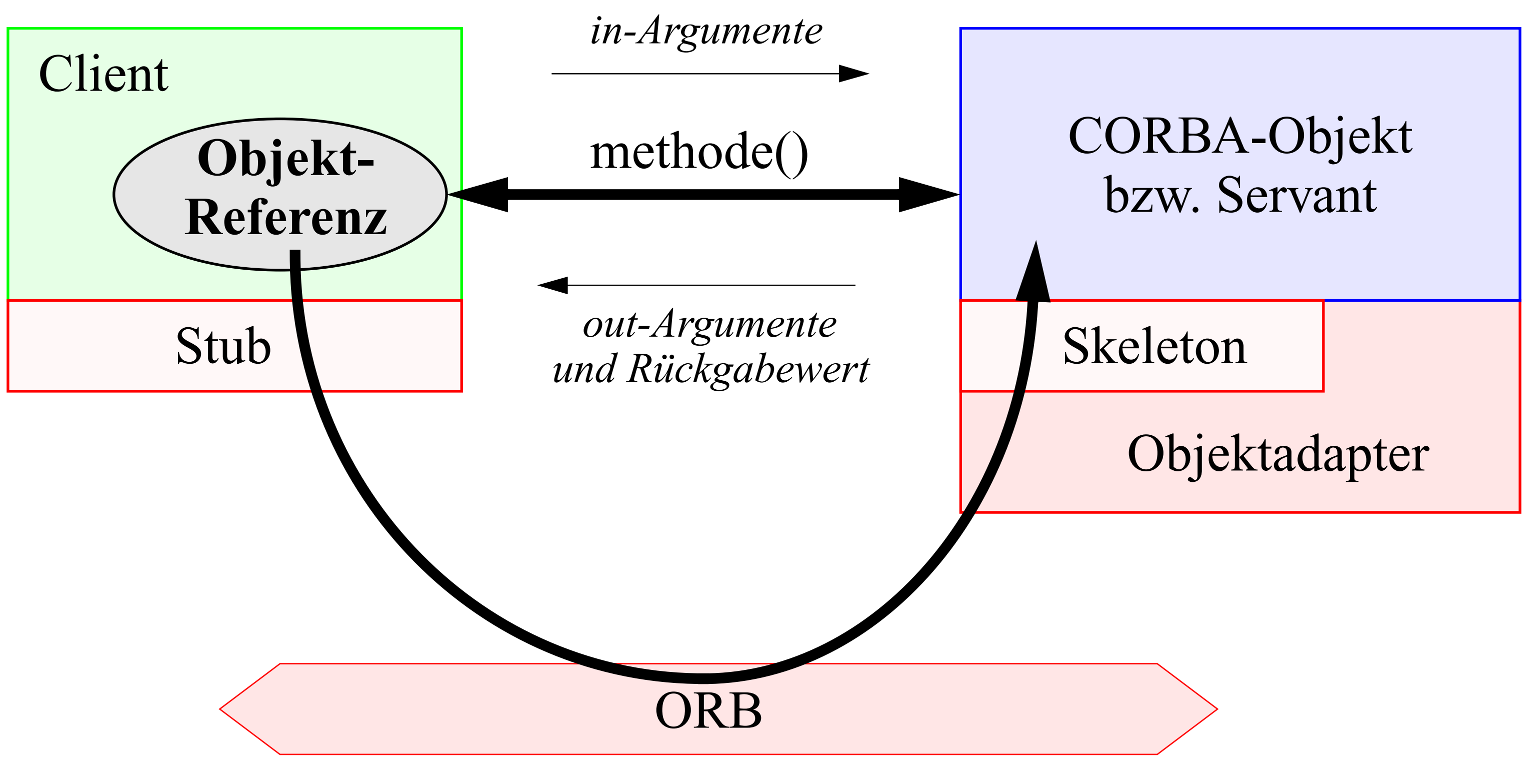
CORBA (3) - Ortstransparenz

- Object Request Broker (ORB)
  - Vermittlung der Dienstaufrufe zwischen verschiedenen Interfaces



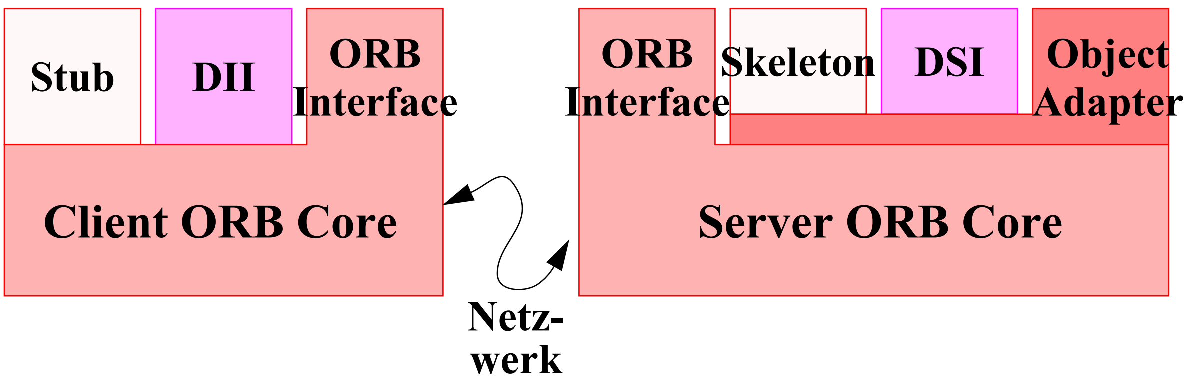
CORBA (4) - Ortstransparenz

- Kommunikation



CORBA (5) - Ortstransparenz

- CORBA-Kernkomponenten
  - Objektreferenzen (Interoperable Object References, IOR)
  - Object Request Broker (ORB)
  - Objektadapter
  - Stubs und Skeletons
  - Dynamic Invocation/Skeleton Interface (DII/DSI)



  - Dienst-spezifisch
    - Stub
    - Skeleton
  - identisch für alle Anwendungen
    - ORB Interface
    - DII
    - DSI

CORBA (6) - Ortstransparenz

- IOR
  - jede IOR identifiziert genau eine Objektinstanz
  - unterschiedliche IORs können auf dieselbe Objektinstanz verweisen
  - eine IOR kann NIL sein
  - *dangling IOR* möglich
  - der interne Aufbau von IOR bleibt den Clients verborgen
  - IORs sind streng getypt (erlaubt Typprüfung zur Compilezeit)
  - IORs unterstützen *late binding* (und damit Polymorphismus)
  - IORs können persistent sein
  - der interne Aufbau ist standardisiert
  - IORs können vom ORB in einen String mit standardisiertem Format umgewandelt und daraus wieder rekonstruiert werden
- ORB
  - übernimmt die Netzwerkkommunikation und das Verbindungsmanagement
  - verwaltet Stubs (Client-Seite)
  - bildet Methodenaufrufe auf Objektadapter ab (Server-Seite)
  - bietet Hilfsfunktionen an (z. B. Konvertierung von Objektreferenzen)
  - Scheduling von Dienstaufrufen auf Threads

