Verteilte objektorientierte Programmierung am Beispiel CORBA

Dr. Christian Schröder
Telelogic Deutschland GmbH

Agenda

1. Was versteht man unter einem „verteilten System“?
2. Was ist CORBA?
3. Wie funktioniert CORBA?
4. CORBA IDL
5. CORBA@Work: Eine Beispielanwendung
6. CORBA@Work - Ausblick
7. Zusammenfassung
8. Literatur

1. Was versteht man unter einem „verteilten System“?

1.1 Definition: Verteiltes System (nach Andrew Tanenbaum)

„Ein verteiltes System ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen.“

- impliziert, daß die Computer miteinander verbunden sind und
- die Ressourcen wie Hardware, Software und Daten gemeinsam benutzt werden.
- Es herrscht eine einheitliche Sicht auf das Gesamtsystem vor.

1. Was versteht man unter einem „verteilten System“?

1.2 Definition: Verteiltes System (nach Leslie Lamport*)

„Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil irgendein Rechner abgestürzt ist, von dem ich nicht einmal weiß, daß es ihn überhaupt gibt.“

- oft die Realität
- wird aber besser
- hoffentlich auch durch diese Vorlesung ...

**Leslie Lamport ist der Vater von LaTeX*

1. Was versteht man unter einem „verteilten System“?

1.3 Wo findet man verteilte Systeme?

- Kaffeeautomat, Fensterhebersteuerung (im Auto), Telefonvermittlungsstellen, Flugzeug, Waschmaschine, ...
- Internet!

Besser: Wo findet man *keine* verteilten Systeme (mehr)?

1. Was versteht man unter einem „verteilten System“?

1.4 Warum ist die Programmierung verteilter Systeme so schwierig?

- weil Softwareentwicklung ohnehin schwierig ist (siehe Vorlesung „Softwareengineering“)
- aber es gibt zusätzliche Herausforderungen bei verteilten Systemen:
  - Komponenten der Anwendung sind über ein Netzwerk verteilt
  - laufen auf verschiedenen Betriebssystemen
    - Unix, Windows, MVS, VxWorks, …
  - sind in verschiedenen Sprachen geschrieben
    - VB oder Java für GUIs; C++ für Core, etc.
  - es müssen legacy Komponenten, d.h. „alte“ Anwendungen eingebunden werden
  - Netzwerksicherheit muss gewährleistet sein

⇒ Man benötigt ein Framework, um diese Probleme zu addressieren!

2. Was ist CORBA?

2.1 CORBA im Überblick

- CORBA ist die Spezifikation dieses Frameworks
  - definiert eine einheitliche Architektur für verteilte Objekte
  - Common Object Request Broker Architecture
- CORBA wird von den Mitgliedern der Object Management Group (OMG) definiert
  - OMG ist das weltweit größte IT Konsortium mit über 800 Mitgliedern
  - Telelogic, IBM, IONA, Sun, ...
  - Die OMG koordiniert die Standardisierungsarbeiten verschiedener Objekttechnologien, wie z.B. CORBA, UML, ...
- CORBA hat zwei Hauptziele
  - Vereinfachung der Implementierung von Software in heterogenen Umgebungen
  - Unterstützung von Anwendungsintegration
- Grundlegender Mechanismus von CORBA:

  Bereitstellung von Objekten, die Dienste auf Anfrage anderer Objekte ausgeführen.

2. Was ist CORBA?

2.2 Was ist der Unterschied von CORBA zum Kommunikations-mechanismus in der Objektorientierung?

- Im Grunde genommen gibt es keinen!
- Der Hauptunterschied liegt darin, dass CORBA Objekte überall sein können.
- Das CORBA System stellt einen Mechanismus zur Verfügung, der „lokale Transparenz“ genannt wird.
- Das bedeutet, dass ein Client Objekt überhaupt nicht sagen *kann*, ob die Anfrage an ein Objekt geht, dass sich im selben Prozess, auf demselben Rechner oder am anderen Ende der Welt befindet!
- diese Art Transparenz kennt man seit den 80er Jahren als „Remote Procedure Call (RPC)“.
- CORBA hat die Funktionalität von RPC erweitert und verwendet einen objektorientierten Ansatz (vgl. RMI bei Java als „Konkurrenztechnologie“)

2. Was ist CORBA?

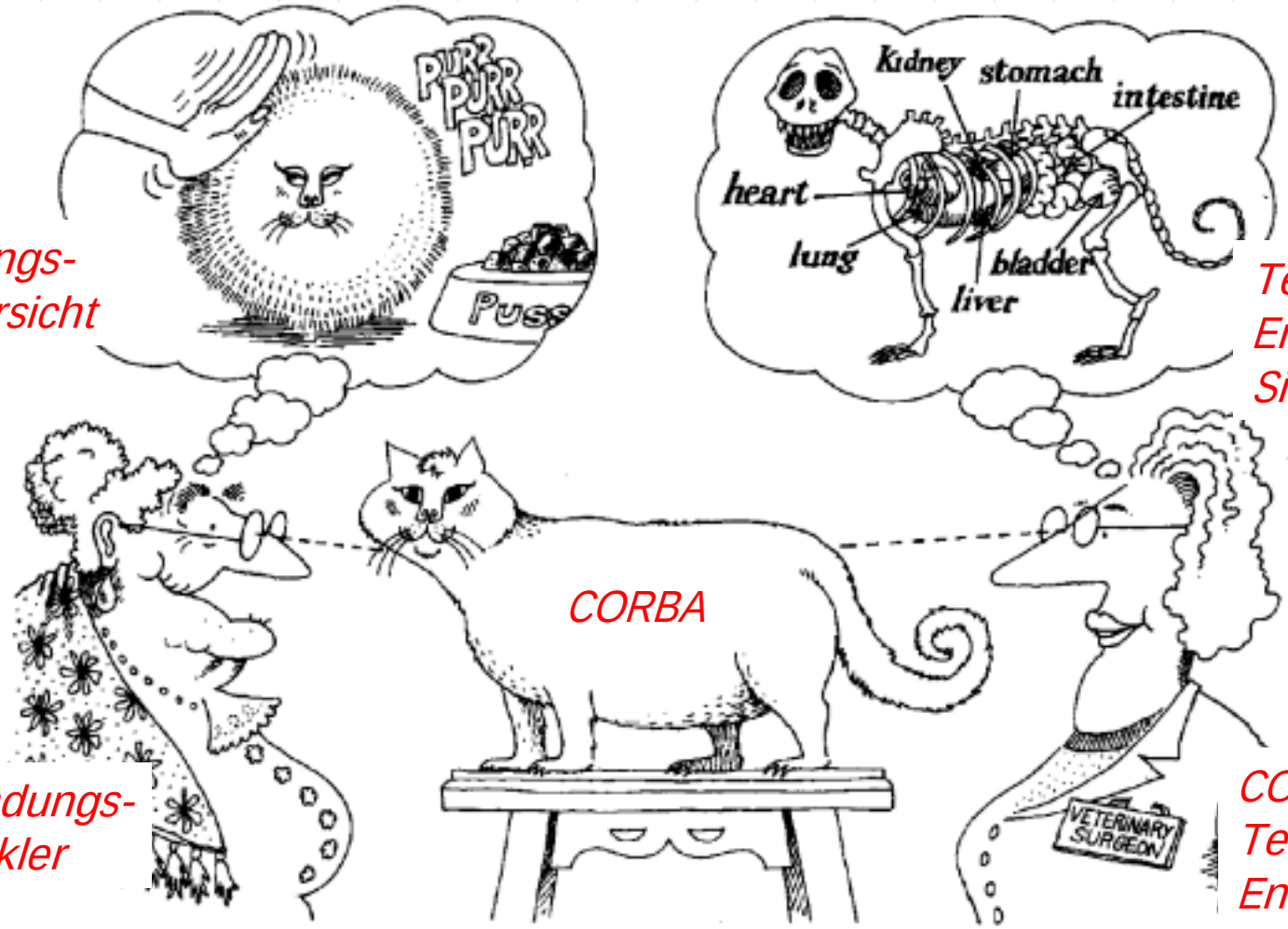
2.3 Was bedeutet das für den Entwickler?

