2 Literatur, URLs

HeVi99. Michi Henning, Steve Vinosk: *Advanced CORBA Programming with C++*. Add
1999.

BrVD01. Gerald Brose, Andreas Vogel, Keith Duddy: Java Programming with CORBA.
Wiley, 2001.

OMG99. Object Management Group, OMG: *The Common Object Request Broker: Arc
Specification.* Rev. 2.3.1, OMG Doc. formal/99-10-07, Oct. 1999.

Pope98. A. Pope: *The CORBA Reference Guide*. Addison-Wesley. 1998.

Linn98. C. Linnhoff-Popien: *CORBA, Kommunikation und Management*. Springer, 199

TaBu01. Zahir Taki, Omran Bukhres: *Fundamentals of Distributed Object Systems*. Wi

Daneben vor allem online-Informationen

- OMG
  `http://www.omg.org/gettingstarted/`
  offizielle Seiten der Object Management Group

5.2 CORBA-Überblick

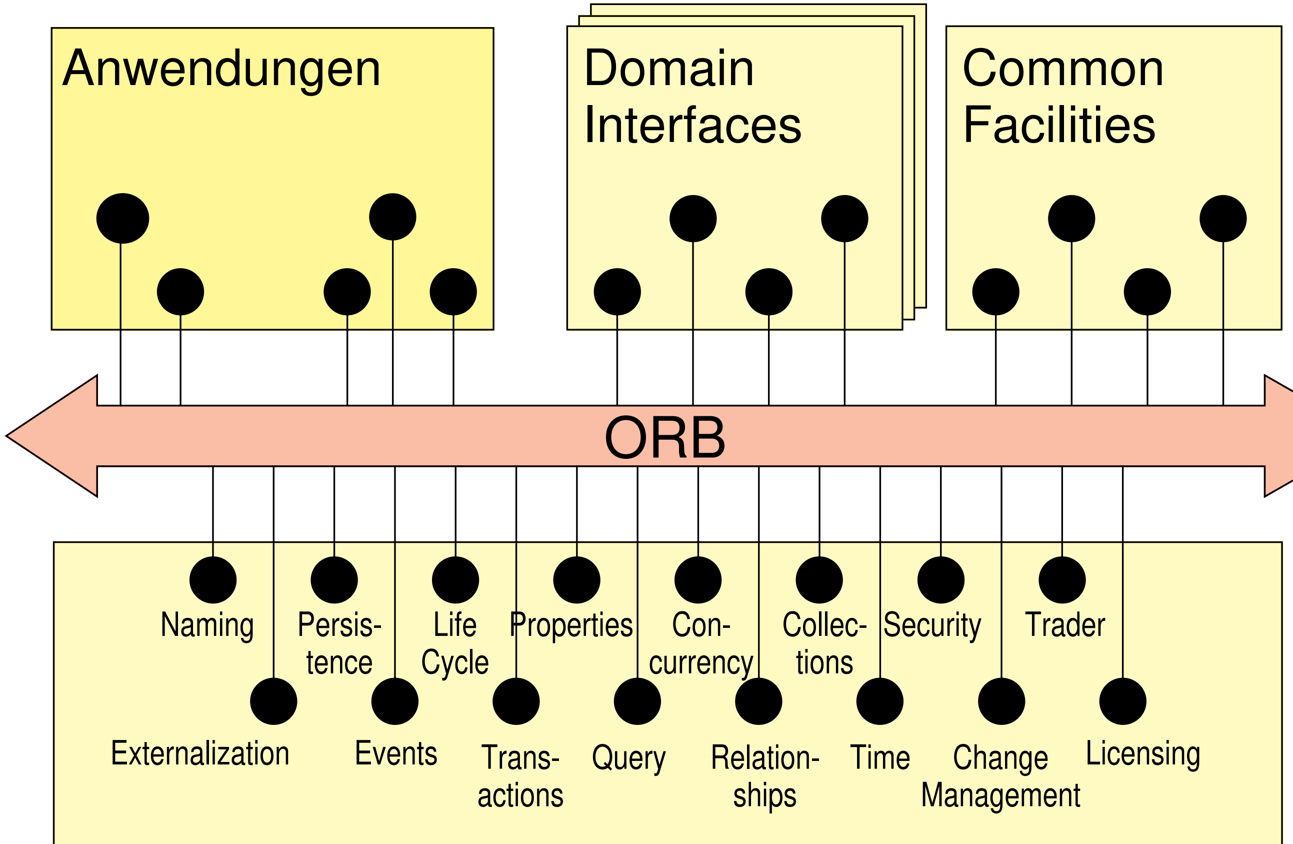
1 Standard

- CORBA = Common Object Request Broker Architecture
  - plattformunabhängige Middleware-Architektur für verteilte Objek
- OMG = Object Management Group
  - Standardisierungsorganisation mit etwa 1000 Mitgliedern
  - Teilbereiche der Standardisierung
    - CORBA
    - UML
    - XMI
    - ...

