# 5 Middleware und verteilte Anwendungen: CORBA

## 1 Überblick

- Motivation
- Object Management Architecture (OMA)
- Anwendungsobjekte und IDL
- Object Request Broker (ORB)
- Portable Object Adaptor (POA)
- CORBA Services


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 2 Literatur, URLs


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

**BrVD01.** Gerald Brose, Andreas Vogel, Keith Duddy: *Java Programming with CORBA*. 3rd Edition, Wiley, 2001.

**OMG99.** Object Management Group, OMG: *The Common Object Request Broker: Architecture and 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, 1998.

**TaBu01.** Zahir Taki, Omran Bukhres: *Fundamentals of Distributed Object Systems*. Wiley, 2001.

**AlKo05** Markus Aleksy, Axel Korthaus, und Martin Schader: *Implementing Distributed Systems with Java and CORBA*. Springer 2005


Daneben vor allem online-Informationen

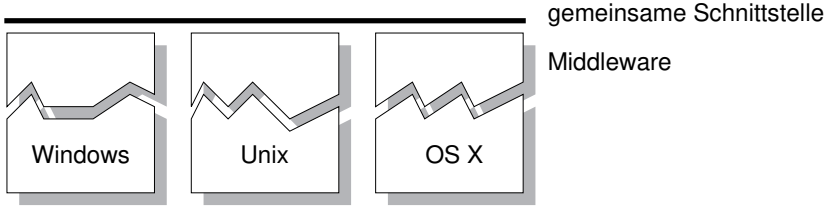
- OMG
  `http://www.omg.org/gettingstarted/`
  offizielle Seiten der Object Management Group
- Douglas C. Schmidt
  `http://www.cs.wustl.edu/~schmidt/corba.html`
  sehr gute Informationssammlung


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5.1 Middleware für verteiltes Programmieren

- Middleware – Software zwischen Betriebssystem und Anwendung




  - Bereitstellung von Diensten für verteilte Anwendungen
  - **gesucht:** Middleware für verteiltes objektorientiertes Programmieren
- CORBA – **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  Standard der OMG


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5.2 CORBA-Überblick

## 1 Standard

- CORBA = Common Object Request Broker Architecture
  - plattformunabhängige Middleware-Architektur für verteilte Objekte
- OMG = Object Management Group
  - Standardisierungsorganisation mit etwa 1000 Mitgliedern
  - Teilbereiche der Standardisierung
    - CORBA
    - MDA
    - UML
    - ...
- Formaler Standardisierungprozess
  - Standard-Dokumente
  - Definition von CORBA -Kompatibilität


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Motivation

- Verteilte objektbasierte Programmierung
  - ◆ verteilte Objekte mit definierter Schnittstelle
- Heterogenität in Verteilten Systemen
  - ◆ verschiedene Hardware-Architekturen
  - ◆ verschiedene Betriebssysteme und Betriebssystem-Architekturen
  - ◆ verschiedene Programmiersprachen
    - ➤ Transparenz der Heterogenität
- Dienste im verteilten System
  - ◆ Namensdienst
  - ◆ Zeitdienst
  - ◆ Transaktionsdienst
  - ◆ …


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Entwurfsziele

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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



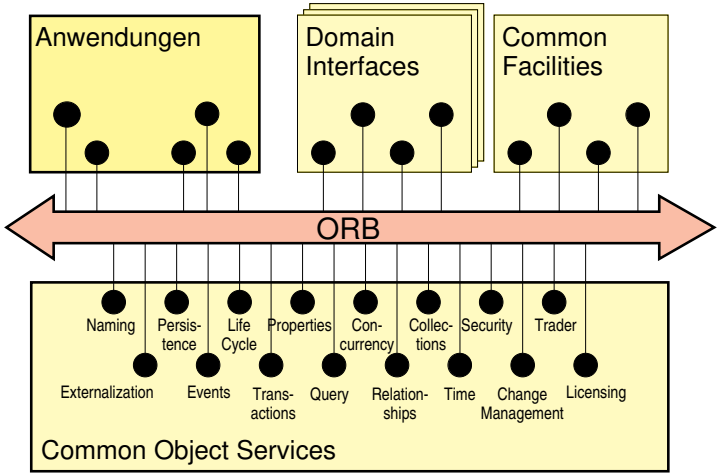
## 3 Entwurfsziele (2)

- Interoperabilität
  - ◆ Eine Anwendung soll ohne Änderungen auf verschiedenen CORBA-Implementierungen ablaufen können
  - ◆ Anwendungen auf verschiedenen CORBA-Implementierungen sollen miteinander kommunizieren können
- Einbindung von Altapplikationen ("Legacy-Anwendungen")
  - ◆ Kapselung von Altanwendungen in CORBA Objekte
- *Anfangs*vision der Business Objects / Components
  - ◆ alle Geschäftsdaten und -vorfälle sind CORBA Objekte


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 4 OMA – Object Management Architecture





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 5 CORBA-Implementierungen

- Implementierung ist Sache eines CORBA-Herstellers (Vendor)
- CORBA-Implementierung muss nicht die vollständige Spezifikation realisieren
  - ◆ ORB-Core: ORB-Funktionen, Objektkommunikation
  - ◆ Interoperabilität bei der Kommunikation mit ORBs anderer Hersteller
  - ◆ Sprachunterstützung für mindestens eine Sprache
  - ◆ keine Services oder Facilities vorgeschrieben
- **Wie** die Implementierung die Anforderungen des Standards realisiert bleibt freigestellt
  - ◆ unterschiedliche Implementierungen sind möglich
    - ➤ als Daemon
    - ➤ als Bibliothek
- Zertifizierung
  - ➤ CORBA-Kompatibilität kann zertifiziert werden
  - ➤ wird jedoch meist nicht gemacht


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 1 Verteilte CORBA-Objekte (2)