- Eine starke Vereinfachung der Anwendungsprogrammierung!
- Man erreicht entfernte Objekte auf anderen Rechnern, die unter
  - anderen Betriebssystemen laufen und in
  - anderen Sprachen geschrieben worden sind, einfach indem man
  - dieselbe Syntax

  verwendet wie üblich.

CORBA abstrahiert von den zugrundeliegenden Kommunikations-mechanismen, d.h. verbirgt deren (komplexe) Details vor dem Anwendungsentwickler!

2. Was ist CORBA?



Anwendungs-entwicklersicht

Technologie-Entwickler-Sicht

CORBA

Anwendungs-entwickler

CORBA-Technologie-Entwickler

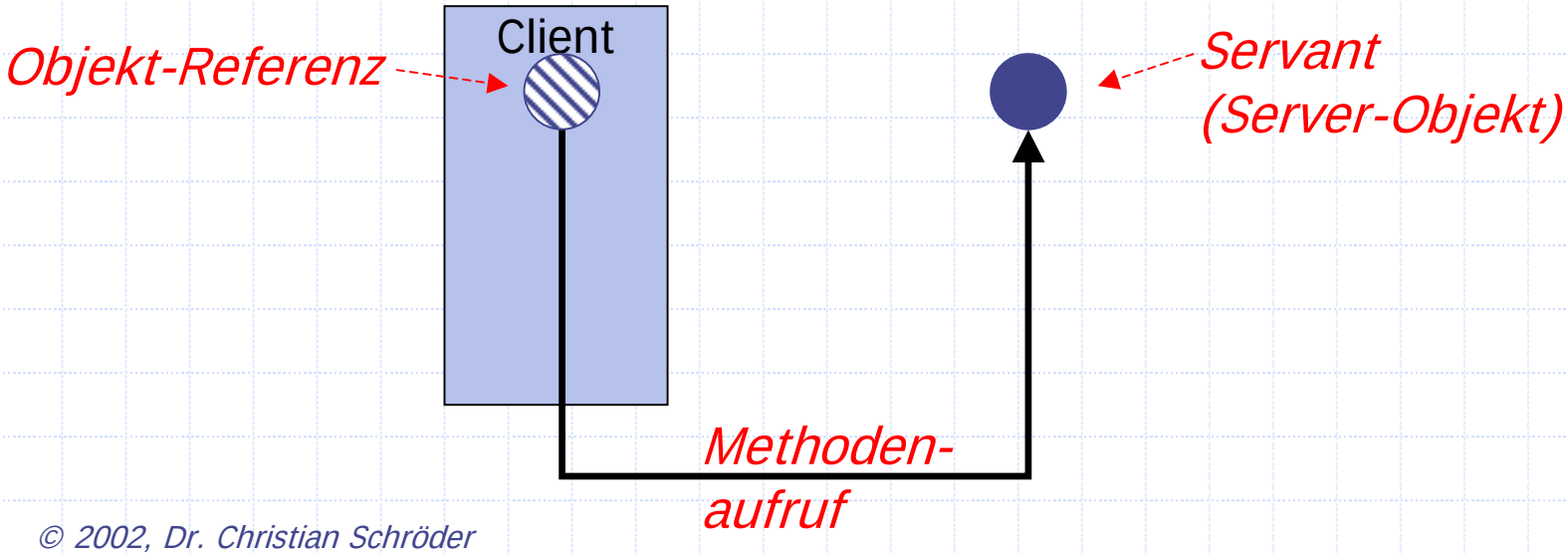
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

3. Wie funktioniert CORBA?

3. Wie funktioniert CORBA?

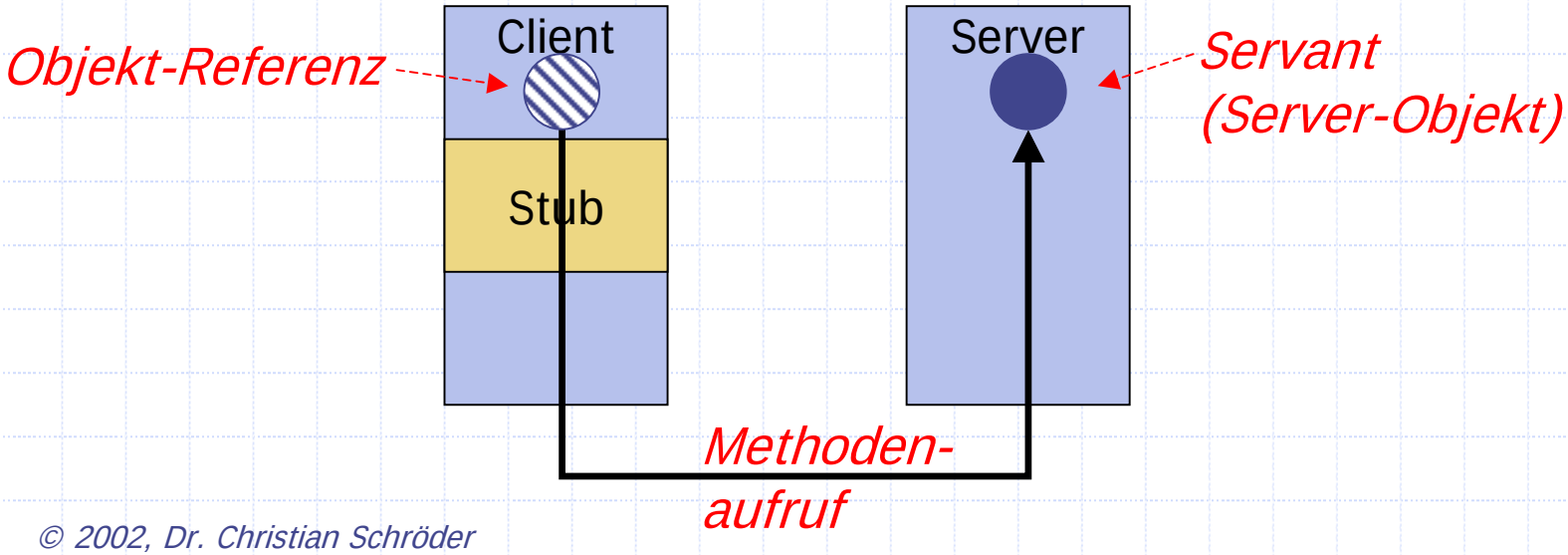
3.1 Prinzip der Objektkommunikation mit CORBA

- Um einen Methodenaufruf machen zu können, benötigt ein (Client-) Objekt die Objekt-Referenz zu einem anderen (Server-) Objekt (Servant).