CORBA (7) - Ortstransparenz

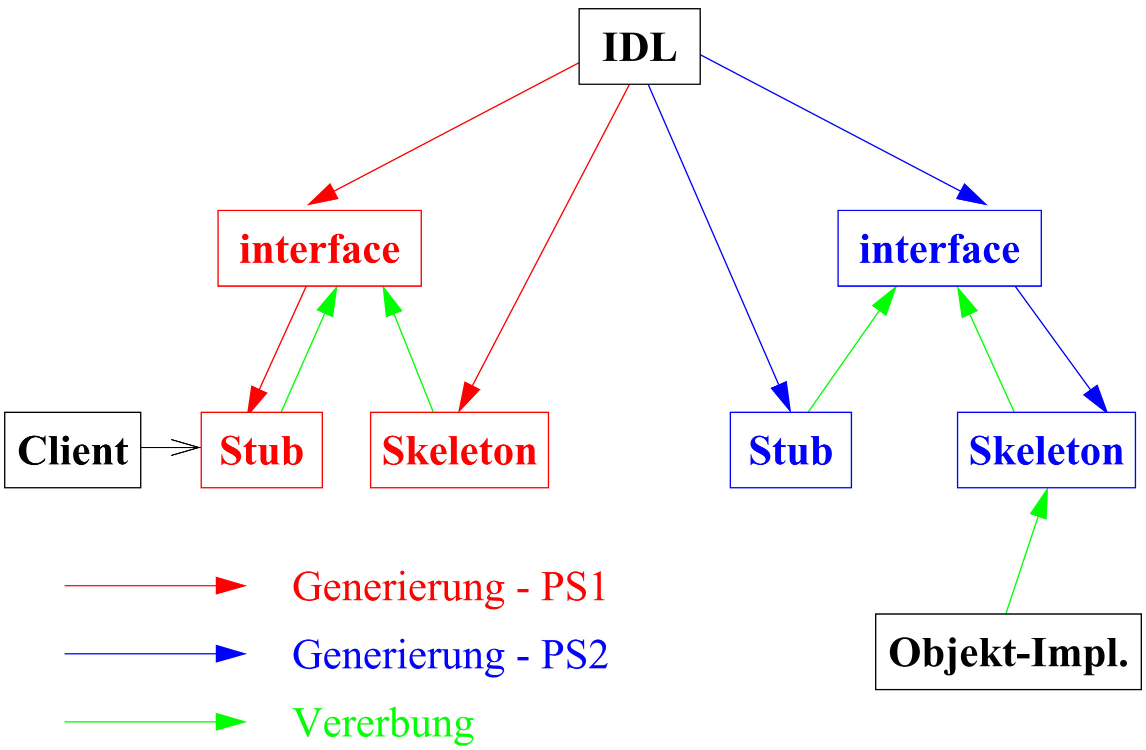
- Objektadapter
  - erzeugt Objektreferenzen
  - bildet CORBA-Methodenaufrufe auf Servants ab
  - aktiviert/deaktiviert Servants (ggf. mit Anwendungsunterstützung)
  - Unzulänglichkeiten des *Basic Object Adapter* (BOA, CORBA 1.0)
    - keine portable Verknüpfung von Skeletons und Servants
    - undefinierte Registrierung der Servants
    - keine Berücksichtigung von Multithreading
    - unpräzise Spezifikation der Server-Zustände (Bereitschaft)
  - *Portable Object Adapter* (POA)
    - aktuell
    - bereits in Produkten, wie Orbix2000 enthalten
- ORB + Objektadapter realisieren den Request Broker

CORBA (8) - Ortstransparenz

- statische Objektaufrufe
  - Stub und Skeleton werden aus der Dienstbeschreibung generiert
  - Skeleton bildet generischen Aufruf des Objektadapters auf tatsächlichen Methodenaufruf ab
  - statische Typprüfung zur Compilezeit
  - effizienter als dynamische Aufrufe (s. u.)
- dynamische Objektaufrufe
  - Objektschnittstellen/Objektaufrufe werden zur Laufzeit ermittelt/generiert (rein syntaktisches Matching, z. B. über ein Interface Repository)
  - DII und DSI sind unabhängig von Dienstbeschreibungen
  - Typüberprüfung zur Laufzeit beim Aufruf
  - unterstützen Flexibilität

CORBA (9) - Sprachunabhängigkeit

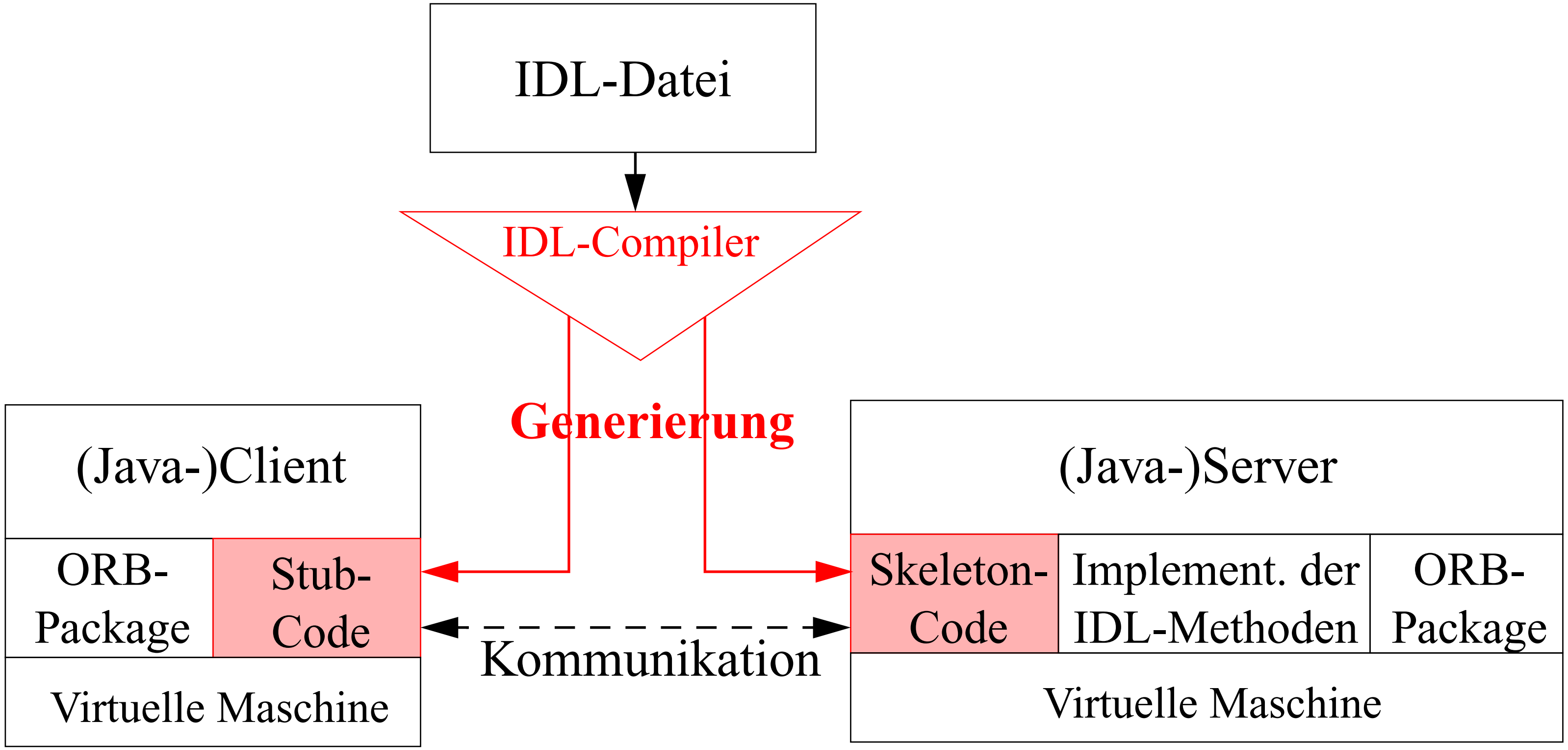
- Ziel: Entwicklung von Clients und Servern in unterschiedlichen Programmiersprachen
  - Trennung von Interface und Implementierung
  - Beschreibung des Interface in Programmiersprachen-unabhängiger *Interface Definition Language* (IDL)
  - Codegenerierung: Stubs, Skeletons
  - Language Mappings; normalerweise IDL-Compiler pro Programmiersprache (PS)



- IDL definiert (rein deklarativ):
  - Typen
  - Konstanten
  - Objekt-Interfaces
    (Attribute, Methoden und Exceptions)