- Verteilte Objekte bilden eine Anwendung
  - ➤ Anwendungsobjekte liegen auf verschiedenen Rechnern
  - ➤ Sie finden sich und kommunizieren miteinander
- ★ Beispiel Druckmanagement-System
  - ➤ Client-,
  - ➤ Spooler- und
  - ➤ Druckerobjekte
- implizite, nicht-orthogonale Interaktion
  - ◆ (Remote-)Methodenaufrufe
  - ◆ Client-Stub vermittelt Aufruf an Server-Objekt
  - ◆ **At-most-once** / **exactly-once** Semantik

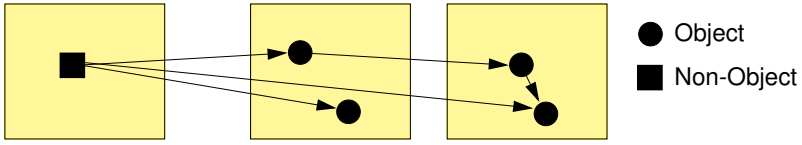

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 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-Seite)
- ◆ CORBA-Objekte können als Aufrufer (Client) agieren




- ➤ Aufrufer müssen nicht unbedingt CORBA-Objekte sein

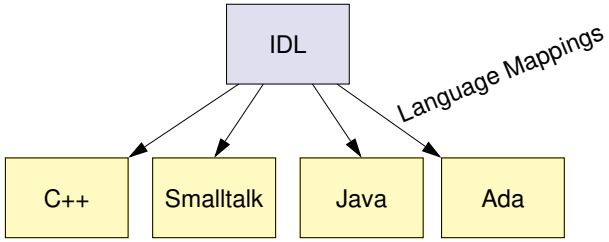
**Achtung:**
Server-Objekt darf nicht mit einem CORBA-Server verwechselt werden.


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 2 Interface Definition Language (IDL)

- Sprache zur Beschreibung von Objekt-Schnittstellen
  - ◆ unabhängig von der Implementierungssprache des Objekts
  - ◆ Sprachabbildung (Language-Mapping) definiert, wie IDL-Konstrukte in die Konzepte einer bestimmten Programmiersprache abgebildet werden
  - ◆ 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. für Perl





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Interface-Definition-Language (2)

- IDL angelehnt an C++
  - ◆ geringer Lernaufwand
  - ◆ eigene Datentypen
    - ➤ Basistypen
    - ➤ Aufzählungstyp
    - ➤ zusammengesetzte Typen
  - ◆ Module
    - ➤ definieren hierarchischen Namensraum für Anwendungsschnittstellen und -typen
  - ◆ Schnittstellen
    - ➤ beschreiben einen von außen wahrnehmbaren Objekttyp
  - ◆ Exceptions
    - ➤ beschreiben Ausnahmebedingungen


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 2 Interface-Definition-Language (4)

- Umsetzung von IDL in Sprachkonstrukte (z.B. C++)

**IDL**

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

**C++**

```
namespace MyModule {
    class MyInterface : ... {
        public:
         virtual CORBA::Long lines();
         virtual void lines( CORBA::Long _val );
         void printLine( const char *toPrint );
        ...
    };
};
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 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 );
    };
};
```