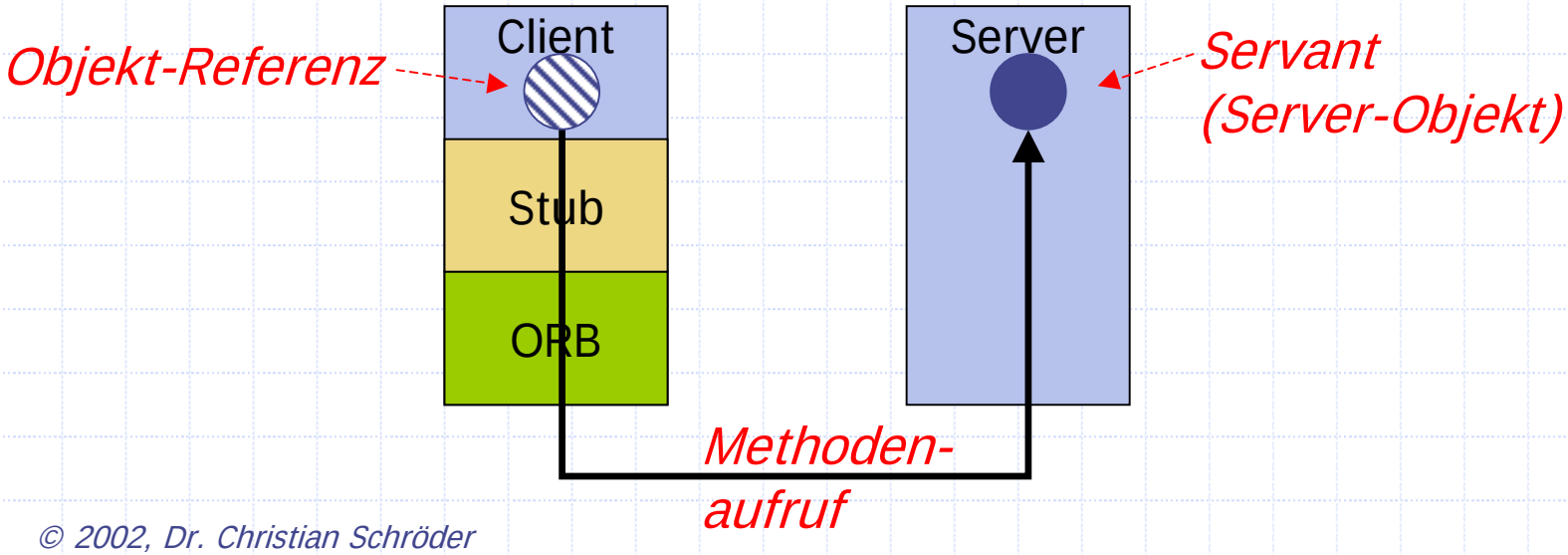
3. Wie funktioniert CORBA?

- Ist das Server-Objekt *nicht-lokal*, so zeigt die Objekt-Referenz auf eine Stub-Funktion.



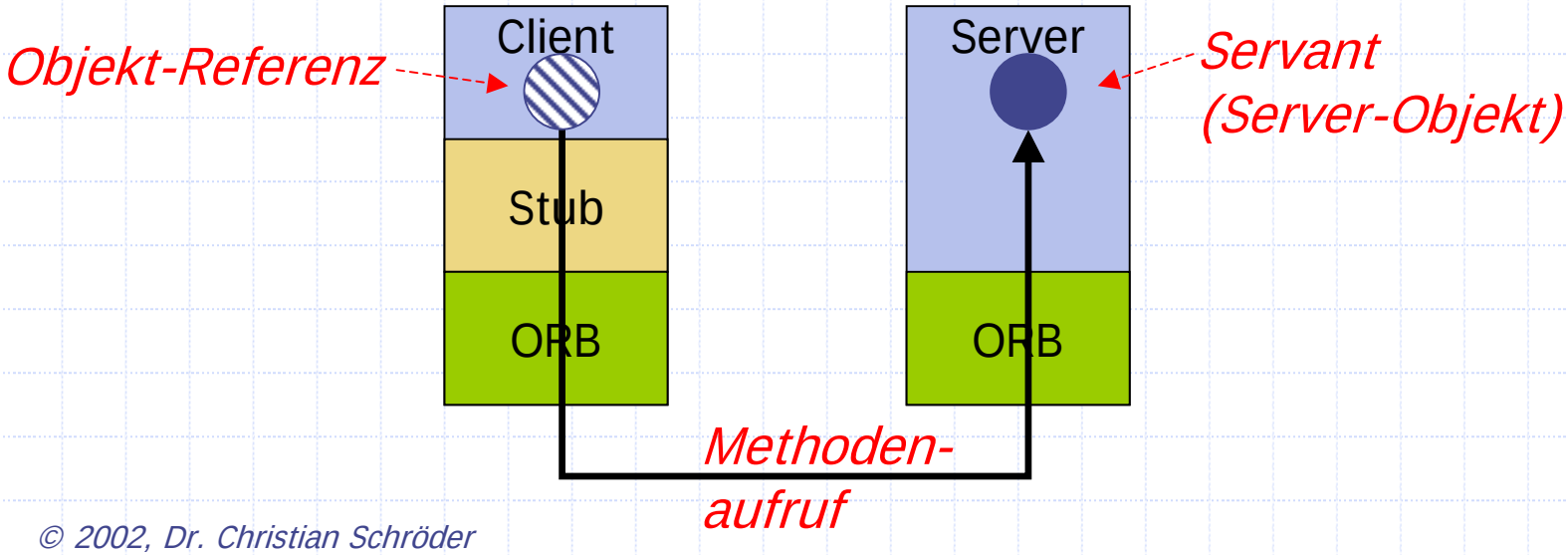
3. Wie funktioniert CORBA?

- Diese Stub-Funktion nutzt den Object Request Broker (ORB), um die Anfrage an das entfernte Server-Objekt weiterzuleiten.
- Der Stub Code verwendet den (lokalen) ORB, um denjenigen Rechner zu identifizieren, auf dem das Server-Objekt läuft.



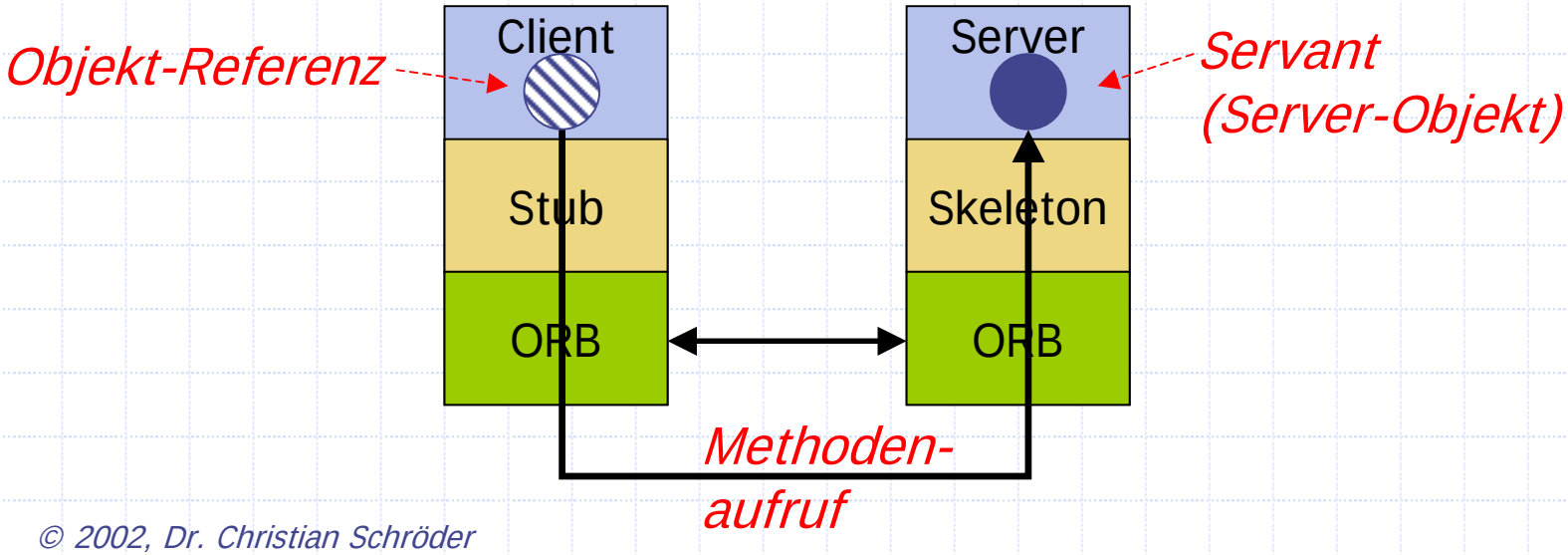
3. Wie funktioniert CORBA?

- Der stub Code fordert vom lokalen ORB eine Verbindung zum ORB der entfernten Maschine des Server-Objekts an.



