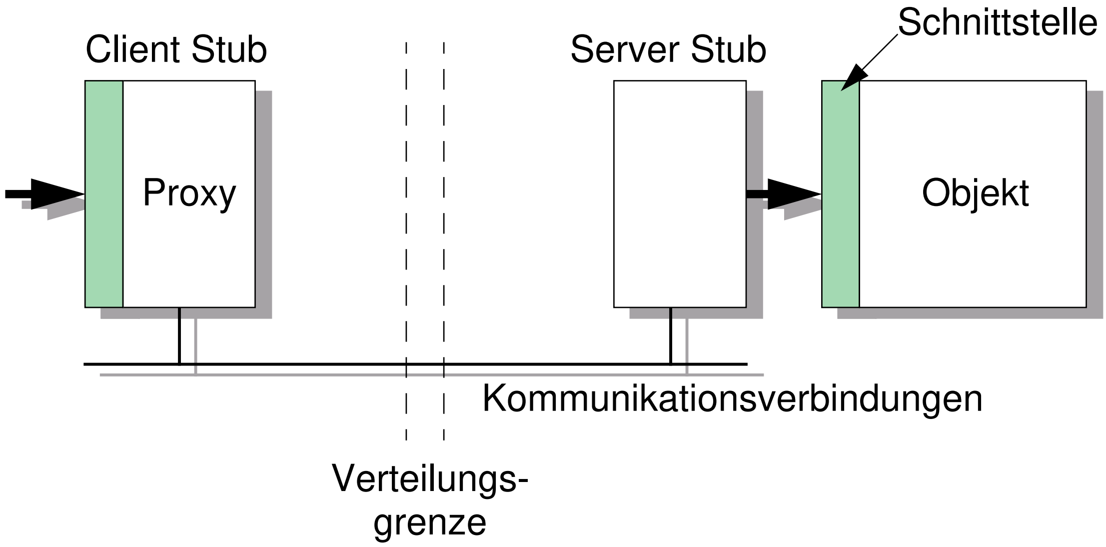
Verteiltes Programmieren mit CORBA

❑ Überblick
- OOP und Verteilung, Middleware
- Motivation für CORBA
- OMA Architektur
- ORB – Object Request Broker
- Services
- Facilities

OOP und Verteilung

- Objektinteraktionen zwischen objektorientierten Programmen *oder* Aufteilung eines Programmes in interagierende Programmteile
  - In beiden Fällen interagieren Objekte über die Grenze eines Adressraums hinweg. Üblicherweise ist eine solche Interaktion nicht von der jeweiligen Programmiersprache direkt unterstützt, so daß zusätzliche Mechanismen ins Spiel kommen müssen.

- Proxy/Stub-Prinzip



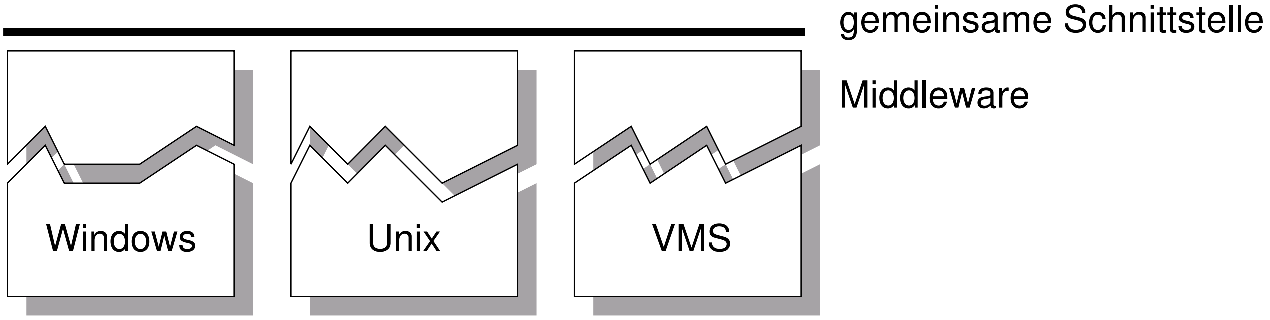
  - Ein Proxy ist ein Vermittler, ein Stub ist ein Stellvertreter.
  - Mit Hilfe von Stubs kann die Objektinteraktion über Adreßraumgrenzen realisiert werden. Auf der Client-Seite gibt es das Proxy- oder Client-Stub-Objekt. Es stellt die gleiche Schnittstelle wie das anzusprechende Objekt selbst bereit. Über einen Kommunikationskanal, der z.B. durch UNIX Sockets oder ähnliches realisiert wird, kommuniziert der Client-Stub mit dem Server-Stub, der letztlich den am Proxy registrierten Methodenaufruf am richtigen Objekt durchführt.
  - Der Client-Stub ist ein Stellvertreter für das eigentliche Objekt, der Server-Stub ein Stellvertreter für den Aufrufer.