**Java**

```
package MyModule;

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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Interface-Definition-Language (6)

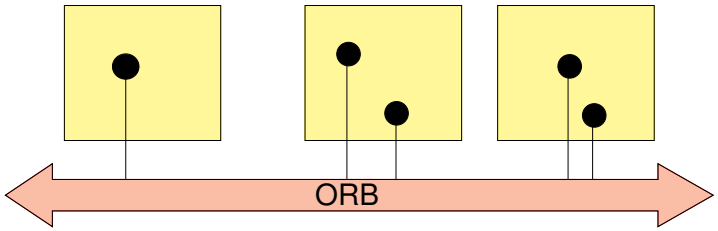
★ Vorteile
- ◆ Schnittstellenbeschreibung unabhängig von Implementierungssprache
- ◆ ermöglicht Unterstützung vieler Programmiersprachen

▲ Nachteile
- ◆ Objektschnittstelle muss in IDL und in der Implementierungssprache beschrieben werden (letzteres kann teilweise automatisiert werden)
- ◆ IDL ist sehr ausdrucksstark (z. B. **`sequence`**)
  - Sprachabbildung für Konstrukte, die in einer Programmiersprache nicht direkt vorhanden sind, ist kompliziert
- ◆ Fähigkeiten einer Sprache, die nicht in IDL beschrieben werden können, können nicht genutzt werden


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 5.4 Object Request Broker – ORB




- Der ORB ist das Rückgrat einer CORBA-Implementierung
- ORB vermittelt Methodenaufrufe von einem zum anderen Objekt
  - ◆ ... innerhalb eines Adressraums / Prozesses
  - ◆ ... zwischen Adressräumen / Prozessen
  - ◆ ... zwischen Adressräumen / Prozessen von verschiedenen ORBs


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Objekte Erzeugen und Binden

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

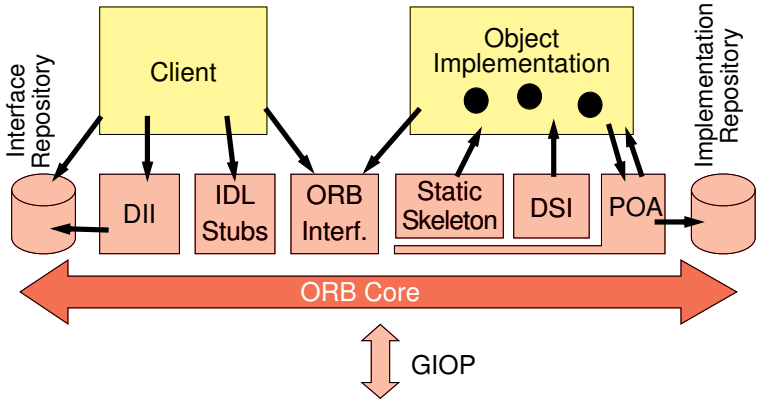
- Binden des Clients an das Server-Objekt
  - ◆ Referenz auf Server-Objekt besorgen
    - ➤ Ergebnis einer Namensdienst-Anfrage
    - ➤ Rückgabewert eines Methodenaufrufs
    - ➤ von "ausserhalb" des Systsems: Benutzer kennt Referenz als String (IORs können in Strings konvertiert werden und umgekehrt)
  - ◆ Erzeugung des Client-Stub
  - ◆ Methodenaufruf über Client-Stub


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 1 Architektur

- Zentrale Komponenten eines ORB




- ◆ mehrere interne Komponenten (teilweise nicht für Anwender sichtbar)
  - ➤ z. B. Stellvertreterobjekte


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Statische Stubs

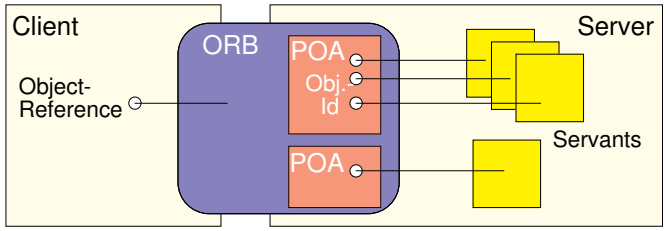
- Stellvertreter auf der Client- und Serverseite
  - ➤ sobald die Schnittstelle eines Objekts feststeht, können Stubs daraus erzeugt werden
  - ➤ Statische Stubs werden aus der IDL Beschreibung automatisch erzeugt
  - ➤ Auf Serverseite werden die Stubs (ausgefüllte) *Skeletons* genannt
- Aufgaben der Stubs
  - ➤ Verpacken und Entpacken der Parameter (Marshalling)
  - ➤ Abschicken bzw. Entgegennehmen von Methodenaufruf-Mitteilungen über den ORB-Core


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Portable Object Adaptor (POA): Terminologie

- ***Server***: Ausführungsumgebung (Prozess) für CORBA-Objekte
- ***Servant***: konkretes Sprachobjekt, das CORBA-Objekt(e) implementiert
- ***Object***: abstraktes CORBA-Objekt (mit Schnittstelle und Identität)
- ***Object Id***: Identität, mit der ein POA ein bestimmtes *Object* identifiziert
- ***POA***: Verwaltungseinheit innerhalb eines Servers
  - ➤ Namensraum für Object-Ids und weitere POAs
  - ➤ setzt Aufruf an einem *Object* in einen Aufruf an einem Servant um
- ***Object Reference***: Enthält Informationen zur Identifikation von Server, POA und Object





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Object Adaptor - Vorschau

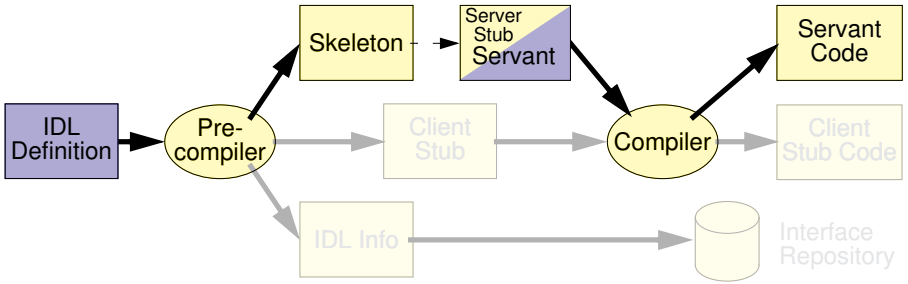
- Lokaler Repräsentant des ORB-Dienstes, direkter Ansprechpartner eines CORBA Objekts
  - ◆ Generiert CORBA-Objektreferenzen (für neue Objekte)
  - ◆ Bildet Objektreferenzen auf Implementierungen ab
  - ◆ Bearbeitet ankommende Methodenaufrufe
- CORBA definiert den Portable Object Adaptor
  - ◆ Basis-Funktionalität,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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 4 Objekterzeugung

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



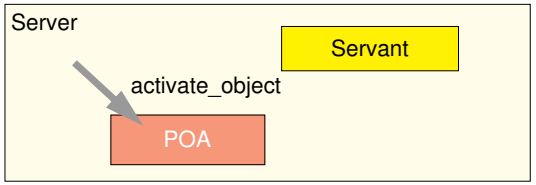

- Prinzipieller Ablauf bei der Definition eines CORBA Objekts


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 4 Objekterzeugung (2)

➥ am Beispiel POA

- Instanziieren des CORBA-Objekts
  - **1**. Instanziieren der Servant-Implementierung im Server
  - **2**. Aktivierung des Servants am Object-Adaptor
    - ➤ Aufruf von **`activate_object`** am POA
    - ➤ Servant erhält dadurch eine *Object Id*





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 4 Objekterzeugung (3)

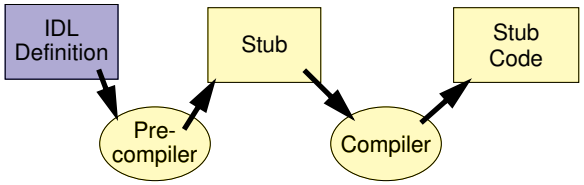
- Herausgabe der Objektreferenz
  - **3**. Servant wird mit *Object Reference* versehen (dadurch wird er zum CORBA-Objekt)