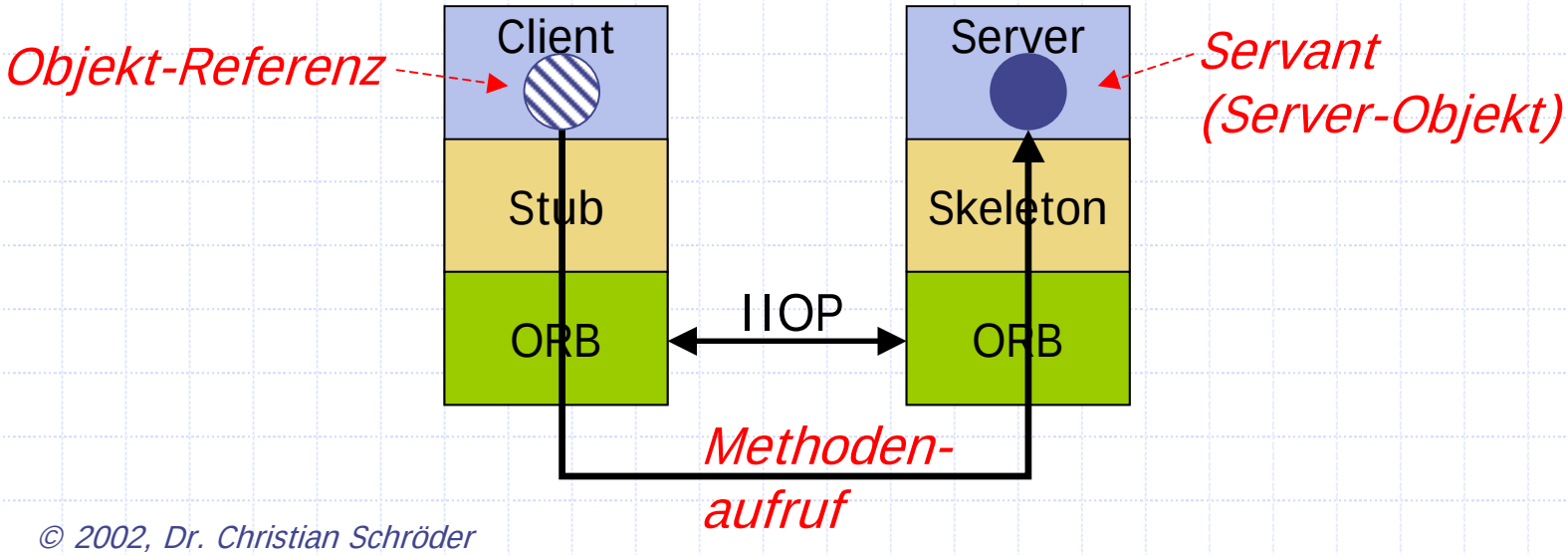
3. Wie funktioniert CORBA?

- Sobald die Verbindung hergestellt ist, sendet der stub Code die Objekt-Referenz und die Parameter zum Skeleton Code auf dem Server, der die Verknüpfung mit der Server-Objekt-Implementierung darstellt.
- Der Skeleton Code transformiert den Aufruf und die Parameter in das erforderliche implementierungsspezifische Format und ruft das Server-Objekt auf.
- Alle Ergebnisse oder Exceptions werden auf dem gleichen Pfad zurückgegeben.



3. Wie funktioniert CORBA?

- Der Client hat *keine* Informationen darüber
  - wo das Server-Objekt läuft,
  - wie das Server-Objekt implementiert ist und
  - welcher ORB verwendet wurde, um das Server-Objekt zu erreichen.
- Verschiedene ORBs kommunizieren hierbei über das von der OMG standardisierte Internet-InterORB Protokoll (IIOP)
- Mit Hilfe der Basisdienste CORBA Services (COS) werden Integration und Interoperation von verteilten Objekten gehandhabt.



3. Wie funktioniert CORBA?

3.2 Implementierung von CORBA Objekten

- Ein Client kann nur Methoden aufrufen, die im Interface des CORBA Objekts definiert sind.
- Ein CORBA Interface
  - definiert den Objekt-Typ und
  - spezifiziert eine Menge von Methoden und Parametern, sowie die
  - Exception-Typen, die diese Methoden zurückgeben können.
- Dieses Interface wird unabhängig von der Implementierungssprache mit Hilfe der von der OMG standardisierten Interface Definition Language IDL spezifiziert.

3. Wie funktioniert CORBA?

3.3 CORBA IDL Compiler

- Ein IDL Compiler übersetzt diese CORBA Objekt Definitionen in spezifische Programmiersprachen gemäß der standardisierten OMG Language Mappings
  - C, C++, Smalltalk, COBOL, Modula3, DCE, Java, ...
- Für jeden Objekt-Typen (d.h. interface) generiert der IDL Compiler die Stub und Skeleton Dateien.
  - Stub Dateien repräsentieren den Client mit Zugriff auf die IDL-definierten Methoden (in der Programmiersprache des Clients)
  - Server Skeleton Dateien binden die Server-Objekt-Implementierung an das ORB Laufzeitsystem. Der ORB verwendet die Skeleton Dateien zur Abbildung der Methoden auf die Objekt-Instanzen (Servants)

4. CORBA IDL

4.1 Die Sprache IDL

- IDL ist keine Programmiersprache!
  - sie definiert nur Interfaces, keine Implementierung
    - d.h. es können keine Objektzustände oder Algorithmen beschrieben werden.
- CORBA Interfaces sind keine Interfaces im Sinne einer OO Sprache (es gibt Attribute!)
  - aber syntaktisch ähnlich zu Java Interfaces / C++ abstract classes
- IDL separiert Interface und Implementierung
- Vorteile einer IDL
  - Plattformunabhängigkeit (z.B. NT, Unix)
  - verstärkte Modularisierung
  - erhöhte Robustheit
  - Programmiersprachen-Unabhängigkeit

4. CORBA IDL

4.2 Sprachelemente

- modules (definieren Namensräume) und interfaces (Objekttypen)
- Operationen und Attribute
- einfache und mehrfache Vererbung
- Basistypen (double, long, char, etc.)
- any type (ein dynamisch getypter Wert)
- Arrays und sequence
- Konstrukt-Typen (struct, enum, union, typedef)
- consts
- exceptions

5. CORBA@Work - Ein Anwendungsbeispiel

5.1 Technische Voraussetzungen

- CORBA Implementierung, d.h. einen ORB
  - kommerzielle, z.B. Orbix (IONA), NEO (SUN), ...
  - freeware, z.B. MICO oder EngineRoom CORBA (hier verwendet)
  - Java IDL (ab JDK 1.2) inkl. IDLtoJava Compiler

- EngineRoom CORBA
  - Entwicklungs- und Laufzeitumgebung für CORBA 2.2 Anwendungen
  - unterstützt C, C++, Java, Perl
    - IDLtoTargetLanguage Compiler ist in Java geschrieben
    - für jede Zielsprache ist eine ORB Bibliothek enthalten
  - EngineRoom verwendet Internet-Inter-ORB Protokoll (IIOP 2.0)
    - basiert auf General-Inter-ORB-Protokoll GIOP 1.1

5. CORBA@Work - Ein Anwendungsbeispiel

5.2 Aufgabe

Es soll eine Java basierte Client-Server Anwendung eines Bankensystems mit folgenden Funktionalitäten erstellt werden:

- Einrichtung eines neuen Kontos
- Geld abheben oder deponieren auf einem existierenden Konto

Hinweis: Wie dies auf der Server Seite umgesetzt wird (Verwendung einer Datenbank, dateibasiert, ...), ist nicht Sache von CORBA

5. CORBA@Work - Ein Anwendungsbeispiel

5.3 Vorgehensweise

- Erstellen einer Interface Beschreibung mit IDL
- Übersetzen der Interface Beschreibung in die Zielsprache
- Implementieren des Servers und der Server-Objekte (Servants)
- Implementieren des Clients
- Aktivieren aller Komponenten (Ausführung der Anwendung)