OOP und Verteilung (2)

- ❑ Anforderungen
  - Überwindung heterogener Hardware- und Softwarestrukturen
    – verschiedene Hardware
    – verschiedene Betriebssysteme
    – verschiedene Programmiersprachen
    - Durch die Verteilung von Programmteilen auf mehrere Rechner entsteht meist automatisch das Problem der Heterogenität der verwendeten Komponenten.
  - Ortstransparenz
    – statische Konfiguration
    – Objektmigration
    - Ist es unwichtig, auf welchem Rechner ein Objekt nun wirklich implementiert ist, so lassen sich zum einen Konfigurationsänderungen bei der Verteilung ohne das Ändern der Applikation durchführen (statische Änderungen). Zum anderen ist es eventuell auch möglich Objekte zur Laufzeit von einem Rechner zum anderen wandern zu lassen (dynamische Änderung).
  - Globale Dienste
    – z.B. Namensdienste, Transaktionsdienst, Persistenz,
    Kontrolle der Nebenläufigkeit
    - Objekte müssen sich nun finden. Dazu wird ein übergreifender Namensdienst benötigt.
    - Andere höherwertige Dienste können für die verteilte Applikation nicht lokal erbracht werden, sondern müssen übergreifend von allen Rechnersystemen bereitgestellt werden.

Middleware für verteiltes Programmieren

❏ Middleware – Software zwischen Betriebssystem und Anwendung



- Bereitstellung von Diensten für verteilte Anwendungen
- **gesucht:** Middleware für verteiltes objektorientiertes Programmieren

  - Middleware ist klassisch eine Softwarekomponente, die Dienste für verteilte Anwendungen bereitstellt. Dabei abstrahiert Middleware von der eingesetzten Hardware- und Softwarearchitektur und bietet eine gemeinsame Schnittstelle an.

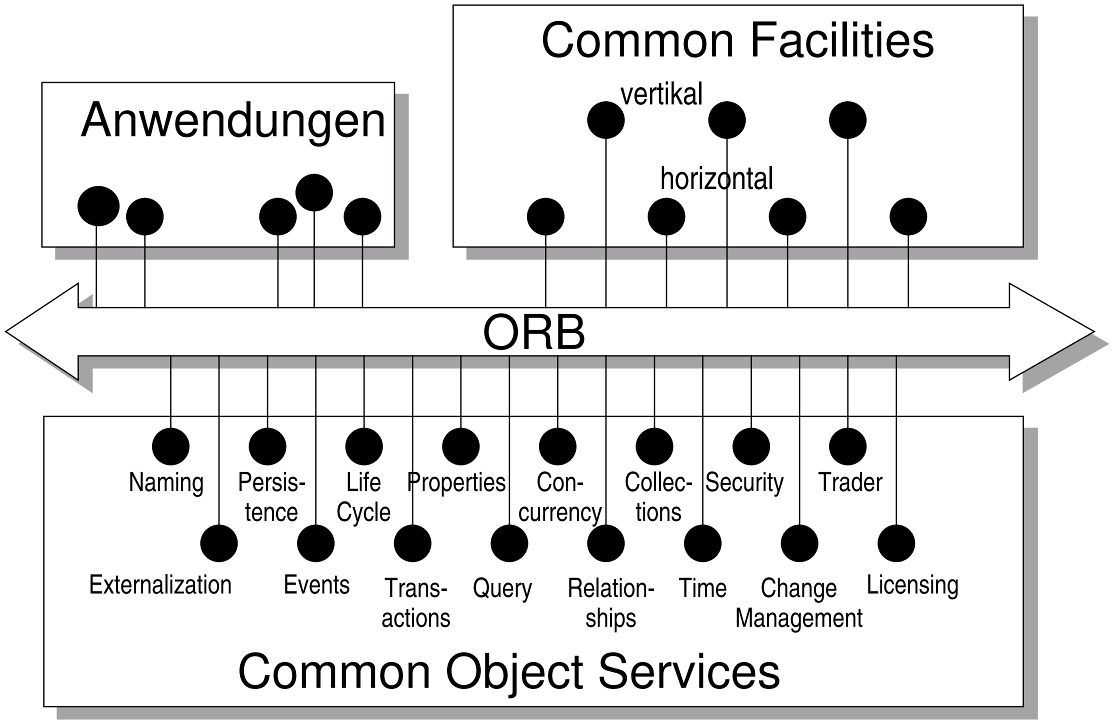
❏ CORBA – Common Object Request Broker Architecture
Standard der OMG

  - Gemeinsame Architektur für einen Object Request Broker (ORB). Ein ORB ist eine Vermittlungseinheit für den Aufruf von verteilten Objekten. Das „gemeinsam“ bezieht sich auf die gemeinsame Anstrengung der wichtigsten IT Firmen in diesem Bereich, zusammengeschlossen in der OMG (Object Management Group), einen Standard für verteiltes Programmieren mit Objekten zu entwickeln.

Motivation für CORBA

- ❏ Middleware, die abstrahiert von
  - Hardware,
  - Betriebssystem und
  - Programmiersprache.
    - CORBA ist damit eine Middleware für das verteilte Programmieren mit Objekten.
- ❏ Interoperabilität
  - lauffähige Programme auf verschiedenen CORBA Implementierungen
  - Kooperation verschiedener CORBA Implementierungen
    - CORBA nimmt damit für sich in Anspruch nicht nur von Hardware, Betriebssystem und Programmiersprache zu abstrahieren, sondern auch von der Implementierung des CORBA Systems selbst.
- ❏ Einbindung von Altapplikationen
  - Kapselung von Altanwendungen in CORBA Objekte
    - Ein CORBA Objekt verdeckt die Altapplikation. Aufrufe an dem Objekt werden in entsprechende Interaktionen mit der Altapplikation umgesetzt.
- ❏ Vision der Business Objects / Components
  - alle Geschäftsdaten und -vorfälle sind CORBA Objekte
    - Damit würde die gesamte Geschäftswelt die „gleiche Sprache“ sprechen und mit Hilfe von CORBA Objektinteraktionen miteinenander in Kontakt stehen können.