    - ➤ Aufruf von **`servant_to_reference`** am POA
      - liefert Objektreferenz auf CORBA-Objekt (nicht auf Servant)
      - Realisierung der Objektreferenz ist sprachabhängig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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5 Objektnutzung

- Erzeugung eines Stubs
  - Auswahl einer Programmiersprache für die Client-Seite
  - Umsetzung der IDL-Schnittstelle in einen Stub





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5 Objektnutzung (2)

- Empfang eines Parameters mit Objekttyp
  - automatische Erzeugung einer neuen Objektreferenz aus dem Stub-Code
    - realisiert auch entfernte Objektreferenz
    - Objekttyp ist zur Compile-Zeit bekannt: Stub-Code kann erzeugt werden
  - 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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 5 Objektnutzung (3)

- Parameterübergabe (fortges.)
  - Übergabe bei **`out`** und **`inout`** Parametern sprachabhängig
    - ➤ Problem: Programmiersprachen kennen meist keine Möglichkeit mehrere Ergebnisse zurückzugeben
    - ➤ z. B. Java: Einführung von Holder-Klassen
      – z. B. **`AccountHolder`**: kann eine Objektreferenz auf ein Objekt vom Typ **`Account`** aufnehmen
      – Java-Referenz auf **`AccountHolder`**-Objekt wird als **`out`**- oder **`inout`**–Parameter übergeben
      – Ergebnisparameter kann aus Holder-Objekt ausgelesen werden


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5 Objektnutzung (4)

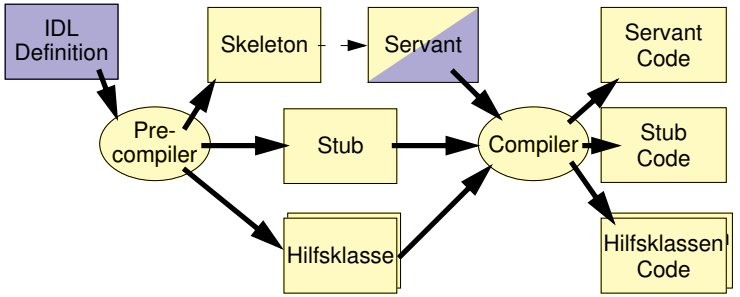
- Typumwandlung
  - Objektreferenz kann einen Typ aus der Vererbungshierarchie der Schnittstellen besitzen
    - ➤ wahrer Objekttyp kann spezifischer in der Vererbungshierarchie sein
  - Erzeugung einer anderen Objektreferenz mit anderem Typ
    - ➤ **`narrow`**-Operation: sprachabhängige Umsetzung
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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5 Objektnutzung (5)

- Sprachabhängige Code-Generierung
  - ➤ z. B. in Java werden aus IDL-Beschreibung neben Stub und Skeleton auch Holder- und Helper-Klasse generiert





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 6 Initialisierung der Anwendung

- Initiale entfernte Objektreferenzen
  - Namensdienst für Registrierung und Anfragen von Objektreferenzen
  - 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 );`**
    - ➤ Referenz kann per String übermittelt werden, z. B. über Datei, TCP/IP, Standardeingabe, Kommandozeile ...


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 5.5 Object-Adaptor

- Aufgaben des Objektadapters
  - Generierung der Objektreferenzen
  - Entgegennahme eingehender Methodenaufruf-Anfragen
  - Weiterleitung von Methodenaufrufen an den entsprechenden Servant
  - Authentisierung des Aufrufers (Sicherheitsfunktionalität)
  - Aktivierung und Deaktivierung von Servants
  - Registriert Serverklassen im Implementation Repository
- Obligatorischer Adapter: Portable-Object-Adaptor
  - definierte Funktionalität für die häufigsten Aufgaben
- weiteres Beispiel: OODB-Adaptor
  - Anbindung einer objektorientierten Datenbank an CORBA
  - OODB-Adaptor repräsentiert alle Objekte in der Datenbank als CORBA-Objekte und vermittelt Aufrufe


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 1 Portable-Object-Adaptor (POA): Ziele

- Portabilität von Objektimplementierungen zwischen verschiedenen ORBs
- Trennung von Objekt (CORBA-Objekt mit seiner Identität) und Objektimplementierung
  - ➤ eine Objektimplementierung kann mehrere CORBA-Objekte realisieren
  - ➤ Objektimplementierung kann bei Bedarf dynamisch aktiviert werden
  - ➤ persistente CORBA-Objekte (Objekte, die Laufzeit eines Servers überdauern)

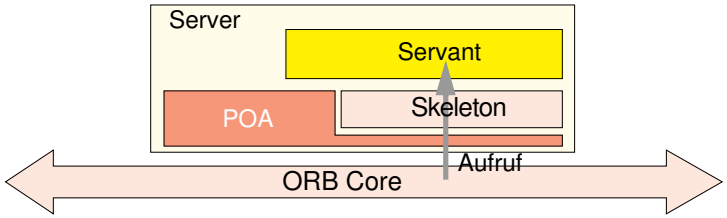
  ➥ eine Object Reference beim Client steht für ein bestimmtes CORBA-Objekt — nicht unbedingt für einen bestimmten Servant!
- Mehrere POA-Instanzen (mit unterschiedlichen Strategien) innerhalb eines Servers möglich


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 2 Portable-Object-Adaptor (POA): Überblick

- Jeder Servant kennt seine POA-Instanz
  - POA-Instanz kennt die an ihm angemeldeten Objekte
  - POA stellt Kommunikationsplattform bereit und nimmt Aufrufe entgegen (für seine Objekte)




- Verhältnis ORB Core - Server - POA: Alternativen
  - ➤ ORB Core ist Teil des Server-Prozesses (Bibliothek), Kommunikation zu POA erfolgt lokal oder
  - ➤ ORB-Serverprozess und lokale IPC zu den Server-Prozessen mit deren POAs