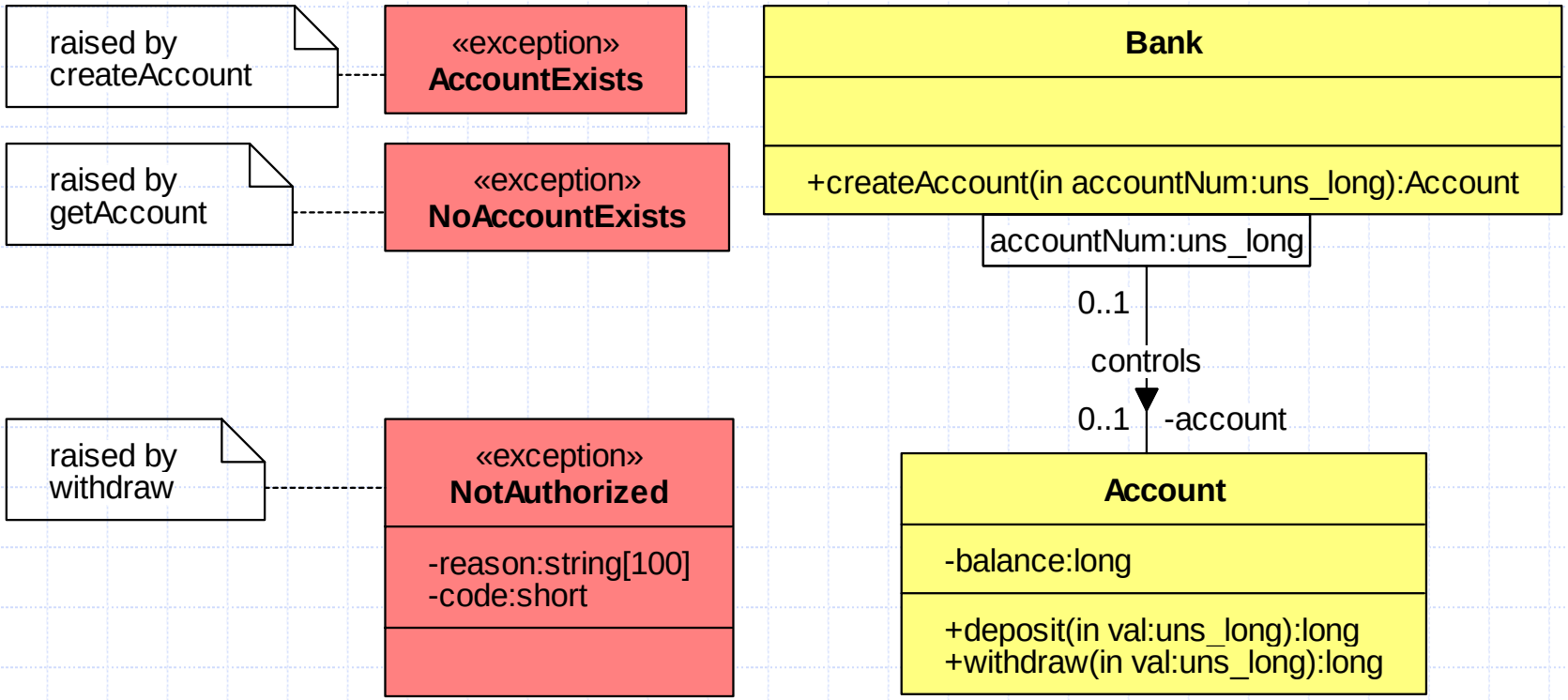
5. CORBA@Work - Ein Anwendungsbeispiel

5.4 Die Interface Definitionen

```
// banking.idl
   interface Account ;
   exception NoAccountExists { } ;
   exception AccountExists { } ;
   exception NotAuthorized { string<100> reason ; short code ; } ;
   interface Bank {
       Account getAccount( in unsigned long accountNum )
             raises( NoAccountExists ) ;
       Account createAccount( in unsigned long accountNum )
             raises( AccountExists ) ;
   } ;
   interface Account {
          readonly attribute long balance ;
          long deposit( in unsigned long val ) ;
          long withdraw( in unsigned long val ) raises( NotAuthorized ) ;
   } ;
```

5. CORBA@Work - Ein Anwendungsbeispiel

5.5 Alternative: Erstellen einer Interface-Beschreibung mit UML (anschließend automatische Code Generierung in IDL)



5. CORBA@Work - Ein Anwendungsbeispiel

5.6 Übersetzen der Interface Beschreibung in die Zielsprache

- Aufruf des IDL Compilers mit entsprechenden Parametern
  - z.B. `idlc -java banking.idl`
- erzeugt eine Anzahl von Dateien, die die Infrastruktur für die Netzwerkkommunikation enthalten
- Für jedes Interface IF:
  - IF.java: Java Interface Klasse IF.
  - IFHolder.java: Klasse für die Abbildung der Übergabemechanismen, da Java Pointer wie C oder C++ kennt (hält die Werte von Variablen).
  - IFHelper.java: Klasse, die Instanzen vom Typ IF liest oder schreibt.
  - _IFStub.java: Stub Klasse für die Client
  - _IFImplBase.java: Server-seitige Klasse, die um die Objekt-Implementierung des Servants erweitert werden muss.
- Für jeden Typ T (und Exceptions):
  - T.java, THolder.java, THelper.java