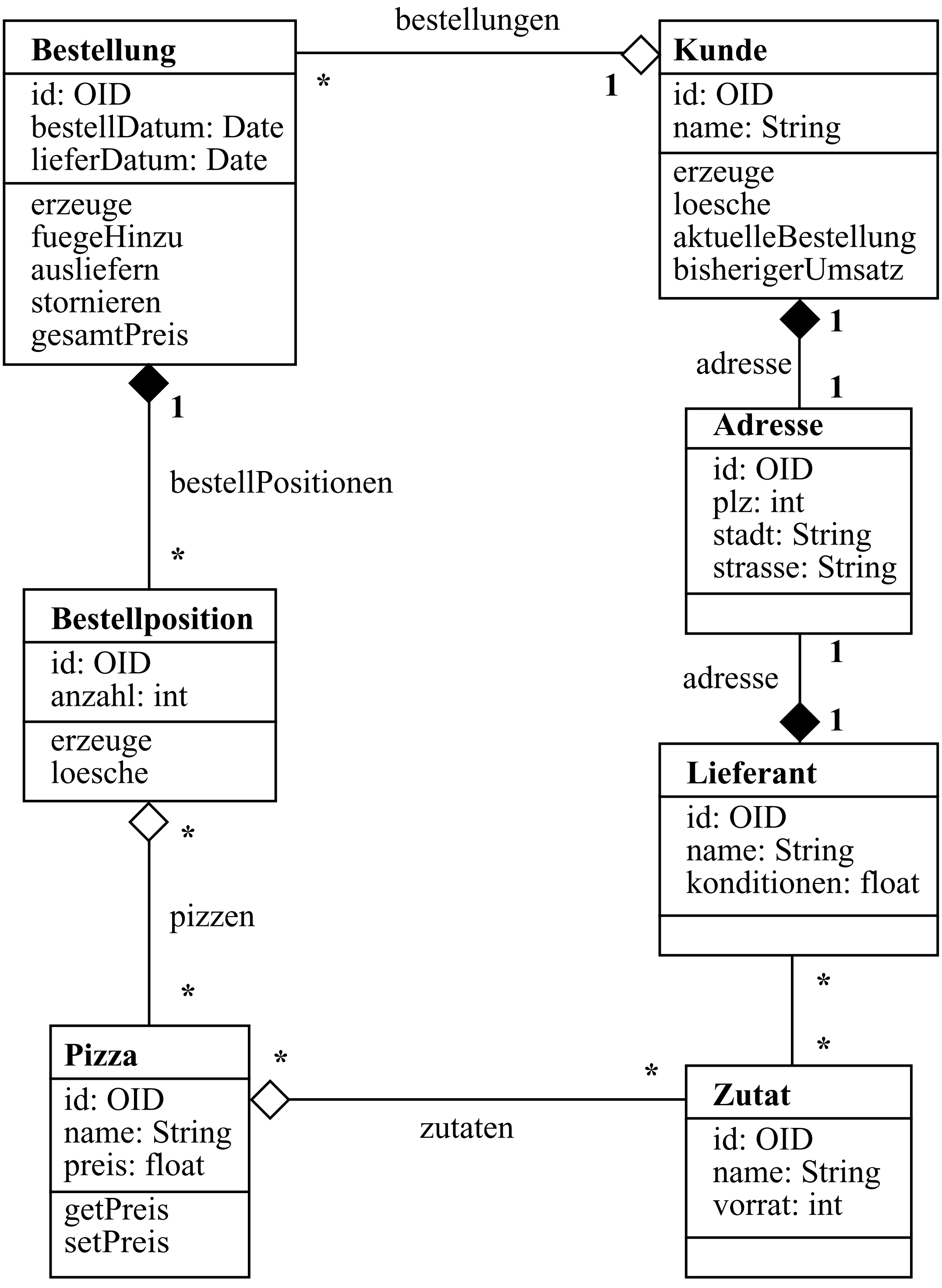
CORBA (10) - Sprachunabhängigkeit

- Interface Definition Language (IDL)



CORBA (11) - Sprachunabhängigkeit

- Beispiel
  - Klassendiagramm *Pizza-Service*



CORBA (12) - Sprachunabhängigkeit

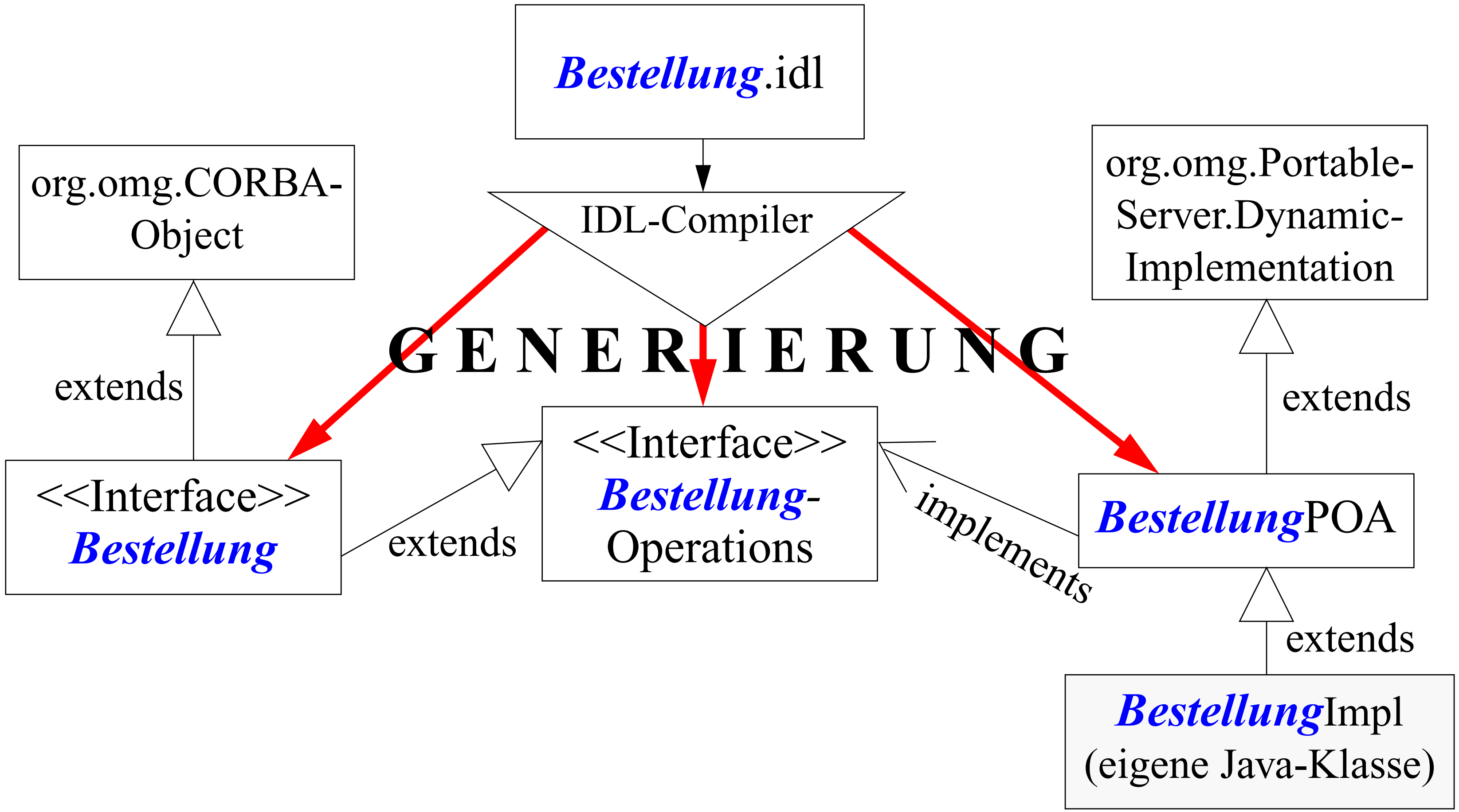
- Beispiel (Forts.)
  - IDL-Spezifikation *Bestellungsdienst*

```
module PizzaService {
    interface Bestellungsdienst {
        void neueBestellung(in long kundennr, out long bestellNr);
        void weitereBestellPosition( in long bestellnr,
                                     in long pizzanr,
                                     in long anzahl);
        };
    interface Auslieferungsdienst {
        long auslieferung(in long kundennr);
        };
    };

interface Bestellung {
    readonly attribute long id;          // nur Get-Methode
    attribute Datum lieferDatum;         // Datum sei ein IDL-Interface
    void neueBestellPosition(in long pizzaId, in long pizzaAnz);
    };
```