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 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 erzeugbar
  - Erzeugung am RootPOA bzw. an untergeordnetem POA (Vater-POA)
  - neuer POA bekommt eigenen Namen (String)
  - baumförmige, hierarchische POA-Struktur mit Wurzel beim Root-POA
  - POA-Instanzen teilen sich Kommunikationskanal
    - ➤ Objektreferenz enthält POA-Name: POA kann ermittelt werden


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 4 Erzeugung von CORBA-Referenzen

- Normalfall
  - Servant existiert und wird dem POA mit `activate_object(...)` gemeldet
  - Referenz wird dann mit `servant_to_reference(...)`-Aufruf am POA erzeugt
- **Implizite** Aktivierung
  - POA erzeugt automatisch eine *Object Reference* wenn ein (nicht aktivierter) Servant als Parameter oder Ergebnis "nach aussen" übergeben wird
- **Explizite** Erzeugung einer Referenz
  - *Object Reference* wird erzeugt, ohne dass ein Servant existiert
    - damit entsteht ein abstraktes CORBA-Objekt, zunächst ohne Implementierung
  - POA hat verschiedene Möglichkeiten, ankommende Aufruf zu bearbeiten:
    - "Default Servant" bearbeitet den Aufruf
    - Servant wird bei Bedarf dynamisch erzeugt
  - sie Abschnitt POA-Policies


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5 Deaktivierung von Objekten, POA und Server

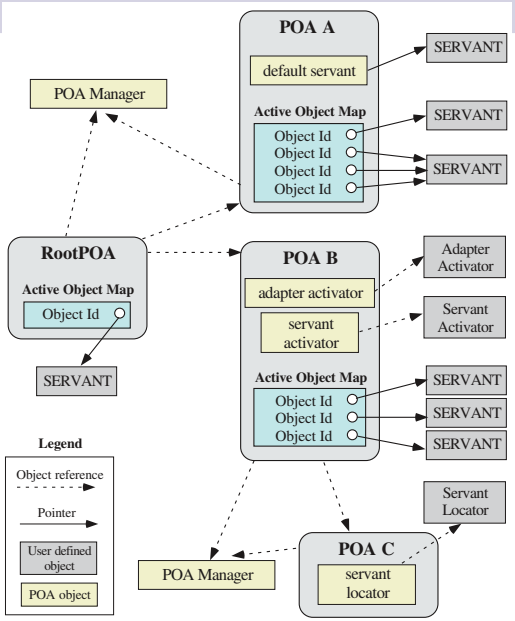
- Ressourcen-Problem
  - sehr viele CORBA-Objekte "in" einem Server
  - mehrere POAs in einem Server
  - Objekte oder auch POAs werden evtl. nur selten genutzt
  - Server-Prozess wird ggf. nur selten benötigt
- CORBA-Objekte können ihren Servant und ihre POA-Instanz überleben
  - persistente Objekte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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 6 POA-Architektur

- Implementation Repository
  - Basis für persistente Referenzen
  - kennt alle Server und zumindest die Root-POAs
- POA Manager
  - steuert POA
- Adapter Activator
- Servant Manager
- POA Policies





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 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-Repository
     - Implementation-Repository hält dauerhafte Information über das CORBA-Objekt, insbesondere wie dieses wieder aktiviert werden kann
     - z. B. Programmdatei für Neustart
     - Starten eines neuen Servers und Aktivieren des Root-POAs
     - Aufruf wird an neue Kommunikationsadresse weitergeleitet *(Location Forward)*


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 7 Bearbeitung von ankommenden Aufrufen (2)

- Aktivierungsmechanismus
  - Speicherung von Daten und Protokoll zwischen Implementation-Repository und POA ist herstellerspezifisch
  - JDK 1.4: **`orbd`**-Prozess
  - Unterstützung im RPC-Protokoll
  - Location-Forward: Stubs werten Weiterleitung aus und kontaktieren Objekt an angegebener Kommunikationsadresse solange möglich


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 7 Bearbeitung von ankommenden Aufrufen (*2*)

2. POA finden
   - ORB lokalisiert zuständigen POA
   - falls POA deaktiviert: ORB ruft *Adapter Activator* bei "Vater-POA" auf
3. Servant finden
   - ORB leitet Aufruf an zuständigen POA weiter
   - POA findet Servant entsprechend seiner *Servant Retention*- und *Request Processing*-Strategie
     - ggf. Servant neu instanziieren
     - sie Abschnitt POA-Policies
4. Skeleton finden
   - im letzten Schritt gibt der POA den Aufruf an das IDL Skeleton weiter
   - Skeleton entpackt Parameter und ruft die eigentliche Servant-Operation auf


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 8 POA Policies

- Konfiguration eines POA bei Erzeugung durch Policy-Objekt
  - nicht bei Root-POA!
  - bei Erzeugung eines POA werden vorab erzeugte Policy-Objekte als Konfiguration übergeben
- Threading policy
  - Aufrufbearbeitung multi-threaded oder single-threaded
- Lifespan policy
  - gibt an, ob CORBA-Objekte, die von POA erzeugt werden, transient oder persistent sind
    - **`TRANSIENT`** für Objekte, die Servant nicht überleben
    - **`PERSISTENT`** für Objekte, die POA und Servant überleben können
- ObjectId uniqueness policy
  - eindeutige IDs über alle Zeit oder nicht


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 8 POA Policies (2)

- ObjectId Assignment policy
  - system- oder benutzervorgegebene Objekt-IDs
- Implicit activation policy
  - falls implizite Aktivierung von Servants und Erzeugung von CORBA-Referenzen unterstützt werden soll
- Servant retention policy
  - Tabelle aktiver Objekte ordnet Servants den CORBA-Referenzen zu