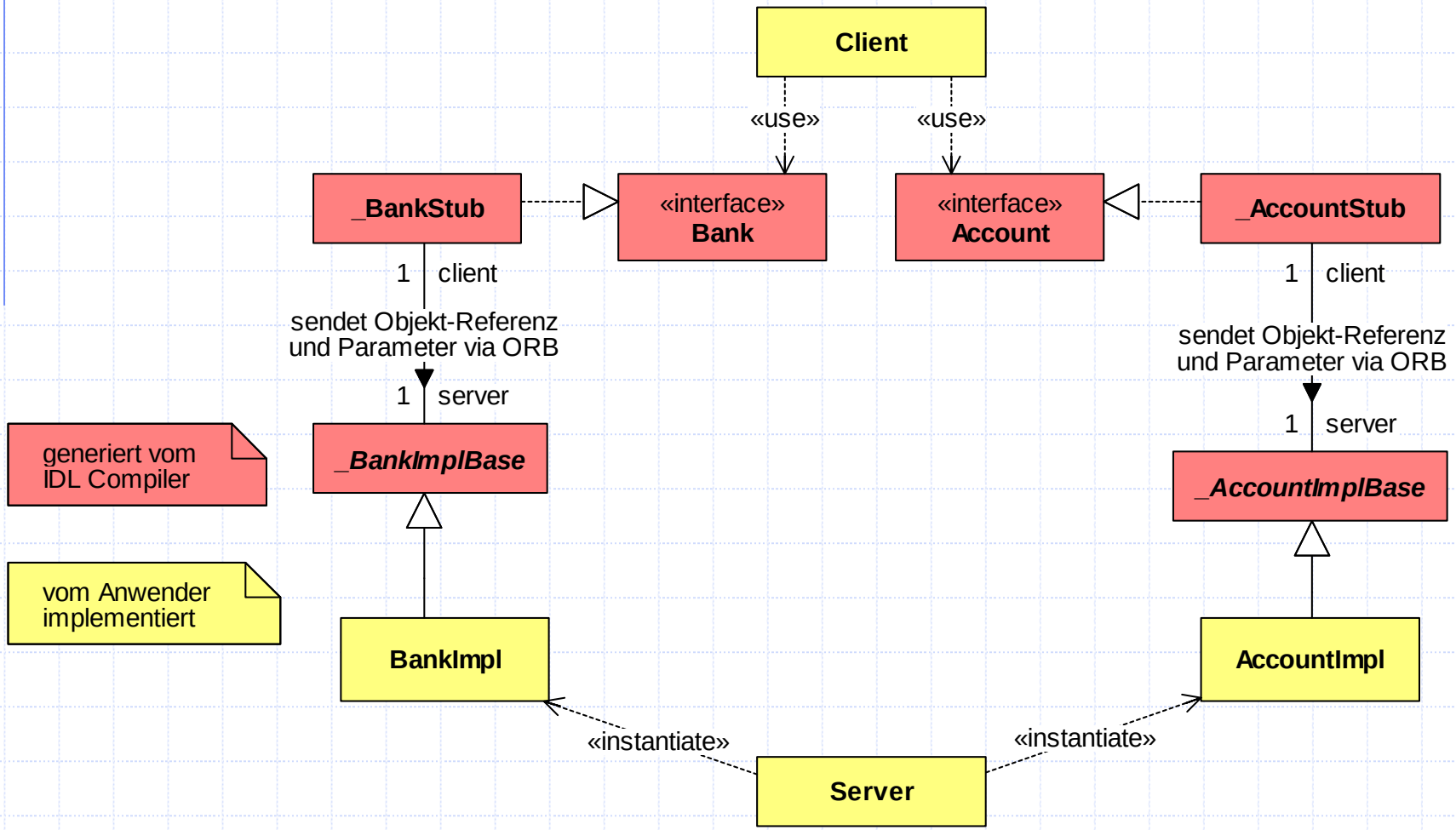
5. CORBA@Work - Ein Anwendungsbeispiel

5.7 Erzeugte Dateien

Dateiname	Größe	Typ
_AccountImplBase.java	5 KB	Java Source File
_AccountStub.java	4 KB	Java Source File
_BankImplBase.java	5 KB	Java Source File
_BankStub.java	4 KB	Java Source File
Account.java	1 KB	Java Source File
AccountExists.java	1 KB	Java Source File
AccountExistsHelper.java	2 KB	Java Source File
AccountExistsHolder.java	1 KB	Java Source File
AccountHelper.java	3 KB	Java Source File
AccountHolder.java	1 KB	Java Source File
Bank.java	1 KB	Java Source File
BankHelper.java	3 KB	Java Source File
BankHolder.java	1 KB	Java Source File
NoAccountExists.java	1 KB	Java Source File
NoAccountExistsHelper.java	2 KB	Java Source File
NoAccountExistsHolder.java	1 KB	Java Source File
NotAuthorized.java	1 KB	Java Source File
NotAuthorizedHelper.java	3 KB	Java Source File
NotAuthorizedHolder.java	1 KB	Java Source File

5. CORBA@Work - Ein Anwendungsbeispiel

5.8 Abhängigkeiten auf Client und Server Seite



5. CORBA@Work - Ein Anwendungsbeispiel

5.9 Serverprogrammierung

5.9.1 Implementierung der Server Objekte (BankImpl, AccountImpl)

```
interface Bank {
        Account getAccount( in unsigned long accountNum )
                raises( NoAccountExists ) ;
        ...
   } ;
```

wird generiert zu

```
abstract public class _BankImplBase extends
        org.omg.CORBA.DynamicImplementation implements Bank {
 ...
}
```

```
// Implementierung für Bank
public class BankImpl extends _BankImplBase {
...
 public Account getAccount( int accountNum ) throws NoAccountExists { ... }
}
```

5. CORBA@Work - Ein Anwendungsbeispiel

5.9.2 Erzeugung der Server Objekte im Server

- Server Objekte werden durch Aufruf von ‚new' erzeugt

```
BankImpl bank = new BankImpl() ;
```

5.9.3 Initialisierung des ORB

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, null ) ;
```

5.9.4 Bekannt machen der Objekt-Referenz

- Die Objekt-Referenz wird in eine IOR* (Interoperable Object Reference) umgewandelt und kann z.B. in eine Datei geschrieben werden

```
String ior;
...
BankImpl bank = new BankImpl() ;
orb.connect( bank ) ;
...
ior = orb.object_to_string( bank ) ;
```

**Die vergebenen IOR sind weltweit eindeutig*

5. CORBA@Work - Ein Anwendungsbeispiel

5.9.5 Client Anfragen bearbeiten

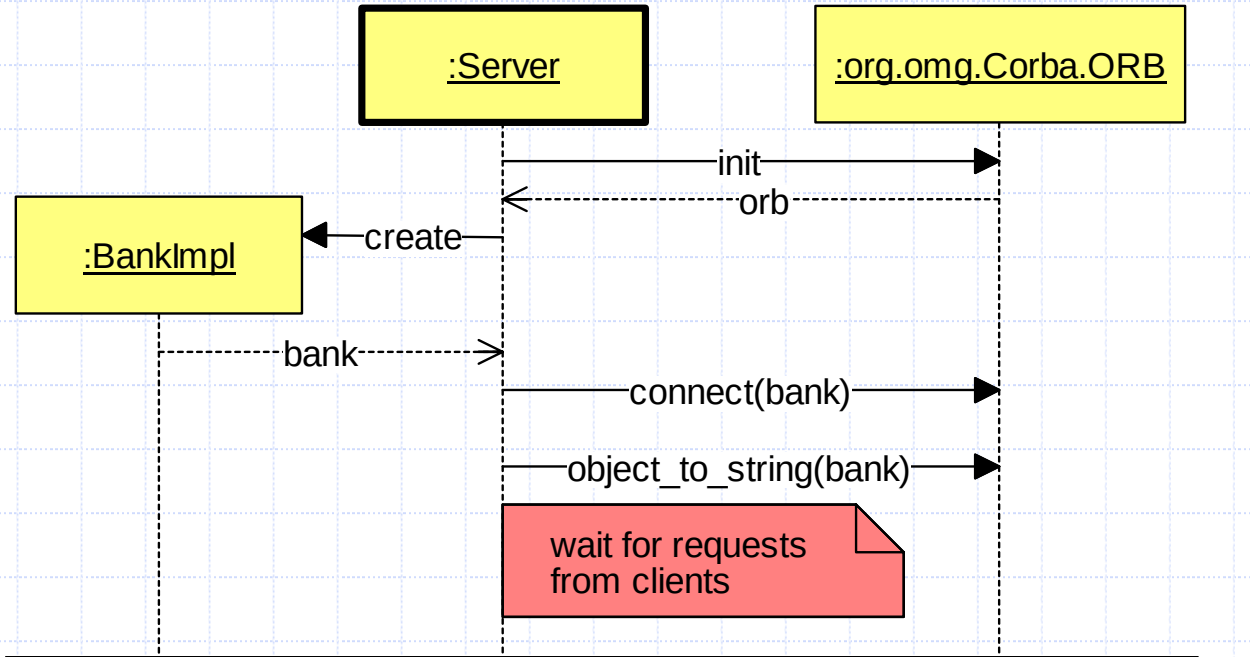
- Nach der Initialisierung und Anmeldung der Objekte kann der folgende Code verwendet werden, um Client Anfragen bearbeiten zu können.

```
// wait for invocations from clients

        java.lang.Object sync = new java.lang.Object() ;

        synchronized ( sync )
        {
            sync.wait() ;
        }
```

- *sync.wait* kehrt nicht mehr zum Aufrufer zurück.

5. CORBA@Work - Ein Anwendungsbeispiel

5.9.6 Übersicht: Serverprogrammierung (UML Sequenzdiagramm)



1. Initialisierung des ORB, so dass Aufrufe ausgeführt werden können.
2. Bank-Objekt erzeugen
3. Beim ORB anmelden
4. Persistente(!) Objekt-Referenz IOR (Interoperable Object Reference) in String umwandeln und speichern (z.B. in eine Datei schreiben) Mit Hilfe dieser Referenz kann jedes Objekt auf dieses Objekt zugreifen.

5. CORBA@Work - Ein Anwendungsbeispiel

5.10 Client Programmierung

5.10.1 Initialisierung des ORB

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, null ) ;
```

5.10.2 Anforderung einer Objekt-Referenz

- Transformation einer IOR in eine Objekt-Referenz

```
org.omg.CORBA.ORB orb;
String ior;
org.omg.CORBA.Object obj = orb.string_to_object( ior ) ;
```

5.10.3 Typ-Verengung der Objekt-Referenz (Narrowing, down-casting)

- da das Objekt noch vom Typ org.omg.CORBA.Object ist, muss es zunächst auf den Interface-Typ eingeschränkt werden

```
...
org.omg.CORBA.Object obj = orb.string_to_object( ior ) ;
bank = BankHelper.narrow( obj ) ;
```