CORBA (13) - Sprachunabhängigkeit

- Beispiel: Codegenerierung



CORBA (14) - Plattformunabhängigkeit

❑ Dienstreferenzen:

➩ enthalten alle Informationen zum Senden eines Requests

- minimal
  - Aufenthaltsort des Dienstes (IP-Adresse des Rechnerknotens, Identifikation des Prozesses)
  - vom Server unterstütztes Interaktions-/Kommunikationsprotokoll
- ergänzend
  - alternative Kommunikationsprotokolle
  - weitere Anlaufstellen (z. B. bei einem Server-Lastverbund)
  - Dienstgüteanforderungen
  - herstellerspez. Information ohne Einfluß auf die Interoperabilität
- sollte erweiterbar sein

➩ *Interoperable Object Reference* (IOR)

❑ "On the wire"-Datenformat:

➩ Aufgaben

- definiert die Kodierung von Datentypen
- definiert die Verantwortlichkeit für notwendige Konvertierungen (kononisch vs. "receiver makes it right")

➩ *Common Data Representation* (CDR)

CORBA (15) - Plattformunabhängigkeit

- Kommunikationsprotokoll
  - definiert die Interaktionen zwischen Client und Server
    - Nachrichtenaufbau
    - Nachrichtensequenzen
  - CORBA 2.0: General Inter-ORB Protocol (GIOP)
  - Internet-Inter-ORB-Protocol (IIOP)
    - konkretisiert GIOP für TCP/IP
    - Intenret als "Backbone-ORB"
  - Kür
    - Environment-Specific Inter-ORB Protocols (ESIOP)
      Beispiel: DCE Common Inter-ORB Protocol (DCE-CIOP)

CORBA (16) - CORBA Services

- Namensdienst
  - Fragestellungen
    - Wie findet ein Client seine Server?
    - Wie kann ein Server seinen Dienst anbieten?
    - Abbildung symbolischer Namen auf Dienstreferenzen (= Bindungen)
  - Analogie: Telefonbuch
  - hierarchische Namen
    - Kontexte (Mengen von Bindungen) sind CORBA-Objekte
    - alle Bindungen verweisen auf Dienstreferenzen
  - standardisierte Schnittstellen
    - Abfragen/Eintragen/Entfernen/Ändern von Bindungen
    - Iterieren über alle Bindungen eines Kontextes
  - "Namensgraph" verfügt über mindestens einen initialen Kontext
    - Einstiegspunkte in den Namensgraph
  - unterstützt verteilte Namensdienste
    - da Kontexte wiederum CORBA-Objekte sind
    - initialer Namenskontext über ObjektID ***NameService***

CORBA (17) - CORBA Services

- Trading
  - Situation
    - Client kennt den Typ des benötigten Dienstes, nicht aber dessen genauen Namen
  - Analogie: Branchenbuch
  - Trader
    - speichert Dienstangebote in *Service-Type Repository*
      - Diensttyp(hierarchie)
      - Dienstbeschreibung
      - Dienstreferenz
    - unterstützt Suchanfragen und Iteration über Dienstangebote
    - kann selbst verteilt sein

CORBA (18) - CORBA Services

- Security
  - Aufgaben
    - Authentisierung, Identität sicherstellen
      - von Benutzern
      - von Diensten
    - Autorisierung, Zugriffskontrolle
    - Sicherheits-Log
    - Verwaltung von Sicherheitseinstellungen
    - Verschlüsselte Datenübertragung
    - Unleugbarkeit, Verantwortlichkeit
  - unterschiedliche Sichten
    - Client-Anwendung (Authentisierung)
    - Server-Anwendung (Rechte, Kontrollen)
    - Systemverwalter
      (Verwaltung von Zugriffsrechten, Zugriff auf Log)
    - Schutzdienst-Entwickler
      (interne Nutzung von Schutzmechanismen)

CORBA (19) - CORBA Services

- Security (Forts.)
  - Level 1: Schutz transparent für die Anwendung
    - auf einem sicheren ORB
    - Authentisierung von Benutzern durch das System
    - Zugriffskontrolle
    - sichere Übertragung
    - Logging
  - Level 2: Anwendung selbst nutzt Schutzdienst
    - Client
      - Authentisierung von Benutzern
      - Kontrollierte Weitergabe von Rechten
    - Server
      - Ändern von Zugriffsrechten
      - Prüfen von Identität und Rechten von Aufrufern
    - optional: Unleugbarkeit (Non-Repudiation)

CORBA (20) - CORBA Services

- Security (Forts.)
  - Sicherer Objektaufruf
    - Aufbau einer Sicheren Verbindung
      - beidseitige Authentisierung,
      - Mitteilung der Berechtigungen des Clients an den Server
      - Schutz vor Modifikation der Nachrichten,
        z. B. Signaturen
      - Schutz vor Abhören (Verschlüsselung)
    - Zugriffskontrolle und Logging
      - auf Client- und Server-Seite möglich
      - durch den ORB und möglicherweise durch die Anwendung

CORBA (21) - CORBA Services

- Security (Forts.)
  - Berechtigungen
    - Credential-Objekte
    - Sicherheitsmerkmale
      - Identität
      - Funktion des Benutzers (z. B. Administrator)
      - Gruppe
      - Autorisierungsrang (z. B. ‘geheim’)
      - Capabilities
        (Recht, bestimmte Methoden bestimmter Objekte aufzurufen)
      - andere
    - Erlangen von Berechtigungen
      - frei zugänglich
      - durch Authentisierung
      - von einem Client übertragen (Delegation)