  - Alternativ: bei ankommendem Aufruf wird jedesmal ein Servant dem angeforderten CORBA-Objekt zugeordnet
- Request processing policy
  - nur aktivierte Objekte sind bekannt
  - ein Default-Servant kann für alle nicht aktivierten Objekte einspringen
  - Servant-Manager kann für unbekannte Objekte einen Servant generieren


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 5.6 Dynamic-Invocation-Interface (DII)

## 1 Überblick

- Typ eines Objekts zur Entwicklungszeit einer Anwendung nicht bekannt
  - z. B. Name-Server-Implementierung, Verzeichnisdienst etc.
  - Wie aufrufen?
- Abfrage der Objektschnittstelle
  - Ermittlung erfolgt über das Interface Repository
  - Methode **`get_interface()`** bei jedem CORBA-Objekt
    - ➤ liefert Referenz auf CORBA-Objekte, die Schnittstelle eines Objekts beschreiben
    - ➤ Methoden, Parameter und deren Typen können erfragt werden


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 1 Überblick (2)

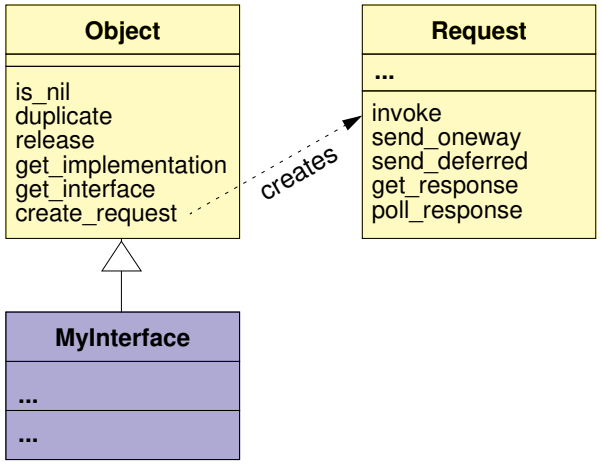
- Dynamische Konstruktion der Aufrufparameter
  - generische Aufrufschnittstelle **`create_request()`** bei jedem CORBA-Objekt
    - ➤ Benennung der Methode als String
    - ➤ Übergabe der Parameter eingepackt in Typ **`any`**
- Einzelne Schritte des Aufrufs (im Language Binding abgebildet)
  - ermittle Methodensignatur aus dem Repository
  - erstelle die Parameterliste
  - erstelle die Aufrufbeschreibung (Request)
  - führe Methodenaufruf aus
    - – als RPC
    - – asynchroner RPC
    - – über Datagramm-Kommunikation (ohne Antwort)


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 1 Überblick (3)

- IDL-Definitionen für das DII





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



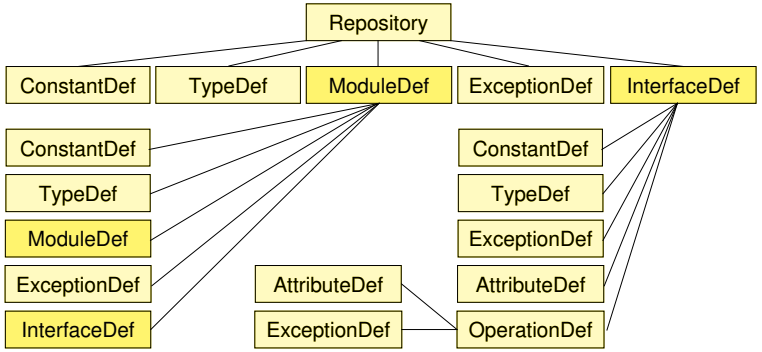
## 2 Interface Repository

- Datenbank für Schnittstellendefinitionen in IDL
- Abfrage der Datenbank
  - für Typechecking
    - ➤ bei Inter-ORB-Operationen: Hinterlegung in mehreren Repositories
  - zum Abruf von Metadaten für Clients und Tools: dynamische Aufrufe, Debugging, Klassenbrowser
  - Implementierung der Methode *get_interface* eines jeden Objekts
- Schreiben der Datenbank
  - durch IDL Compiler
  - durch Schreibmethoden


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Interface Repository (2)

- IDL-Konstrukte werden jeweils als CORBA-Objekt realisiert
- Zusammenhänge zwischen den Repository-Objekttypen:




  - Komponenten einer IDL-Datei werden als Objekte, die in IDL spezifiziert sind abgelegt
  - Hierarchie der Komponenten bzw. des Interface Repositories


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Aufrufarten

- Normaler Aufruf
  - Aufruf von **`invoke`** am Request-Objekt
    - kehrt zum Aufrufer zurück, wenn eigentlicher Aufruf durchgeführt
- Asynchroner Aufruf
  - Aufruf von **`send_deferred`** am Request-Objekt
    - eigentlicher Aufruf wird angestoßen
    - Aufrufer kann weiterarbeiten
    - Ergebnissynchronisation durch **`get_response`**
    - Ergebnisabfrage (Polling) mit **`poll_response`**


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 2 Interface-Repository (3)

- Repräsentation von Typen durch TypeCode-Objekte
  - Repräsentation der Basistypen:
    **`int, float, boolean`**
  - Repräsentation zusammengesetzter Typen:
    **`union, struct, enum, sequence, string, array`**
  - Repräsentation von IDL-basiertenObjekttypen: Interface-Repository-ID
- TypeCode-Objekte repräsentieren Typ und Struktur durch IDL-Typen
  - TypeCode-Objekte haben Operation für Typvergleich
  - TypeCode-Objekte haben Operationen zur Strukturerkundung (z. B. welchen Elementtyp hat eine sequence)


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 3 Aufrufarten (2)

- Datagramm-Aufruf
  - Aufruf von **`send_oneway`** am Request-Objekt
    - Aufruf wird angestoßen
    - es wird nicht auf Ergebnis gewartet (Methode darf kein Ergebnis liefern)
    - Aufruf muss nicht sicher durchgeführt werden („May-be-Once-Semantik“)
  - **`oneway`**-Methoden werden mit dem Schlüsselwort **`oneway`** in IDL markiert


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 4 Dynamic Skeleton Interface (DSI)

- DSI ermöglicht das Entgegennehmen von Methodenaufrufen für Objekte, deren Schnittstelle nur dynamisch ermittelbar ist, z. B. für
  - Bridges zu anderen ORBs
  - CORBA-gekapselte Datenbank
  - dynamisch erzeugte Objekte und Schnittstellen
- Anhand der Parameter wird das Objekt und dessen Signatur ermittelt
- ★ **Beachte:**