5. CORBA@Work - Ein Anwendungsbeispiel

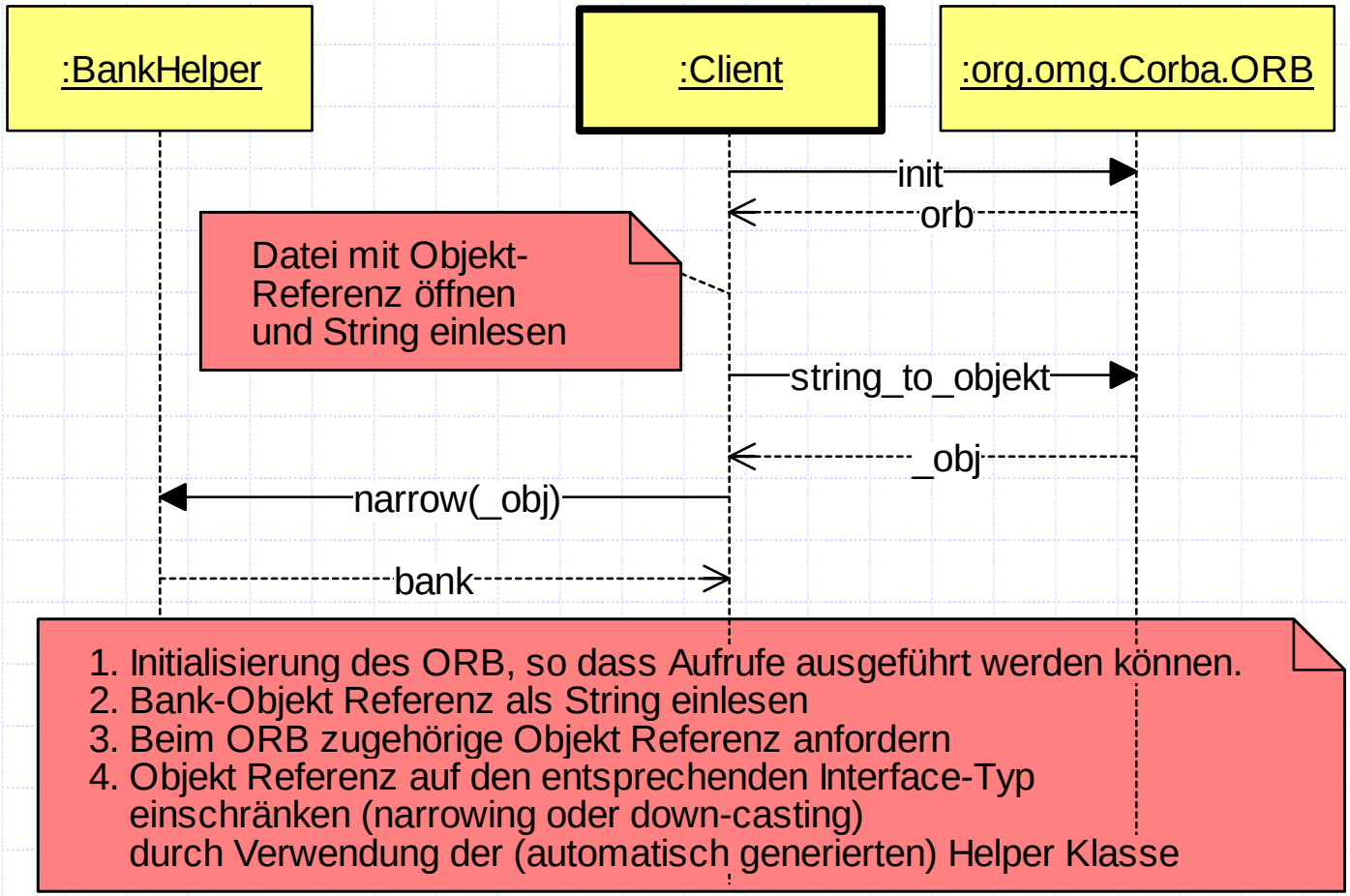
5.10.4 Kommunikation mit den Server-Objekten

- Die entfernten Methodenaufrufe unterscheiden sich nun syntaktisch nicht mehr von lokalen Aufrufen

```
private void deposit( int val )
    {
      System.out.println( "Deposit " + val +"- new balance =" + account.deposit( val ) ) ;
     }
 private void withdraw( int val )
    {
       int bal = account.withdraw( val ) ;
       System.out.println( " - new balance = " + bal ) ;
     }
 public void test()
    {
        int accNum = 42 ;
        account = bank.createAccount( accNum ) ;
        deposit( 100 ) ;
        withdraw( 154 ) ;
    }
```

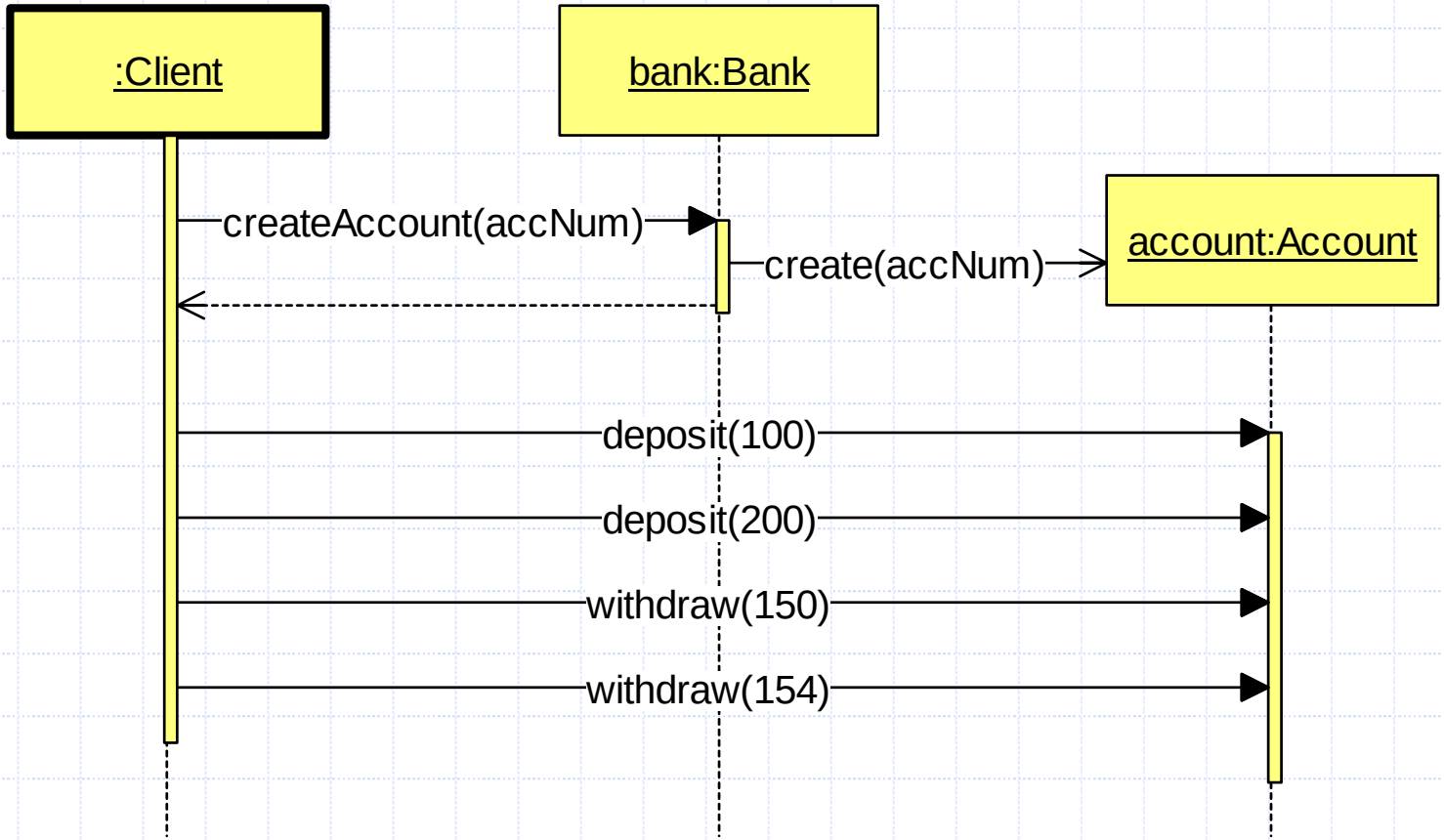
5. CORBA@Work - Ein Anwendungsbeispiel

5.10.5 Übersicht: Client Programmierung (UML Sequenzdiagramm)