Überblick über die Architektur



❑ Object Management Architecture – OMA

- Die CORBA Architektur besteht aus dem ORB (Object Request Broker). Dieser vermittelt Aufrufe zwischen Objekten. Der ORB weiß wo welches Objekt liegt. Die Anwendungsobjekte können untereinander mit Hilfe des ORB kommunizieren.
- Die Common Object Services sind allgemeine standardisierte Dienste, die eine CORBA Implementierung anbieten kann bzw. muß, z.B. den Naming Service zum Auffinden von Objekten anhand eines Namens.
- Die Common Facilities sind Dienste, die anwendungsspezifisch sind. Horizontale Facilities sind relativ breit verwendbare Dienste wie z.B. Benutzerinterface oder Anwendungsmanagement. Vertikale Facilities definieren branchenspezifische Dienste z.B. Dienste des Gesundheitswesens oder Dienste für Großhändler.

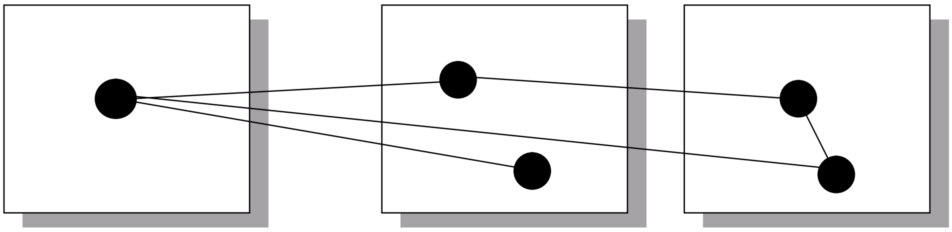
Überblick (2)

- ❏ objektbasierte Client/Server Architektur
- ❏ verteilte Objekte
  - – Identität
  - – Zustand
  - – Methoden
  - – Objekte können Client und Server sein
  - – Server sind immer Objekte
  - – Clients müssen keine Objekte sein
    - Da Clients keine Objekte sein müssen, muß man kein CORBA Objekt erzeugen, um mit einem anderen Objekt in Interaktion zu treten.

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

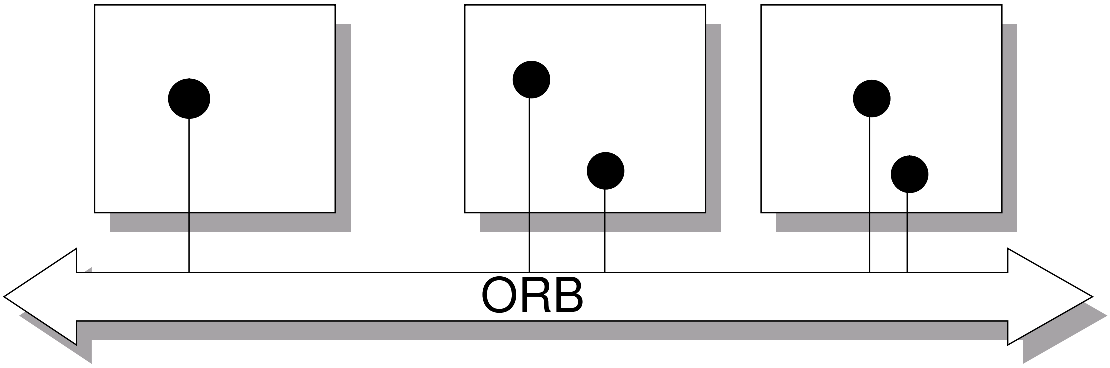
- Server Objekte sind Server im Sinne der Client/Server Interaktion. Sie werden über Methodenaufrufe angesprochen.
- CORBA Server sind Einheiten des darunterliegenden Betriebssystems, die CORBA Objekte beherbergen, z.B. UNIX Prozesse.

Anwendungsobjekte



- ❑ Anwendungsobjekte liegen auf verschiedenen Rechnern
  - Die Anwendungsobjekte sind die Einheiten der Verteilung in CORBA. Objekte auf einem Rechner können auf eine oder mehrere Systemeinheiten verteilt sein, z.B. auf mehrere Prozesse eines UNIX Systems.
- ❑ Sie finden sich und kommunizieren miteinander
  - CORBA bietet den Objekten eine transparente Kommunikation, d.h. die Objekte müssen nicht wissen, wo der jeweilige Kommunikationspartner liegt.
- ❑ Beispiel Druckmanagement-System
  - Client-,
  - Spooler- und
  - Druckerobjekte
    - Verschiedene Anwendungsobjekte liegen verteilt im System. Sie sind für den Zugriff auf das Drucksystem (Clients), für das Puffern von Druckaufträgen (Spooler) und für das eigentliche Drucken zuständig.