CORBA (22) - CORBA Services

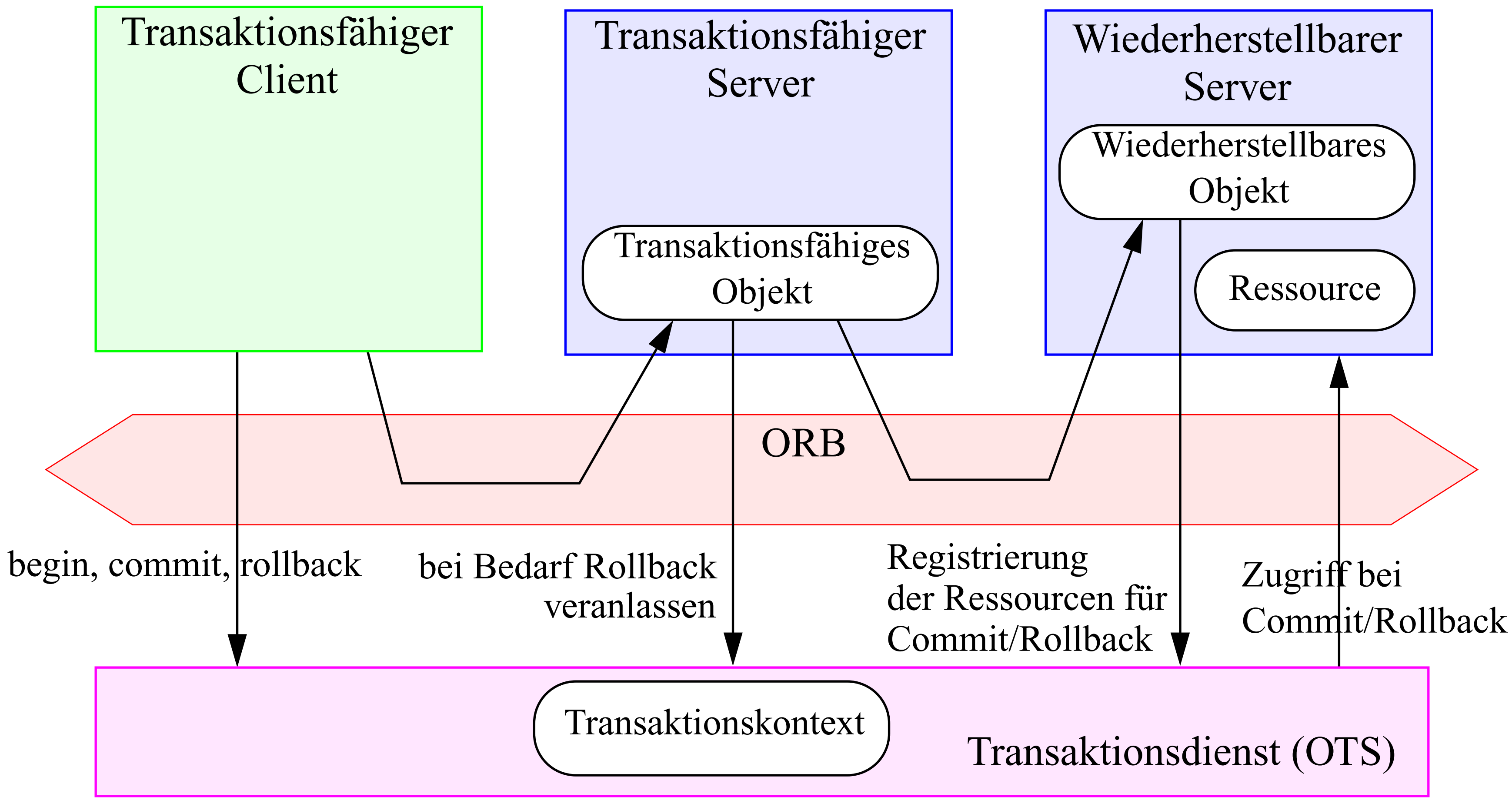
- Security (Forts.)
  - Autorisierung, Client
    - Level 1
      - automatische Zuordnung von Credentials bei Login des Benutzers
    - Level 2
      - Aufrufe mit eingeschränkten Rechten möglich
      - (De-)Aktivierung von Sicherheitsfunktionalität
  - Autorisierung, Server
    - Level 1
      - Objekte sind einer Security Domain mit festgelegten Policies zugeordnet
      - ORB prüft Zugriffe gemäß dieser Policies basierend auf vorgelegten Berechtigungen sowie Zielobjekt/Methode
    - Level 2
      - Current-Objekt enthält Aufrufkontext (liefert Credentials des Aufrufers)
      - Änderung der Zugriffs-Policies für eigene Objekte
      - Zugriffskontrolle durch Anwendung selbst (flexibler)
  - Sichere Protokolle
    - Secure Inter-ORB-Protocol (SECIOP)
    - SSL

CORBA (23) - CORBA Services

- Transactions
  - flache, geschachtelte Transaktionen
  - X/OPEN DTP
  - Aufgaben des Object Transaction Service (OTS) entsprechen im wesentlichen denen des TA-Managers im X/OPEN-Modell

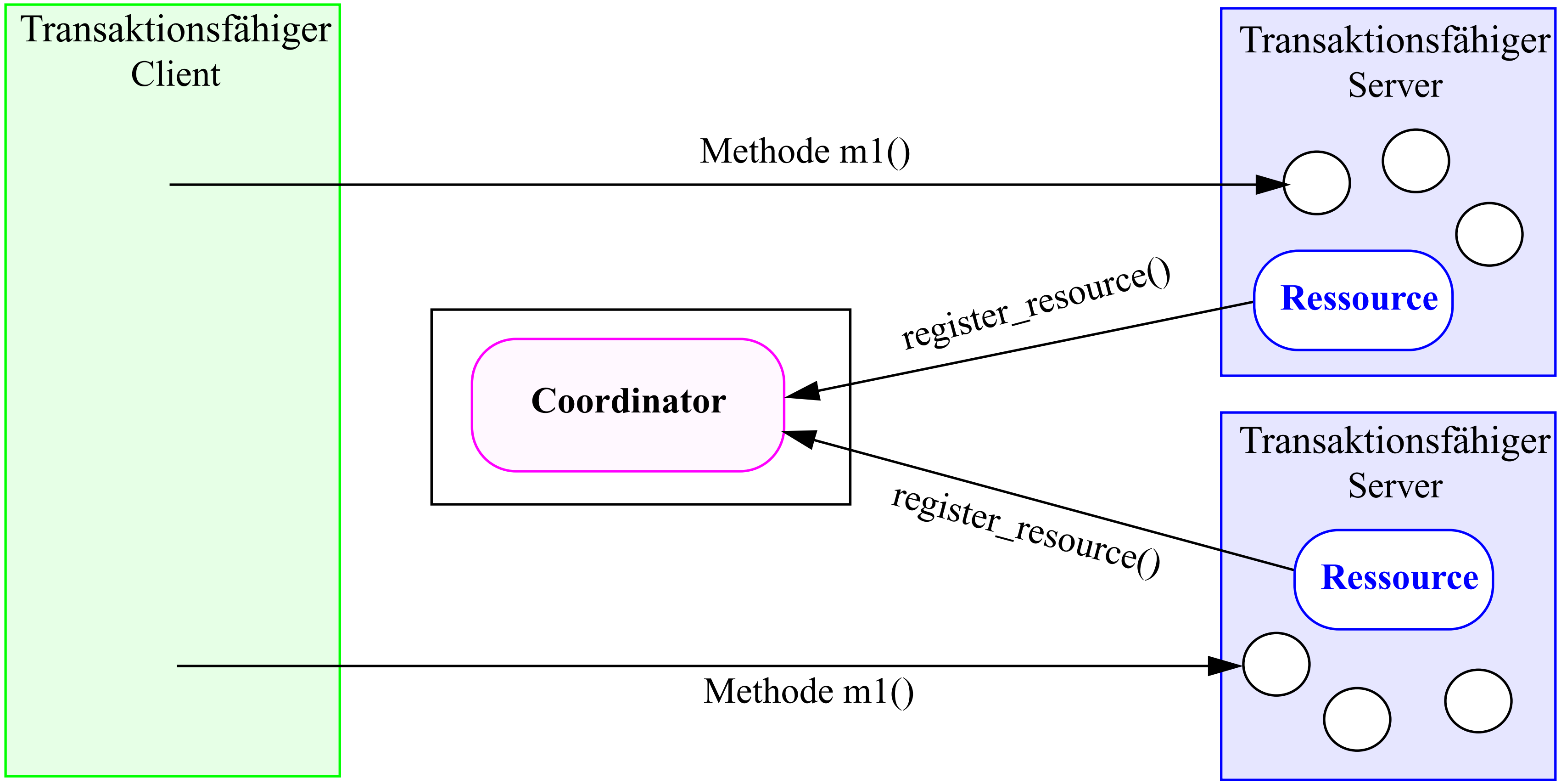
CORBA (24) - CORBA Services

- Transactions (Forts.) - Object Transaction Service (OTS)