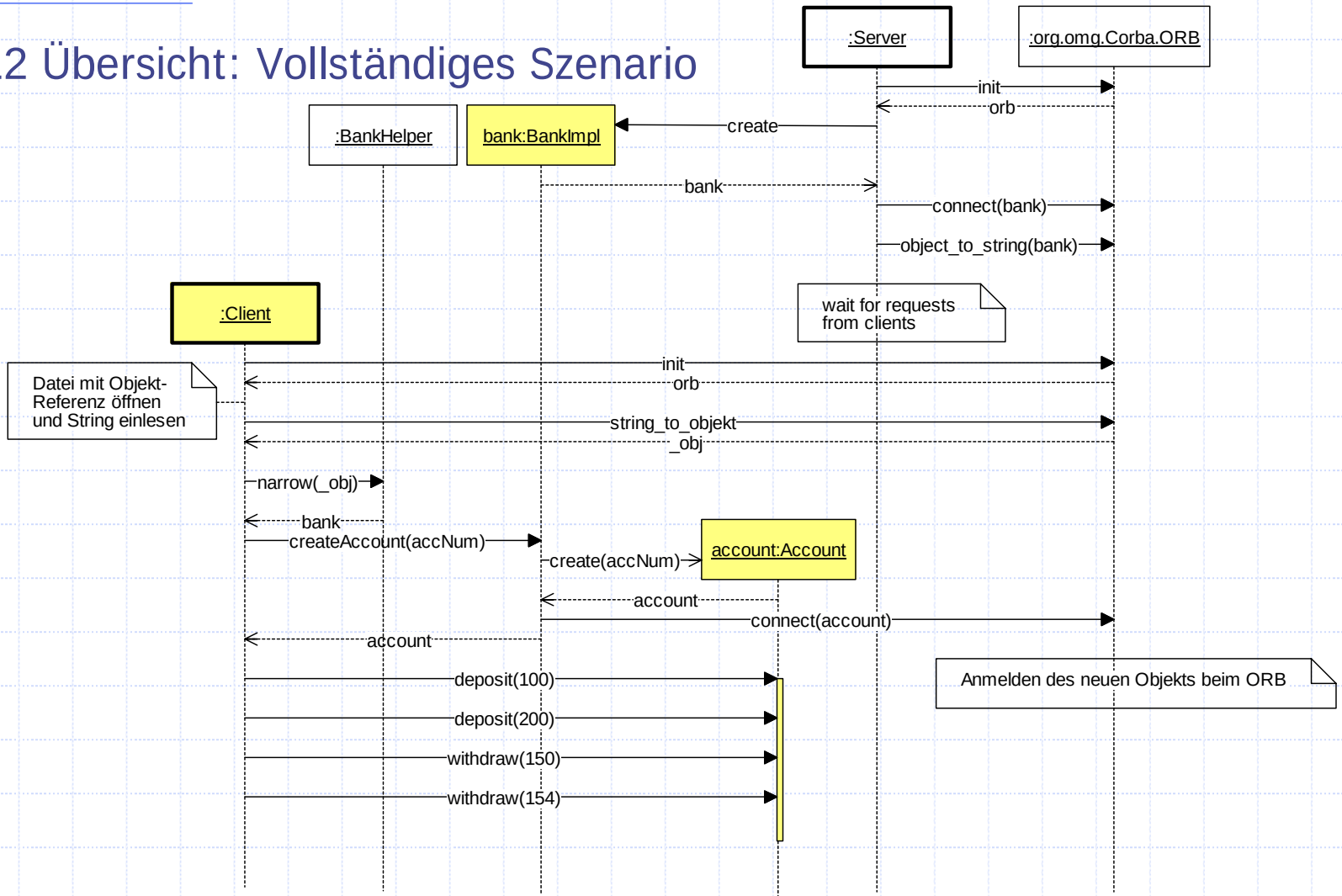
5. CORBA@Work - Ein Anwendungsbeispiel

5.11 Ausführen der Anwendung: Ein Testfall-Szenario



5. CORBA@Work - Ein Anwendungsbeispiel

5.12 Übersicht: Vollständiges Szenario



6. CORBA@Work - Ausblick

6.1 Wie sich Objekte finden können

- Ein Client erhält die Objekt-Referenz
  - als Parameter
  - als Rückgabewert einer Anfrage
  - über einen String (string_to_object)
  - über einen Name Service („Telefonbuch“)
    - Objekt-Referenzen werden über Namen gespeichert
    - der Client kann über lookup & resolve die Objekt-Referenz erfragen
  - über einen Trader
    - Auffinden von Objekten über deren Eigenschaften („Gelbe Seiten“)

6.2 Ausnahmebehandlung: Exceptions

- CORBA exceptions sind sehr ähnlich zu Java exceptions
- CORBA unterstützt System und User Exceptions

6. CORBA@Work - Ausblick

6.3 Möglichkeiten der ORB Realisierung

- Client- und Implementierungs-resident
  - ORB ist implementiert als Bibliothek
  - ORB ist resident in den Klienten und in der Objektimplementierungen
- Library-resident (single-process resident)
  - ORB und Objektimplementierungen sind als Bibliothek implementiert und Klienten-resident.
- Server-based
  - ORB ist als Server implementiert (separater Prozess)
  - ORB dient als Vermittler (broker) zwischen Klienten-Requests und Implementierungs-Prozessen (server).
- System-based
  - ORB ist Teil des Betriebssystems.

6. CORBA@Work - Ausblick

6.4 Static Invocation Interface (SII)

- alle Operationen sind zur Compilezeit festgelegt
- sie sind Klient und Server durch Proxies bekannt (Stubs and Skeletons)
- Stubs verpacken Operationsaufrufe in Request-Nachrichten (marshalling)
- Skeletons entpacken die Nachrichten, rufen die Implementierung und rückübertragen die Antwort
- SII ist einfach, typsicher und effizient

6.5 Dynamic Invocation Interface (DII)

- DII erlaubt Klienten dynamisch:
- Objekte zu finden;
- Objekt-Interfaces zu finden;
- Requests (Methodenaufrufe) zu erzeugen;
- Methodenaufrufe abzusetzen;
- Ergebnisse zu empfangen.
- DII ist flexibler als SII aber komplizierter, weniger typsicher, ineffizienter

7. Zusammenfassung

- CORBA ist eine Spezifikation (Standard) von der Open Management Group für eine einheitliche (standardisierte) Interaktion zwischen Objekten bezüglich der SW-Systemsicht
- CORBA ist unabhängig bezüglich
  - Programmiersprache
  - Betriebssystem
  - Netzwerkumgebung
- In CORBA ist jede Komponente ein Objekt
  - mit einem objekt-orientierten Interface (definiert in IDL)
- CORBA realisiert Lokationstransparenz
  - Dienst-Nutzung ist unabhängig von Dienst-Lokation
- Trennung von Interface und Implementierung
  - Klienten hängen von Interfaces ab, nicht von (Server-) Implementierungen
- Eine Implementierung von CORBA ist ein ORB

7. Zusammenfassung

Die CORBA Infrastruktur

- bietet einen lokalen, wohldefinierten Endpunkt für alle Objektaufrufe eines Klienten
- leitet den Aufruf an die Ziel-Objektimplementierung, lokal wie entfernt
- versteht IDL
- verwaltet ein Repository verfügbarer Objektinterfaces
- verwaltet ein Repository verfügbarer Objektimplementierungen
- föderiert diese Information im Gesamtsystem
- realisiert als ein Netz verbundener ORBs

8. Literatur

- Jens-Peter Redlich, „CORBA 2.0: Praktische Einführung für C++ und Java“, Addison-Wesley, 1996
- www.cetus-links.org

Ergänzung

- Beispiel für eine URL IOR