Object Request Broker – ORB



- ❑ Der ORB ist das Rückgrat einer CORBA Implementierung
  - Der ORB vermittelt Aufrufe zwischen Objekten. Der ORB weiß wo welches Objekt liegt.
- ❑ Alle Kommunikations zwischen den Objekten läuft über den ORB

IDL Schnittstellen

- ❏ Sprachunabhänge Schnittstellendefinition
  - IDL = Interface Definition Language
  - IDL ist syntaktisch an C++ angelehnt
    - Durch die Sprachunabhängigkeit ist man nicht an die Features einer bestimmten Sprache gebunden. Es ist nur das an Parameterübergabe möglich, was durch IDL definiert wird.
    - Ein Nachteil ist, daß die Schnittstelle immer noch zusätzlich in IDL erstellt werden muß.
- ❏ Sprachanbindung durch standardisierte "Language bindings"
  - Abbildung der IDL Typen und Deklarationen auf eine Sprache
  - standardisiert sind: C, C++, Smalltalk, [Java]
    - Ein anderer Nachteil von IDL ist, daß die Sprachanbindung recht komplex wird, falls IDL Dinge erlaubt, die in der Sprache selbst nicht möglich sind.
- ❏ IDL Schnittstellen erlauben volle Interoperabilität

IDL und das Binden

- Im folgenden wird vereinfacht dargestellt, wie man ein CORBA Objekt in das System bringt und wie es letztlich aufgerufen wird:

❏ Serverseitige Implementierung
- Definition der Schnittstelle in IDL
- Implementierung des Serverobjekts in einer Sprache der Wahl
- Anmeldung des Objekts am System

❏ Clientseitige Implementierung
- Ermittlung des Objektidentifikators / der Objektreferenz
  - Die Ermittlung kann über einen Namensdienst erfolgen. Oder die Referenz kann als Rückgabeparameter eines Methodaufrufs bekannt werden.
  - Objektreferenzen lassen sich auch in Strings umwandeln und speichern. Der umgekehrte Weg kann dann dazu benutzt werden, um an eine Objektreferenz zu gelangen.
- Erzeugung des Client Stubs
- Methodenaufruf über den Stub

Services

- ❏ Basisdienste eines verteilten Systems
  Erweiterungen des ORB
  - Namensdienst
  - Transaktionen
  - Persistenz
  - ...
- ❏ Dienste werden durch IDL Schnittstellen angesprochen
  - z. B. Schnittstellen zum Namensdienst und dessen Strukturen
    - Dadurch, daß alle Dienste letztlich durch IDL Schnittstellen repräsentiert sind, können diese mit dem bereits bestehenden Mechanismus „Methodenaufruf" benutzt werden, ohne daß man zusätzliche Anteile im Language Binding braucht. Neue Dienste können ohne Probleme hinzugefügt werden.