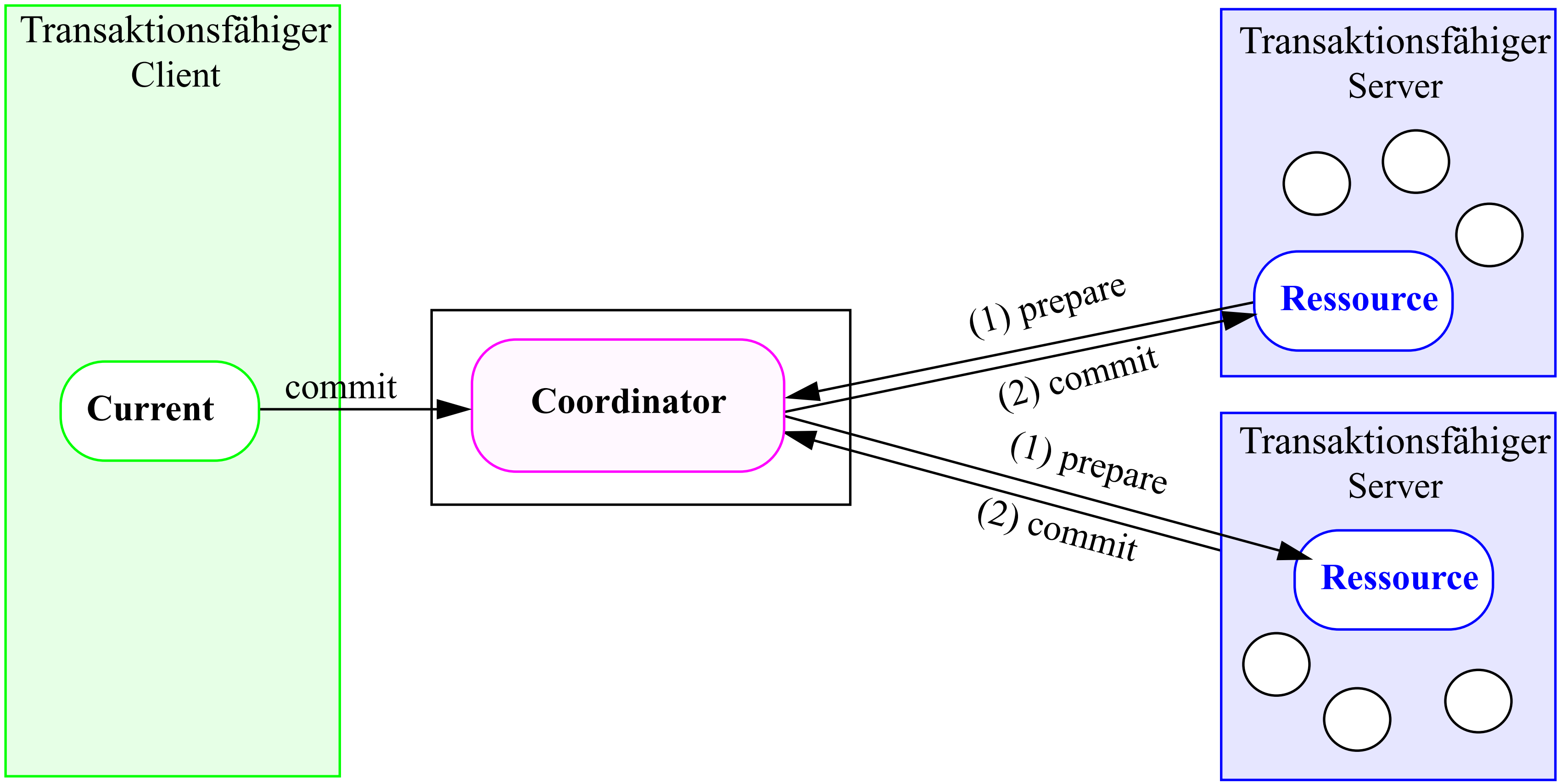
CORBA (25) - CORBA Services

- Transactions (Forts.) - OTS - Schnittstellen



CORBA (26) - CORBA Services

- Transactions (Forts.) - OTS - Schnittstellen



CORBA (27) - CORBA Services

- Concurrency
  - Interfaces zum Anfordern und Freigeben von Sperren auf *shared resources*
  - falls im Dienst eines transaktionalen Clients:
    Freigabe von Sperren wird durch den OTS veranlaßt
  - falls im Dienst eines nicht-transaktionalen Clients:
    Verantwortlichkeit für die Sperrfreigabe bei Client
  - Serialisierbarkeit
  - R/X/Intention/U-Sperren
  - Methode *try_lock* gibt Kontrolle zurück, statt zu blokkieren
  - Locksets
    - Menge von Sperren , die später als Einheit freigegeben werden sollen
    - mehrere Locksets können miteinander assoziiert werden
  - Lock-Coordinator
    - koordinierte Freigabe von Locksets
    - wird vom OTS genutzt

CORBA (28) - CORBA Services

- ❑ Persistent Objects
  - ➩ Ziel: einheitliches Interface für unterschiedliche Datenspeicher
  - ➩ Komponenten des POS (*Persistent Object Service*)
    - PO: *Persistent Object*
      - werden identifiziert durch *PID* (*persistent object identifyer*)
      - PID beschreibt Objekt-Lokation
    - POM: *Persistent Object Manager*
      - Mediator zwischen POs und PDS
      - realisiert Interface für Persistenz-Operationen
      - interpretiert PIDs
      - implementierungsunabhängig
    - PDS: *Persistent Data Store*
      - Mediator zwischen POM und persistenten Datenspeichern
      - Austausch von Daten zwischen Objekt und Datenspeicher
      - implementierungsabhängig

CORBA (29) - CORBA Services

- Weitere Services
  - Query Service
    - mengenorientierte Anfragen zum Auffinden von CORBA-Objekten
    - SQL, OQL
    - Query-Ergebnisse werden durch *Collection*-Objekte repräsentiert
  - Relationship Service
    - Verwaltung von Objektabhängigkeiten
    - Relationship: Typ, Rollen, Kardinalitäten
  - Object Lifecycle Service
    - Erzeugen, Löschen, Kopieren und Migrieren von einfachen und strukturierten Objekten
  - Event Service
    - Event-Kanäle zwischen *suppliers* and *consumers*
    - *supplier* produzieren Events
    - *consumer* verarbeiten Events mittels *Event-Handler*
    - *Push* und *Pull*-Modelle

CORBA (30) - CORBA Services

❑ Weitere Services (Forts.)

- Externalization Service
  - Abbildung zwischen Objektzustand und Bytestrom
- Licensing Service
  - unterstützt Vielzahl von Lizenzmodellen
- Property Service
  - Verwaltung von Eigenschaften als Paare von Name und Wert (außerhalb der IDL-Strukturen)
  - z. B. Markieren eines Objekte mit “ready to be archived”
- Time Service
  - Uhr, Zeitintervalle, Timer Events
- Change Management Service
  - einfache Versionierung