3 Entwurfsziele

- CORBA ist ein Standard
  - Standard-Dokumente
  - Definition von "CORBA compliance"
    - CORBA schreibt keine Implementierung vor sondern Fur
    - CORBA fordert Interoperabilität verschiedener Implemen
  - Hersteller realisieren Implementierungen des Standards (z. B., VisiBroker, Orbix, Orbacus, MICO, etc.)
- CORBA ist eine Middleware zur Abstraktion von
  - Hardware,
  - Betriebssystem und
  - Programmiersprache

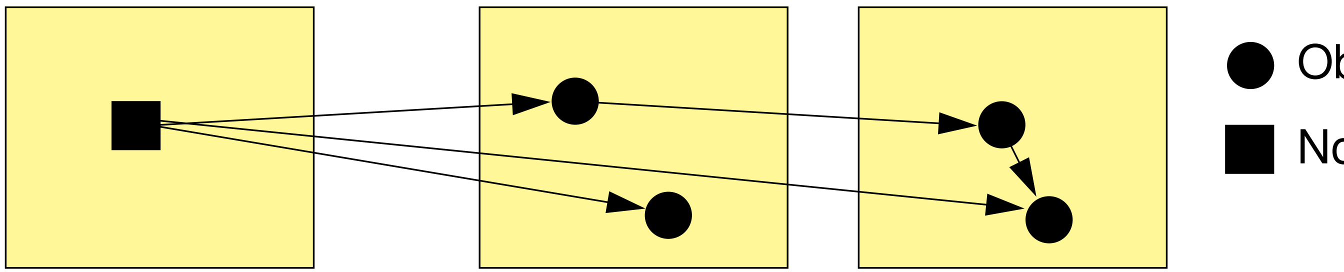
4 OMA – Object Management Architecture



5.3 CORBA-Anwendungsobjekte

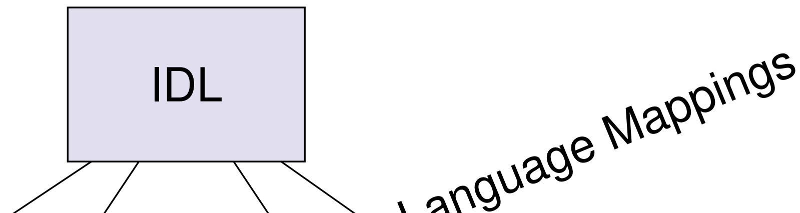
1 Verteilte CORBA-Objekte

- haben Identität, Zustand, Methoden
- bilden eine Anwendung
- Kommunikation zwischen den Objekten über ORB
- CORBA-Objekte können aufgerufen werden (realisieren Server-
- CORBA-Objekte können als Aufrufer (Client) agieren



2 Interface Definition Language (IDL)

- Sprache zur Beschreibung von Objekt-Schnittstellen
  - unabhängig von der Implementierungssprache des Objekts
  - Sprachabbildung (Language-Mapping) definiert, wie IDL-Konstr
    Konzepte einer bestimmten Programmiersprache abgebildet we
  - Language-Mapping ist Teil des CORBA standards
  - Language mappings sind festgelegt für:
    - Ada, C, C++, COBOL, Java, Lisp, PL/I, Python, Smalltalk
    - weitere inoffizielle Language-Mappings existieren,z.B. fü



2 Interface-Definition-Language (3)

- Beispiel: Kontoschnittstelle

```
exception WrongAccountNumber { string errorMsg;

interface Account
{
    readonly attribute double balance;

    double withdraw( in double amount )
        raises WrongAccountNumber;
    double deposit( in double amount );
        raises WrongAccountNumber;
};
```

- verteilte Objekte können **Account**-Schnittstelle implementieren

3.2Interface-Definition-Language (5)

- Umsetzung von IDL in Sprachkonstrukte (z.B. Java)

IDL

```
module MyModule
{
    interface MyInterface
    {
        attribute long lines;
        void printLine( in string toPrint );
    };
};
```

```
na
```

Java

```
package MyModule;

public interface MyInterface extends ... {
    public int lines();
    public void lines( int lines );
    public void printLine(
        java.lang.String toPrint );
```