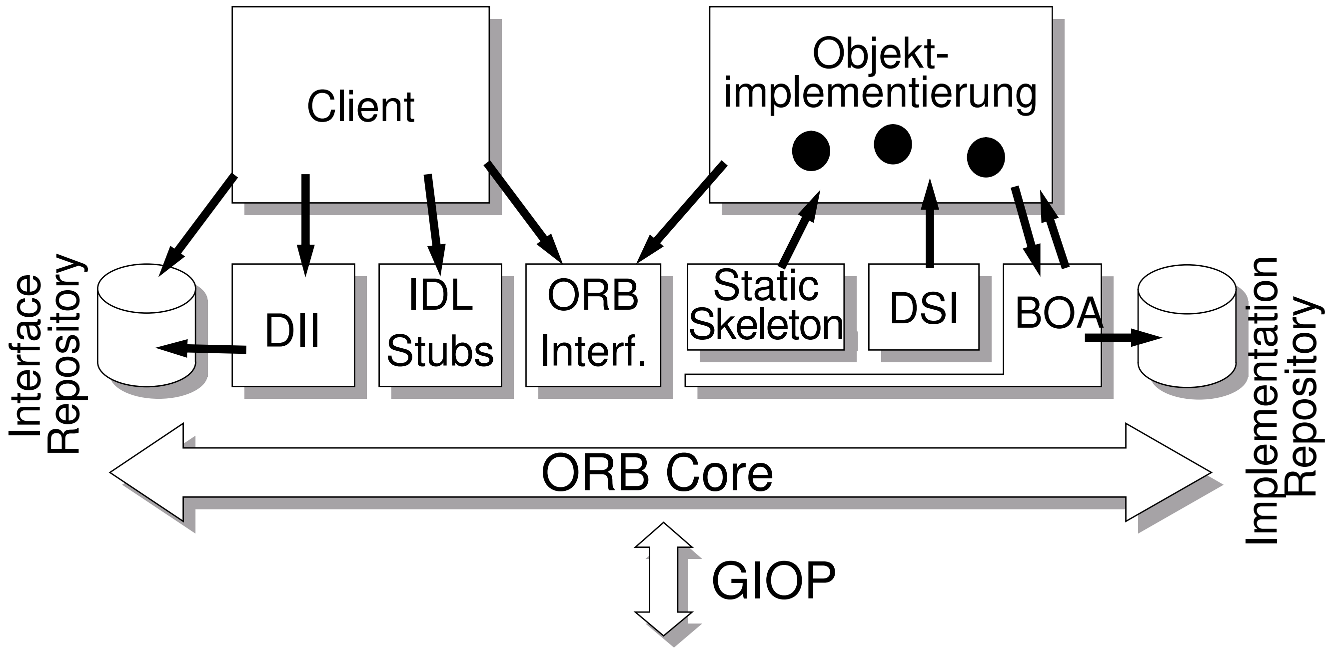
Common Facilities

- Höherwertige Dienste
  - Die Facilities unterscheiden sich von den Services dadurch, daß Services allgemeine Basisdienste definieren, die einerseits für nahezu alle Anwendungen von Bedeutung sind und andererseits sich kaum ohne Eingriff in die CORBA Implementierung selbst außerhalb definieren lassen, man denke z.B. an den Transaktionsdienst.
- Unterscheidung von allgemeinen und branchenbezogenen Facilities
  - Benutzerinterface
  - Dienste zur Unterstützung des Großhandels
- Spezifiziert mit IDL Schnittstellen

CORBA Implementierung

- ❑ Eine CORBA Implementierung muß enthalten
  - eine Implementierung der Core Architecture
  - ein Language Binding (z.B. für C++)
- ❑ Sie kann enthalten
  - beliebige Anzahl von Diensten
    - Heute gibt es kaum CORBA Implementierungen, die wenigstens die meisten definierten Dienste anbieten. Meist wird nur ein Namensdienst angeboten.
  - Interoperabilität mit anderen CORBA Implementierungen (GIOP/IIOP)
    - Eine CORBA 2.0 konforme Implementierung muß aber wenigstens IIOP anbieten
- ❑ Der Implementierung ist freigestellt **wie** sie die Anforderungen realisiert
  - viele verschiedene Implementierungen möglich
    - Die Implementierung des ORB kann z.B. durch einen oder mehrere Daemon-Prozesse erfolgen. Eine andere Implementierung des ORB könnte lediglich aus dazugebundenen Bibliotheken bestehen.

Der ORB und seine zentrale Komponenten



Aufgaben

- ❏ Kommunikation zwischen Objekten
  - innerhalb eines Adreßraums / Prozesses
  - zwischen Adreßräumen
  - zwischen Adreßräumen verschiedener ORBs
    - Mit den verschiedenen möglichen Kommunikationsformen stellt der ORB die Ortstransparenz bei Methodenaufrufen sicher.
- ❏ Management der Objekte
- ❏ Bekanntmachung der Objekte
    - Hierbei ist nicht gemeint, daß der ORB einen Namensdienst implementiert. Vielmehr macht er bekannt für welche Schnittstellen Objekte vorhanden sind.
    - Der ORB vergibt auch einen internen Namen für jedes Objekt, die Objektreferenz. CORBA 2.0 definiert die sogenante Interoperable Object Reference (IOR), die auch zwischen CORBA Implementierungen eindeutig ist.
- ❏ Bereitstellen von Metadaten
    - Repositories stellen Schnittstellen und Implementierungen bereit.

Statische Stubs

❏ Stellvertreter auf der Client- und Serverseite

- Wenn feststeht, welche Schnittstelle das aufzurufende Objekt besitzt, können die Stubs statisch gebunden werden.

❏ Statische Stubs werden aus der IDL Beschreibung automatisch erzeugt

- Mittels Precompiler und Compiler werden die Stubs aus der IDL Beschreibung erzeugt. Dabei helfen oft entsprechende Makefiles bei der Erstellung.

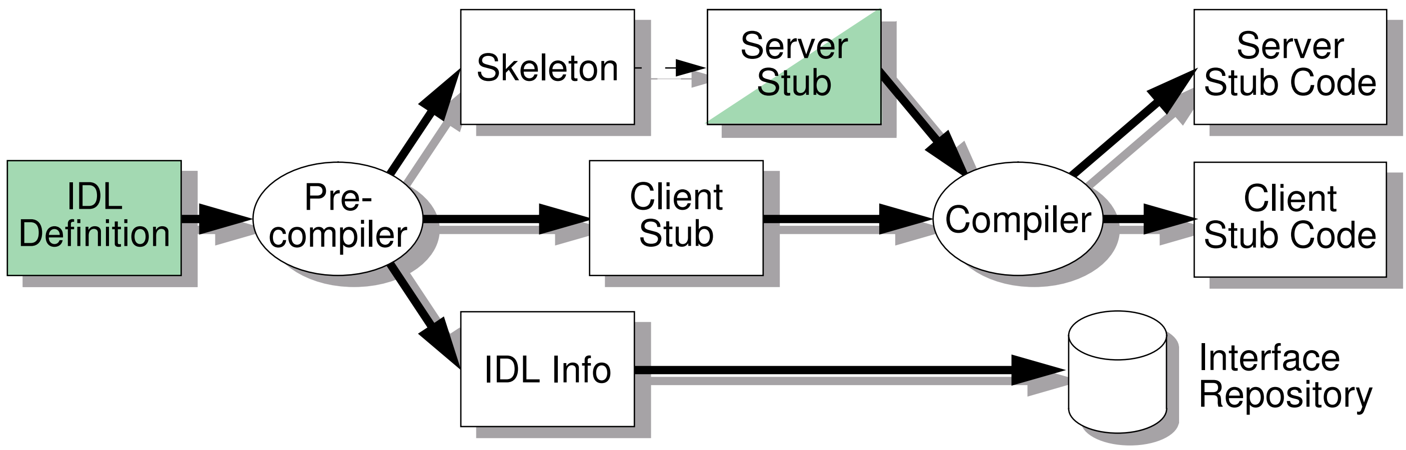
❏ Auf Serverseite werden die Stubs (ausgefüllte) *Skeletons* genannt