CORBA (31) - Zusammenfassung

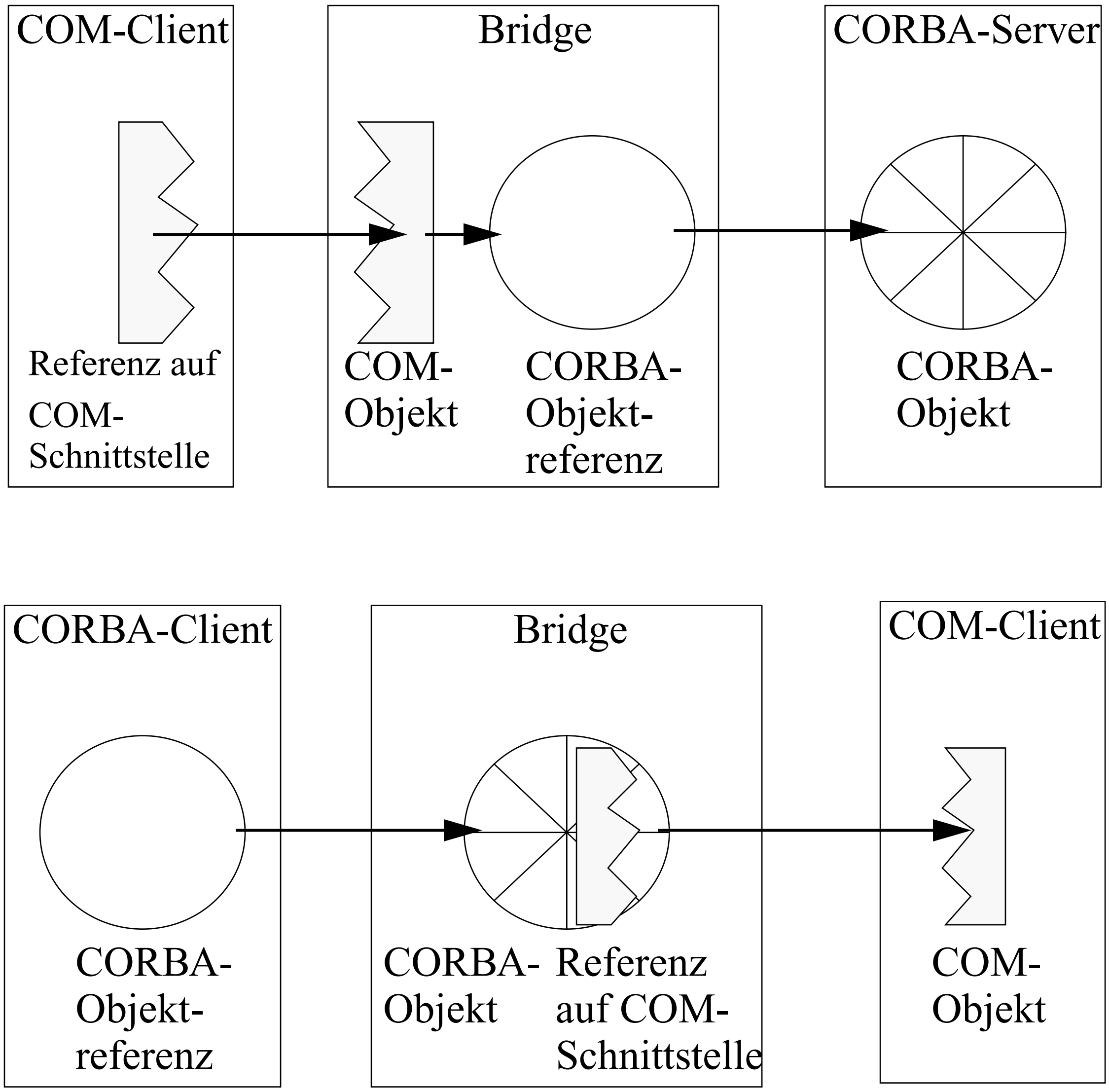
- universellste aller Komponenten-Architekturen
  - Ortstransparenz
  - Plattformunabhängigkeit
  - Sprachunabhängigkeit
- DB-Bezug durch DB-orientierte Basisdienste, wie OTS, Concurrency Service, Persistent Object Service, Query Service, Security Service, ...

(D)COM/COM+

- ❑ COM+ erstmals mit *Windows 2000* ausgeliefert
- ❑ Integration von
  - ➪ Component Object Model (COM, 1993)
    - Kommunikation von COM-Komponenten innerhalb eines Prozesses oder zwischen verschiendenen Prozessen auf dem gleichen Rechnerknoten
    - COM-Objekt kann mehrere Schnittstellen bereitstellen
    - jede Schnittstelle enthält Menge funktional zusammengehörender Methoden
    - COM-Client interagiert mit COM-Objekt, in dem eine Referenz auf eine der Schnittstellen benutzt wird
    - jede Schnittstelle muß bestimmtem Speicherlayout folgen (vgl. *Virtual Function Table* in C++)
    - Spezifikation auf binärer Ebene ermöglicht Nutzung verschiedener Programmiersprachen (heute vor allem *Java, C++, VisualBasic*)
  - ➪ Distributed COM (DCOM, 1996)
    - Kommunikation über verschiedene Rechnerknoten hinweg
    - baut auf DCE-RPC auf
  - ➪ Microsoft Transaction Server (MTS, 1996)
    - ermöglicht Kommunikation über DCOM mit Windows- und Unix-Clients
    - Speicherung von Objekten und Koordination verteilter TA
    - Applikationsserver-Laufzeitumgebung
  - ➪ *COM+ Services*

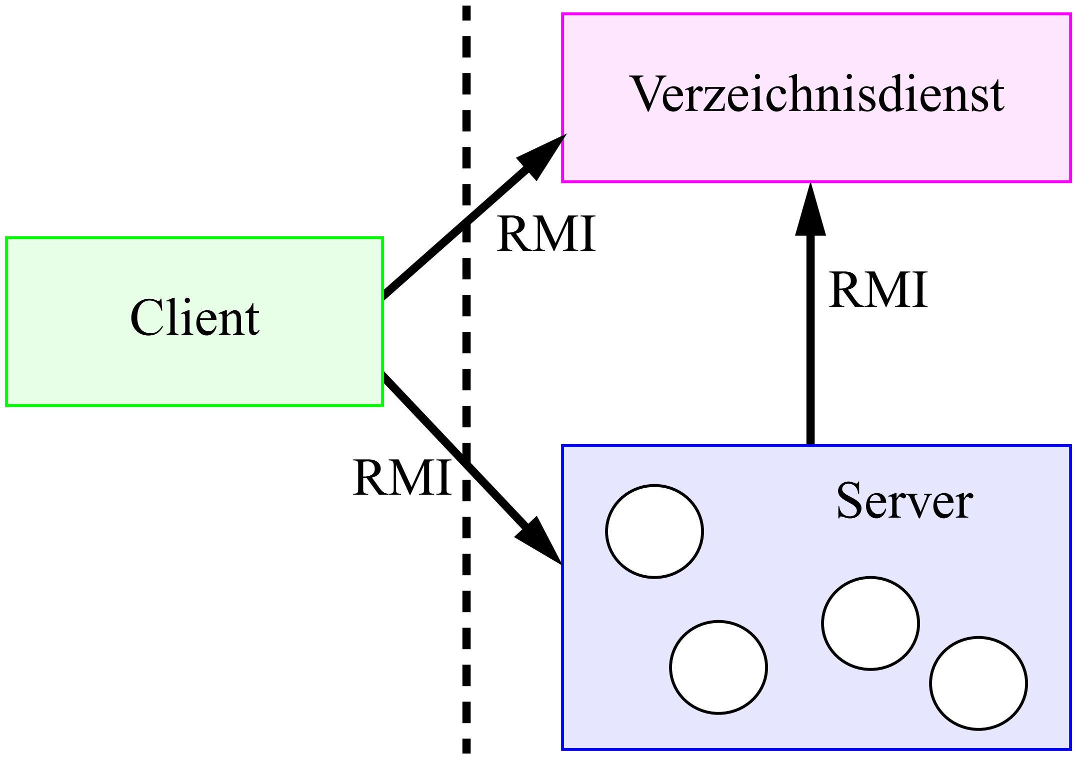
CORBA-COM-Bridge

- Bridge



Java-RMI (1)

- Mechanismus zur Kommunikation
  - zwischen Java-Programmen
  - zwischen Java-Programmen und Applets
- Kommunikationsbeziehungen