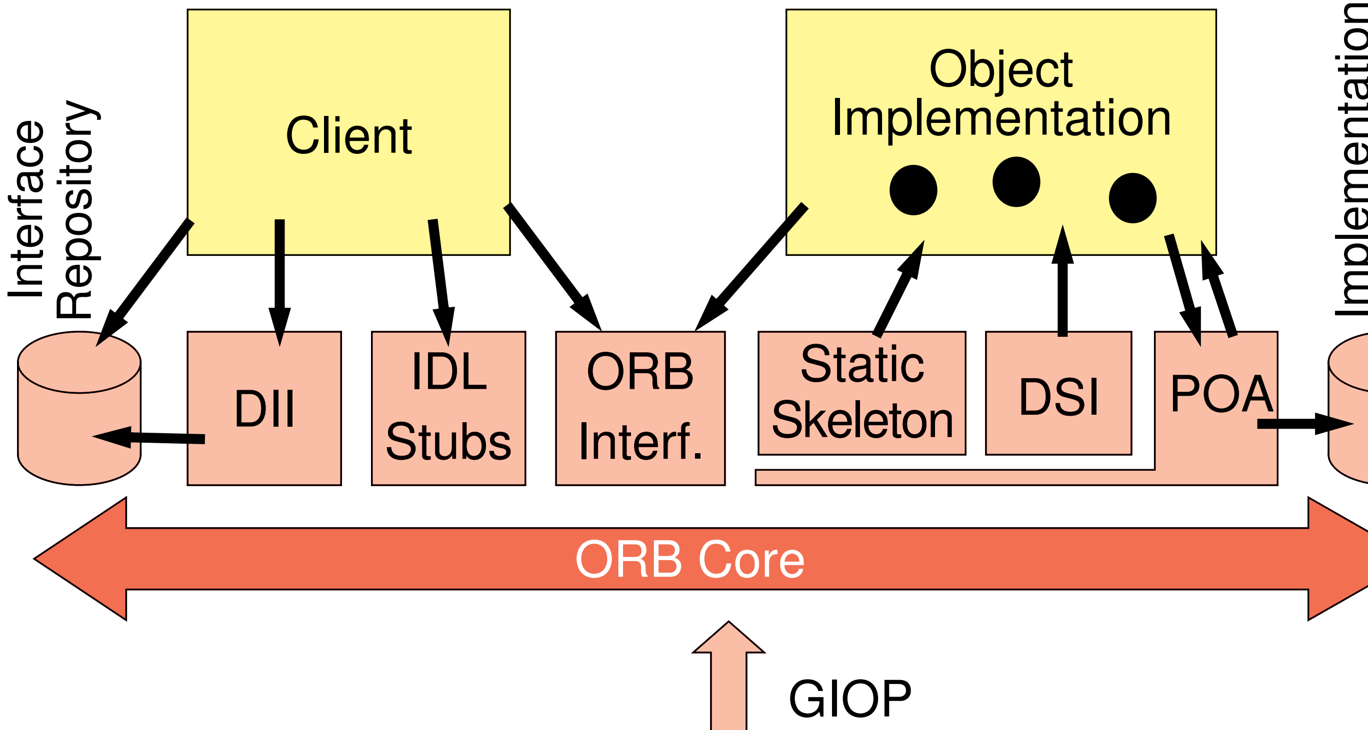
3 Objekte Erzeugen und Binden

- Beschreibung der Objektschnittstelle in IDL
- Programmierung des Server-Objekts in der Implementierungssp
- Registrierung des Objekts am ORB (bzw. dem Object Adaptor)
  - der ORB erzeugt eine "Interoperable Object Reference" (

- Binden des Clients an das Server-Objekt
  - Referenz auf Server-Objekt besorgen
    - Ergebnis einer Namensdienst-Anfrage
    - Rückgabewert eines Methodenaufrufs
    - von "ausserhalb" des Systsems: Benutzer kennt Referen
      (IORs können in Strings konvertiert werden und umgekel
  - Erzeugung des Client-Stub