- Die Skeletons müssen mit der Implementierung des Objekts ausgefüllt werden. Sie enthalten lediglich das Methodenskelett.

❏ Aufgaben:

- Verpacken und Entpacken der Parameter (Marshalling)
- Abschicken bzw. Entgegennehmen von Methodenaufrufen

Stuberzeugung



❏ Prinzipieller Ablauf bei der Definition eines CORBA Objekts

- Aus der IDL Definition der Objektschnittstelle werden mittels eines Precompilers mehrere Ausgabedateien erzeugt.
- Das Skeleton enthält ein Skelett für das Serverobjekt. Das Skelett muß ausgefüllt werden mit den Methoden der Objektimplementierung und bildet dann den Server Stub inklusive der Implementierung des Objekts selbst.
- Der Client Stub enthält den Code für den Client-Anteil.
- Eine zusätzliche Datei enthält die Informationen, die später in das Interface Repository geladen werden.
- Client und Server Stub müssen in der Regel noch von einem Compiler übersetzt werden. Die daraus entstehenden Codestücke werden in der Regel dynamisch oder statisch in die Applikation eingebunden.

Interface Repository

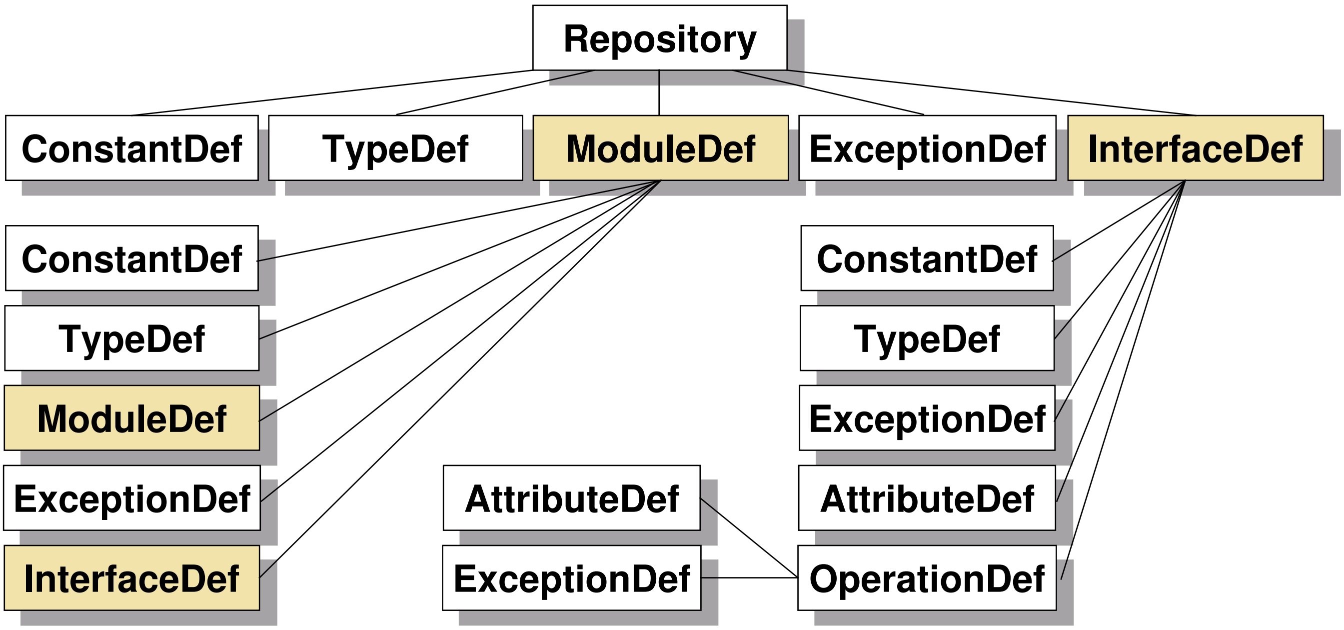
❏ Datenbank für Schnittstellendefinitionen in IDL
    - Alle IDL Schnittstellen werden dort abgelegt.