  DII und DSI sind jeweils von der Gegenseite nicht von statischen Stubs zu unterscheiden, d. h. voll interoperabel!


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5.7 Inter-ORB-Kommunikation

- Innerhalb einer ORB-Instanz können Objekte mit herstellerspezifischem Protokoll kommunizieren
- Zwischen ORBs verschiedener Herstellern
  - GIOP – General Inter-ORB Protocol (standardisiert in CORBA)
    - Basisprotokoll für RPC-basierte Objektaufrufe
    - Common Data Representation (CDR) definiert Marshalling und Unmarshalling von IDL-Typen
  - IIOP – Internet Inter-ORB Protocol
    - GIOP über TCP/IP
    - Unterstützung durch den ORB für CORBA-Kompatibilität vorgeschrieben
  - GIOP-Implementierungen über andere Protokolle möglich


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5.7 Inter-ORB-Kommunikation (2)

- ESIOP – Environment Specific Inter-ORB Protocols
  - z.B. DCE/ESIOP
  - Inter-ORB-Protokoll auf der Basis von DCE RPC
- IOR – Interoperable Object Reference
  - Darstellung einer Objektreferenz als IDL-Datenstruktur
  - muss verwendet werden für GIOP
  - Repräsentation als String möglich (`object_to_string`)
  - Profile in der IOR
    - für jedes Protokoll eigenes Profil möglich
    - Objekt kann theoretisch mehrere Protokolle unterstützen z.B. IIOP und herstellerspezifisches Protokoll


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 5.7 Inter-ORB-Kommunikation (3)

- IOR legt Typ des Objekts fest: Interface-Repository-ID
- IOR-Profil für IIOP
  - Eintragung eines Profils
    - Hostname des POA
    - TCP/IP-Portnummer des POA
    - Name des POA
    - ObjektID: eindeutiger Bezeichner innerhalb des POA
- Gängige Implementierung in Standard-ORBs
  - Nutzung von IIOP auch innerhalb des ORBs
  - IOR wird immer als eindeutiger Objektbezeichner benutzt


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 5.8 CORBA Services

- Basisdienste eines verteilten Systems – Erweiterungen des ORB

| | |
|---|---|
| Collection Service | Persistent Object Service |
| Concurrency Service | Property Service |
| Enhanced View of Time | Query Service |
| Event Service | Relationship Service |
| Externalization Service | Security Service |
| Naming Service | Time Service |
| Licensing Service | Trading Object Service |
| Life Cycle Service | Transaction Service |
| Notification Service | |

- Services sind über IDL-Schnittstellen aufrufbar


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 1 Naming Service

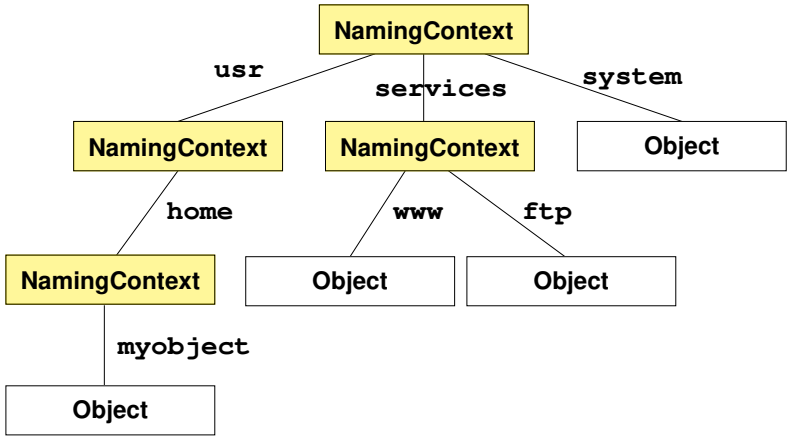
- CORBA definiert einen hierarchischen Namensdienst ähnlich dem UNIX Dateinamensdienst
  - ◆ Namensraum ist baumförmig
  - ◆ Name besteht aus mehreren Komponenten (Silben - syllables)
    z. B. < "usr"; "home"; "myobject" >
    (Entsprechung im Dateisystem wäre "**`/usr/home/myobject`**" )
  - ◆ Schnittstelle des Namensdienstes ist in IDL beschrieben
    - ➤ ansprechbar als CORBA-Objekte


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 1 Namensdienst (2)

- Beispielbaum





Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 1 Namensdienst (3)

- Kontext- und Iteratorschnittstelle

| NamingContext |
|---|
| resolve<br>list<br>destroy<br>new_context<br>unbind<br>bind<br>rebind<br>bind_context<br>rebind_context<br>bind_new_context |

| BindingIterator |
|---|
| next_one<br>next_n<br>destroy |

- ➤ beliebige Objekte können gebunden (**`bind`**) und mit Namen wieder gesucht werden (**`resolve`**)
- ➤ Auflistung erfolgt durch Iterator-Pattern (**`list`**)


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

# 2 Life Cycle Service (1)

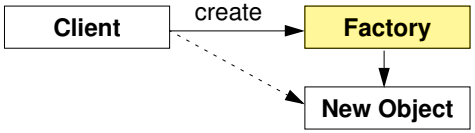
- *Status quo:* Transparenter Zugriff auf verteilte Objekt
- *Anforderung:* Verteilung soll sichtbar und kontrollierbar sein
  - Verwaltung von verteilten Anwendungen
  - Unterstützung von Anwendungen mit expliziten Verteilungsanforderungen
    - Lastverteilung
    - Fehlertoleranz
    - Performance
- *Lsg.:* CORBA Life Cycle Service


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 3 Life Cycle Service (2)

- Lebenszyklus eines Objekts
  - Erzeugung
  - Kopieren, **Verlagern**
  - Löschen
- Life Cycle Service definiert eine gemeinsame Schnittstelle für Operationen während des Lebenszyklus
- ★ Modell des Lebenszyklus
  - Erzeugung eines (entfernten) Objekts:



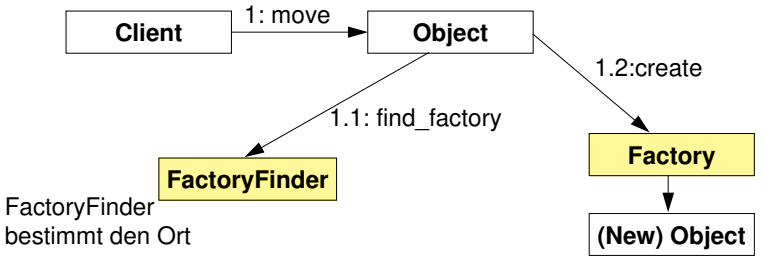

Aufruf einer create-Methode an einem Factory-Objekt


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

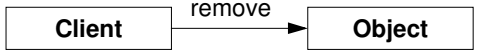


# 3 Life Cycle Service (3)

- Kopieren und **Verlagern**:




- Löschen:



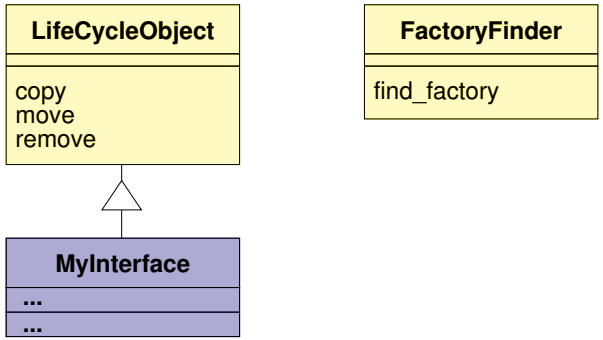


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



# 3 Life Cycle Service (4)

- Objekte müssen das Interface LifeCycleObject implementieren




- **`copy`** und **`move`** benötigen ein **`FactoryFinder`**-Objekt um ein **`Factory`**-Objekt zu finden, das dann die Kopie bzw. das verlagerte Objekt erzeugt
- **`remove`** löscht ein Objekt


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 3 Life Cycle Service (5)

- Implementierung am Beispiel copy
  (1) Client ruft **`copy`**-Methode auf und übergibt einen **`FactoryFinder`**
  (2) Die **`copy`**-Methode stellt entweder der Programmierer des Objekts oder die CORBA-Implementierung bereit
  (3) Die **`copy`**-Methode ruft den **`FactoryFinder`** auf, sucht eine passende **`Factory`** aus und erzeugt mit ihrer Hilfe ein neues Objekt.
  (4) Das neue Objekt wird mit den Daten des bestehenden Objekts initialisiert.
- Nach aussen klare Schnittstelle
- Nach innen offen und **implementationsabhängig** ...
  - Zustandstransfer - sehr problematisch in heterogenen Umgebungen
    - Format des Zusands
    - Art der Migration (schwach oder stark)
  - Behandlung von mehrfädigen Anwendungen
  - Realisierung des **`FactoryFinder`**s


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 4 CORBA Services - Zusammenfassung

- Von OMG standardisierte Spezifikationen von Dienste für Probleme, die in verteilten Systemen häufig auftreten
- Spezifikationen legen fest:
  - Immer: Einheitliche Schnittstelle (IDL)
  - Meistens: generelle Vorgehensweise bei der Problemlösung
  - Manchmal: konkrete Strategien (oft auch offen gelassen bzw. implementierungsabhängig)
- Weitere Dokumentation/Informationen:
  - http://www.omg.org/technology/documents/formal/corbaservices.htm
  - http://developer.java.sun.com/developer/onlineTraining/corba/corba.html


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 5.9 Zusammenfassung

- CORBA hat ein monolithisches Objektmodell
- Eindeutige Objektbezeichner
  - IOR mit jeweiligen Profilen
    - IIOP: Hostname, Portname und ObjektID identifizieren Objekt
- Erzeugung neuer verteilter Objekte
  - Erzeugung eines Servants, Aktivieren des Servants
    - implizites oder explizites Aktivieren
  - Factory-Objekt mit `create()`-Methode (Factory Pattern)
- Schnittstellenspezifische Stellvertreterobjekte
  - Stubobjekte pro IDL-Interface
  - implementieren Methoden und Attribute aus der IDL-Beschreibung


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 5.9 Zusammenfassung

- RPC-basiertes Kommunikationsprotokoll
  - GIOP bzw. IIOP
  - standardisiertes Inter-ORB-Protokoll
  - beliebige eigene Protokolle verwendbar
  - Codierung der Adressen in IOR-Profilen
- Parameterübergabe
  - IOR wird als Objektbezeichner bei Parameterübergabe übermittelt
  - ein ausgewähltes Profil bestimmt Typ der zu instanziierenden lokalen Stub-ImplementierungZusammenfassung
  - Automatische Generierung der Stellvertreter
  - IDL-Compiler (eigentl. Präcompiler) erzeugt Stellvertreter und Skeletons


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 5.9 Zusammenfassung

- Namensdienst zum Finden von Objekten
  - konfigurierter Namensdienst über ORB-Interface erreichbar
  - `resolve_initial_reference("NamingService");`
- Aktivierung und Deaktivierung
  - nicht im Detail spezifiziertes Konzept zur Aktivierung und Deaktivierung
  - persistente Objektreferenzen durch POA-Konfiguration
  - Implementierung des Aktivierungsmechanismus ist herstellerabhängig
- Flexibler Objektadapter
  - POA
  - ServantManager für dynamisch erzeugte Servants
  - DefaultServant für Anbindung an große Objektmengen


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 5.10 Status

- Marktprobleme
  - Späte Entwicklung eines Komponentenmodells
  - Keine Integration des Webs
- Technische Probleme
  - Komplexe Sprachmappings
  - Komplexe APIs
  - Opaque Referenzen
- Standardisierung
  - Demokratischer Prozess
  - Unterschiedliche Benutzergruppen
  - Keine Referenzimplementierung nötig!


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



## 5.10 Status

**He06.** Michi Henning: *The Rise and Fall of CORBA*. ACM Queue, Volume 4, Issue 5, 2006.

- CORBA bildetet Anfang der 90iger die Plattform für verteilte Anwendungen
  - siehe Motivation
- CORBA bildet aktuell eine Nischentechnologie
  - Verwendung in Intranets
  - Entwicklung im Bereich von Embedded- und Echtzeit-Systemen


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.