1 Architektur

- Zentrale Komponenten eines ORB

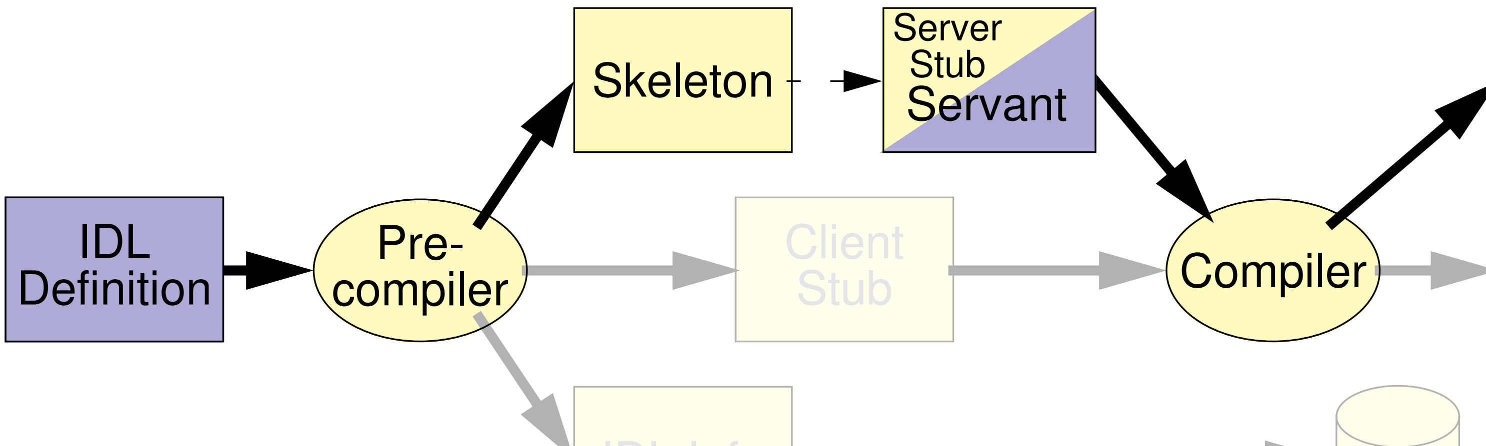


3 Object Adaptor - Vorschau

- Lokaler Repräsentant des ORB-Dienstes, direkter Ansprechpartner eines CORBA Objekts
  - Generiert CORBA-Objektreferenzen (für neue Objekte)
  - Bildet Objektreferenzen auf Implementierungen ab
  - Bearbeitet ankommende Methodenaufrufe
- CORBA definiert den Portable Object Adaptor
  - Basis-Funktionalität, muss unterstützt werden

4 Objekterzeugung

- Beschreibung der IDL-Schnittstelle
- Implementierung des Servant
  - ➤ Auswahl einer Programmiersprache
- Stub/Skeleton-Erzeugung



4 Objekterzeugung (3)

- Herausgabe der Objektreferenz
  - **3**. Servant wird mit *Object Reference* versehen (dadurch wird er zum CORBA-Objekt)
    - ➤ Aufruf von **`servant_to_reference`** am POA
      - liefert Objektreferenz auf CORBA-Objekt (nicht auf S
      - Realisierung der Objektreferenz ist sprachabhängig
      - z. B. in Java: Stellvertreterobjekt (lokaler Stellvertret optimiert)
  - **4**. *Object Reference* kann herausgegeben werden
    - ➤ über Name Service verfügbar gemacht werden
    - ➤ als Ergebnis eines Aufrufs an einen Client übergeben we

5 Objektnutzung (2)

- Empfang eines Parameters mit Objekttyp
  - automatische Erzeugung einer neuen Objektreferenz aus dem S
    - realisiert auch entfernte Objektreferenz
    - Objekttyp ist zur Compile-Zeit bekannt: Stub-Code kann erzeugt w
  - Aufruf von Operationen gemäß Sprachanbindung
    - z. B. Java: Aufruf von Methoden am Stellvertreterobjekt

- Parameterübergabe
  - Objekttypen: Übergabe der Objektreferenz
  - Basistypen: Übergabe des abgebildeten Sprach-Datentyps
  - komplexere Typen: Übergabe sprachabhängig

5 Objektnutzung (4)

- Typumwandlung
  - Objektreferenz kann einen Typ aus der Vererbungshierarchie d
    Schnittstellen besitzen
    - ➤ wahrer Objekttyp kann spezifischer in der Vererbungshie
  - Erzeugung einer anderen Objektreferenz mit anderem Typ
    - ➤ **`narrow`**-Operation: sprachabhängige Umsetzung
    - ➤ z.B. Java: Helper-Klasse pro Objekttyp
      – z. B. **`AccountHelper`**
      – Aufruf der statischen Methode **`narrow`** mit Objektrefer
      Parameter erzeugt neue Objektreferenz vom Typ **`Acc`**
    - ➤ Umwandlung in weniger spezifischen Typ (einen Obertyp
      immer möglich
    - ➤ Umwandlung in spezifischeren Typ (einen Untertyp):