❏ Abfrage der Datenbank zum
- Typechecking
    - Der ORB kann prüfen, ob die Schnittstelle des aufgerufenen Objekts mit der Schnittstelle übereinstimmt, die der Client Stub des Objekts repräsentiert.
- Inter-ORB-Operationen: Hinterlegung in mehreren Repositories
- Metadaten für Clients und Tools: dynamische Aufrufe, Debugging, Klassenbrowser
    - Beispielsweise für dynamische Aufrufe, d.h. für solche, bei denen der Typ des Objekts zur Compilezeit noch nicht feststand, kann mittels Interface Repository der genaue Typ der Schnittstelle, der Operationen und Parameter zur Laufzeit ermittelt werden.
- Implementierung der Methode *get_interface* eines jeden Objekts
    - Jedes Objekt hat eine Methode get_interface mit dem sich ein Verweis auf die dazugehörige Schnittstellenbeschreibung ermitteln läßt. Diese Beschreibung kommt dann aus dem Interface Repository.

❏ Schreiben der Datenbank
- durch IDL Compiler
    - Bei der Generierung von Stubs und Language Binding werden auch Informationen zum Laden des Interface Repositories erzeugt.
- durch Schreibmethoden
    - Das Interface Repository besitzt auch Schreibmethoden, mit denen explizit Einträge in der Interface Datenbank vorgenommen werden können.

Interface Repository (2)

❏ Gespeicherte Informationen / Aufbau von IDL Dateien

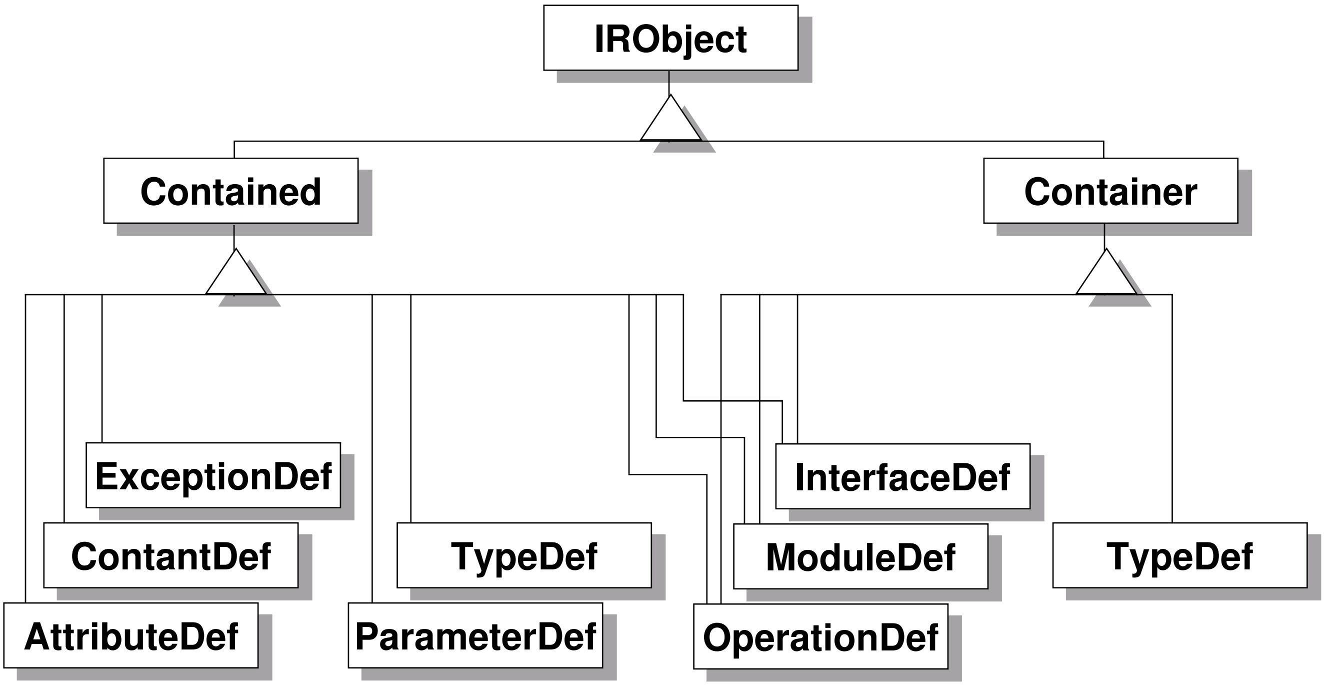


❏ Hierarchie der Komponenten einer IDL Datei bzw. des Interface Repositories

- Die Grafik gibt den strukturellen Aufbau des Repositories wieder. Die Kästchen entsprechen Klassen bzw. Objekten. Die Verbindungslinien sind Assoziationen.
- Ein Module kann beispielsweise Definitionen für Konstanten, Typen, Exceptions, Interfaces und wiederum weiterer Module enthalten.
- Die Struktur entspricht damit dem generellen Aufbau einer IDL-Beschreibung.

Interface Repository (3)

❏ Vererbungshierarchie der IDL Schnittstellen von IR Objekten



- Die Vererbungshierarchie führt zwei neue Klassen ein Contained und Container. Sie werden dazu benutzt um Objekte darzustellen, die in einem Container enthalten sein können bzw. einen solchen Container darstellen.
- Eine Moduldefinition beispielsweise ist sowohl Container – sie enthält Interfacedefinitionen etc. – als auch Element eines Containers, dem Repository oder eines anderen Moduls.