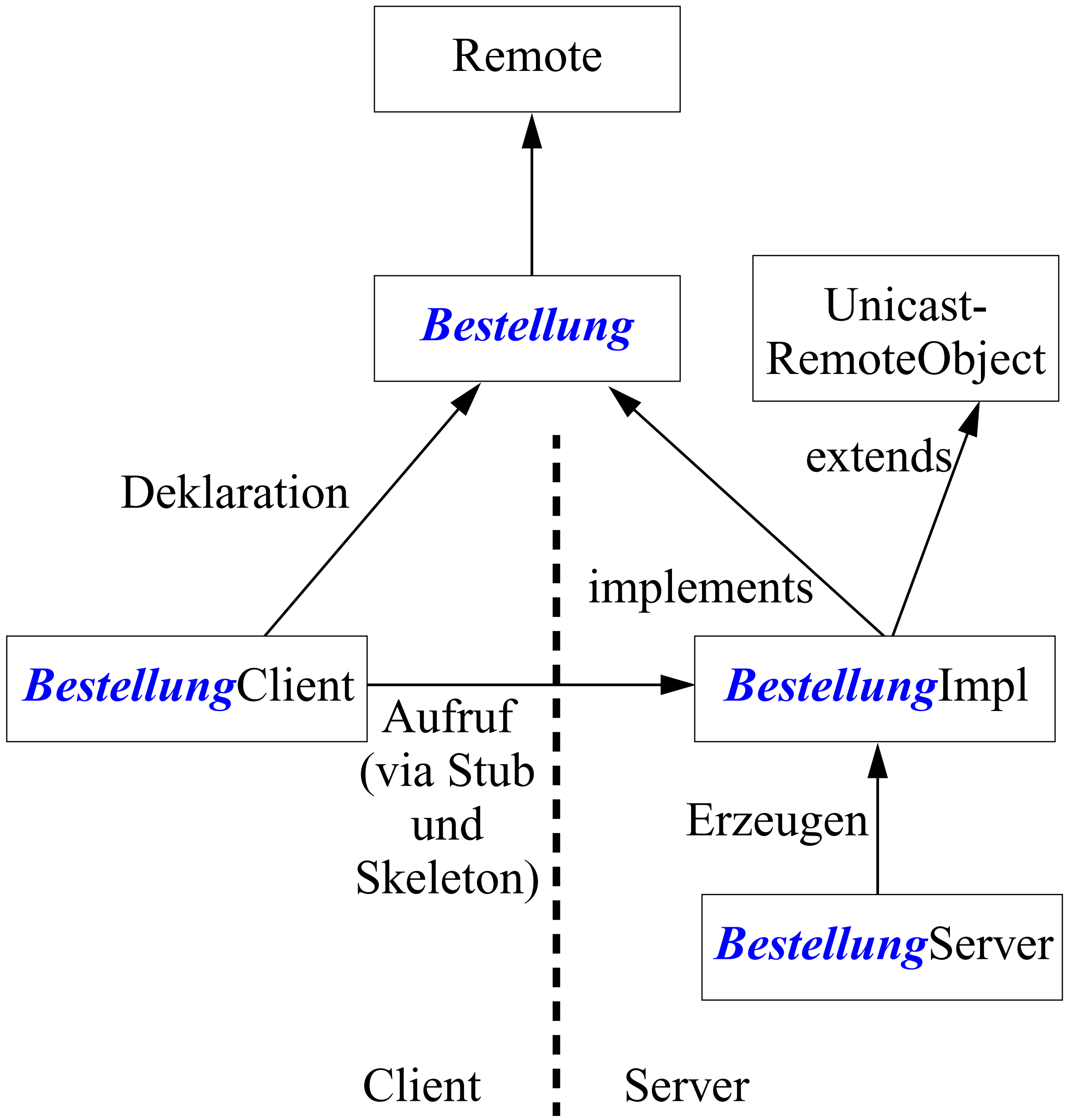
- Aufgaben
  - Lokalisierung entfernter Objekte
  - Transparente Kommunikation mit entfernten Objekten
  - Laden von Java-Bytecode für entfernte Objekte

Java-RMI (2)

- Stub
  - Herstellen der Verbindung mit der VM, die das entfernte Objekt enthält
  - *Marshalling* der Methodenargumente
  - Warten, bis Server die Methode abgearbeitet hat und Resultate zurückgibt
  - *Unmarshalling* (Rückgabewert oder Ausnahme)
  - Weiterreichen des ermittelten Wertes an die ursprüngliche Aufrufstelle des Clients
- Skeleton
  - *Unmarshalling* (Argumente)
  - Ausführung der Methode
  - *Marshalling* (Resultat)

Java-RMI (3)

- Beispiel - Zusammenspiel der Klassen



Java-RMI (4)

- Beispiel (Forts.)

```
import java.rmi.*;
import java.util.Date;
public interface Bestellung extends Remote {
   public void neuebestellPosition(int pizzaId, int anzahl)
                                              throws RemoteException;
   public Date getLieferDatum() throws RemoteException;
   public Date setLieferDatum(Date neuesDatum) throws RemoteException;

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*
public class BestellungImpl
   extends UnicastRemoteObject
   implements Bestellung {
      private Vector fBestellPositionen;
      private Date fLieferDatum;
      public BestellungImpl(String name) throws RemoteException {
         super();
         try {
            Naming.rebind(name, this); // Registrierung bei Nameserver
            fBestellPositionen = new Vector();
            fLieferDatum = null;
         }
         catch (Exception e) {
            System.err.println(“Ausnahme: “ + e.getMessage());
            e.printStackTrace();
         }
      }
      public void neueBestellPosition(int pizzaId, int anzahl )
         throws RemoteException {
         // die Klasse bestellPosition sei bekannt
         BestellPosition bestellPosition = new BestellPosition(pizzaId, anzahl);
         fBestellPositionen.addElement(bestellPosition);
      }
      ... // Impl. v. getLieferDatum und setLieferDatum
}
```

Java-RMI (5)

❑ Beispiel (Forts.)

```
import java.rmi.*;
public class BestellungClient {
    public static void Main(String args[]) {
        try {
            Bestellung bestellung = (Bestellung)
                        Naming.lookup(“rmi://berlin:9000/meine_best”);
            int pizzaId = Integer.parseInt(args[0]);
            int anzahl = Integer.parseInt(args[1]);
            bestellung.neueBestellPosition(pizzaId, anzahl);
        }
        catch (Exception e) {
            System.err.println(“Systemfehler: “ + e);
        }
    }
}
```

```
import java.rmi.*;
import java.server.*;
public class BestellungServer {
    public static void main(String args[]) {
        try {
            BestellungImpl bestellung = newBestellungImpl(“meine_best”);
            System.out.println(“Der Bestellungsserver laeuft”);
        }
        catch (Exception e) {
            System.err.println(“Ausnahme: “ + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Java-RMI (6)

- Beispiel (Forts.)
  - Übersetzung Quellcode in Java-Bytecode:
    javac Bestellung.java BestellungImpl.java BestellungClient.java BestellungServer.java
  - Erzeugung von Stub- und Skeleton-Code:
    rmic BestellungImpl
  - administrative Schritte:
    - Starten des Verzeichnisdienstes: ***start rmiregistry***
    - Starten des RMI-Servers: ***start java BestellungServer***
    - Aufruf eines Clients: ***java BestellungClient***

Zusammenfassung

- Verteilte Objektarchitekturen
  - Anwendung besteht aus interoperablen Objektkomponenten, mglw. über Rechnernetz verteilt
  - Remote Method Invocation
  - Zusätzliche Dienste (Transaktionen, Sicherheit, ...)
- CORBA
  - universell, flexibel
  - Ortstransparenz, Plattformunabhängigkeit, Sprachunabhängigkeit
  - CORBA object services - sehr reichhaltig
- (D)COM
  - herstellerabhängig, plattformabhängig
  - Ortstransparenz, Sprachunabhängigkeit
  - object services zusammen mit MTS/COM+
- Java RMI
  - Java-spezifisch
  - Ortstransparenz, Plattformunabhängigkeit
  - object services durch andere Javatechnologien (JNDI, JTS, ...)