6 Initialisierung der Anwendung

- Initiale entfernte Objektreferenzen
  - Namensdienst für Registrierung und Anfragen von Objektrefere
  - Woher bekommt man die Referenz auf den Namensdienst?
    - ORB-Interface: Anfrage nach initialen Referenzen
    - z. B. `orb.resolve_initial_reference( "NamingService" )`
    - Vorkonfiguration durch Systembetreiber
  - Alternative: Übergabe der Referenzen außerhalb des Systems
    - ORB-Interface: Umwandlung in einen String:
      `orb.object_to_string( objref );`
    - Rückumwandlung:
      `orb.string_to_object( str );`

1 Portable-Object-Adaptor (POA): Ziele

- Portabilität von Objektimplementierungen zwischen verschiede
- Trennung von Objekt (CORBA-Objekt mit seiner Identität) und Objektimplementierung
  - ➤ eine Objektimplementierung kann mehrere CORBA-Objekte r
  - ➤ Objektimplementierung kann bei Bedarf dynamisch aktiviert w
  - ➤ persistente CORBA-Objekte (Objekte, die Laufzeit eines Serv überdauern)
  - ➥ eine Object Reference beim Client steht für ein bestimmte Objekt — nicht unbedingt für einen bestimmten Servant!
- Mehrere POA-Instanzen (mit unterschiedlichen Strategien) in

3 POA-Erzeugung

- Root-POA existiert in jedem ORB
  - voreingestellte Konfiguration
  - kann zur Aktivierung von Objekten verwandt werden
- Referenz auf Root-POA
  - Anfrage am ORB mit: **`orb.get_initial_reference( "RootPOA" )`**
- Weitere POAs mit unterschiedlichen Konfigurationen erzeugb
  - Erzeugung am RootPOA bzw. an untergeordnetem POA (Vater
  - neuer POA bekommt eigenen Namen (String)
  - baumförmige, hierarchische POA-Struktur mit Wurzel beim Roo
  - POA-Instanzen teilen sich Kommunikationskanal

5 Deaktivierung von Objekten, POA und Server

- Ressourcen-Problem
  - ➤ sehr viele CORBA-Objekte "in" einem Server
  - ➤ mehrere POAs in einem Server
  - ➤ Objekte oder auch POAs werden evtl. nur selten genutzt
  - ➤ Server-Prozess wird ggf. nur selten benötigt
- CORBA-Objekte können ihren Servant und ihre POA-Instanz
  - ➤ persistente Objekte
  - ◆ Servants können deaktiviert werden
  - ◆ POA kann deaktiviert werden
  - ◆ Serverprozess kann beendet werden

7 Bearbeitung von ankommenden Aufrufen

- Ankommende Aufrufe enthalten *Object Key*
  (= Time Stamp + POA Id + Servant Id)

1. Server-Prozess finden
   - ◆ ORB lokalisiert zuständigen Server-Prozess
   - ◆ falls Server deaktiviert: Aufrufe landen beim Implementation-Re
     - ➤ Implementation-Repository hält dauerhafte Information ü CORBA-Objekt, insbesondere wie dieses wieder aktivier kann
     - ➤ z. B. Programmdatei für Neustart
     - ➤ Starten eines neuen Servers und Aktivieren des Root-PC
     - ➤ Aufruf wird an neue Kommunikationsadresse weitergelei *(Location Forward)*

7 Bearbeitung von ankommenden Aufrufen (*2*)

2. POA finden
   - ORB lokalisiert zuständigen POA
   - falls POA deaktiviert: ORB ruft *Adapter Activator* bei "Vater-POA

3. Servant finden
   - ORB leitet Aufruf an zuständigen POA weiter
   - POA findet Servant entsprechend seiner *Servant Retention*- und *Processing*-Strategie
     - ggf. Servant neu instanziieren
     - sie Abschnitt POA-Policies

4. Skeleton finden

8 POA Policies (2)

- ObjectId Assignment policy
  - system- oder benutzervorgegebene Objekt-IDs
- Implicit activation policy
  - falls implizite Aktivierung von Servants und Erzeugung von COR
    Referenzen unterstützt werden soll
- Servant retention policy
  - Tabelle aktiver Objekte ordnet Servants den CORBA-Referenze
  - Alternativ: bei ankommendem Aufruf wird jedesmal ein Servant
    angeforderten CORBA-Objekt zugeordnet
- Request processing policy