Inteface Repository (4)

❑ Standardtypen werden durch Type Codes repräsentiert

TypeCode

- Repäsentation von Basistypen
  – int, float, boolean
- Repräsentation von zusammengesetzten Standardtypen
  – union, struct, enum
- Repräsentation von Template Types und komplexen Deklaratoren
  – sequence, string, array

❑ Typecode repräsentiert den Typ bzw. die Struktur mittels IDL Schnittstelle

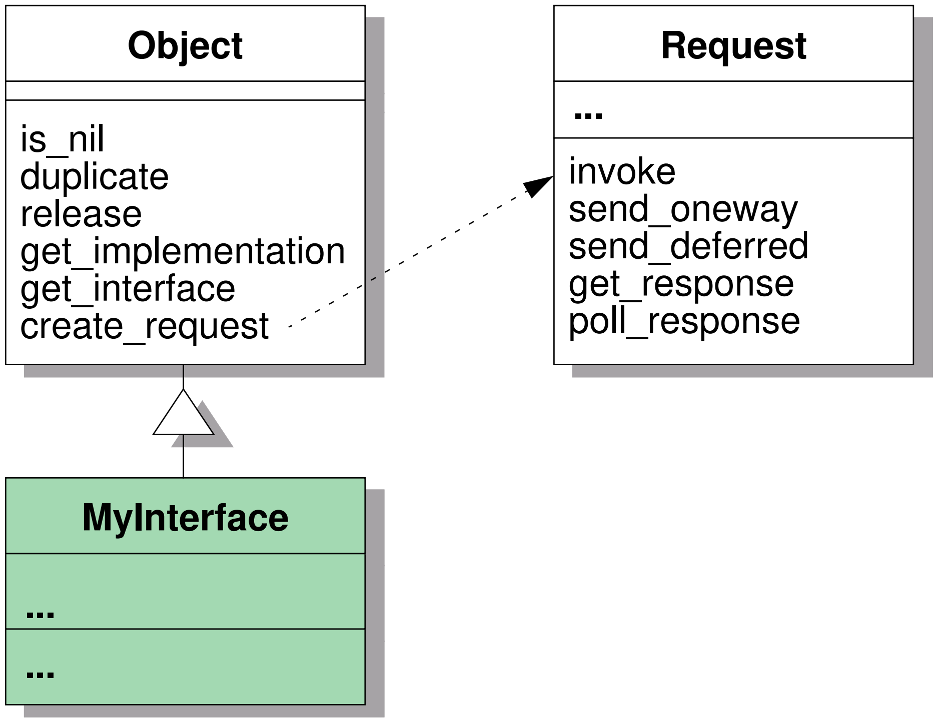
- Typecodes werden zur Typprüfung eingesetzt und ergänzen damit die beschreibenden Objekte aus dem Interface Repository. Alle Typen, die sich durch IDL beschreiben lassen werden durch ein entsprechendes Objekt aus dem Interface Repository selbst repräsentiert, alle Standardtypen durch ein geeignetes Typecode-Objekt.

DII – Dynamic Invocation Interface

- ❑ DII ermöglicht Aufruf von Objekten deren Schnittstelle erst dynamisch ermittelt werden können
  - Ermittlung erfolgt über das Interface Repository
- ❑ 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
      - Es wird der Aufruf generiert und auf das Ergebnis gewartet.
    - – asynchroner RPC
      - Es wird der Aufruf generiert. Der Aufrufer kann weiterarbeiten und sich das Ergebnis irgendwann später abholen.
    - – über Datagramm-Kommunikation (ohne Antwort)
      - Diese Aufrufart funktioniert nur bei Methoden, die keine Parameter zurückerwarten. Der Aufruf wird generiert; auf ein Ergebnis bzw. den Abschluß der Methodenausführung kann nicht gewartet werden. Hinzu kommt, daß der Aufruf eventuell über einen Datagramm versandt wird. Das bedeutet, daß nicht sichergestellt ist, ob der Aufruf überhaupt durchgeführt wird.

Dynamischer Aufruf

❑ IDL Definitionen für das DII



- Mit Hilfe der Methode create_request wird ein Requestobjekt erzeugt. Dabei werden diesem die bereits gesammelten Aufrufparameter übergeben.
- An dem Requestobjekt können dann mit verschiedenen Methoden, die entsprechenden Aufrufe am eigentlichen Objekt ausgeführt werden:

invoke	Aufruf als RPC
send_oneway	Aufruf als Datagramm ohne Antwort
send_deferred	asynchroner Aufruf
get_response	Warten auf Antwort
poll_response	Prüfen, ob Antwort bereits vorliegt

DSI – Dynamic Skeleton Interface

- ❑ DSI ermöglicht das Entgegennehmen von Methodenaufrufen für Objekte, deren Schnittstelle nur dynamisch ermittelbar ist, z. B. für
  - Bridges zu anderen ORBs
    - Bridges stellen eine Verbindung zu anderen ORBs her. Sie haben ein „Bein“ auf jeder Seite und bilden eine „Brücke“.
  - CORBA-gekapselte Datenbank
  - dynamisch erzeugte Objekte und Schnittstellen
- ❑ Anhand der Parameter wird das Objekt und dessen Signatur ermittelt
    - In der Implementierung wird der Aufruf an eine registrierte Callback-Funktion zugestellt, die dann das entsprechende Objekt aufrufen muß.

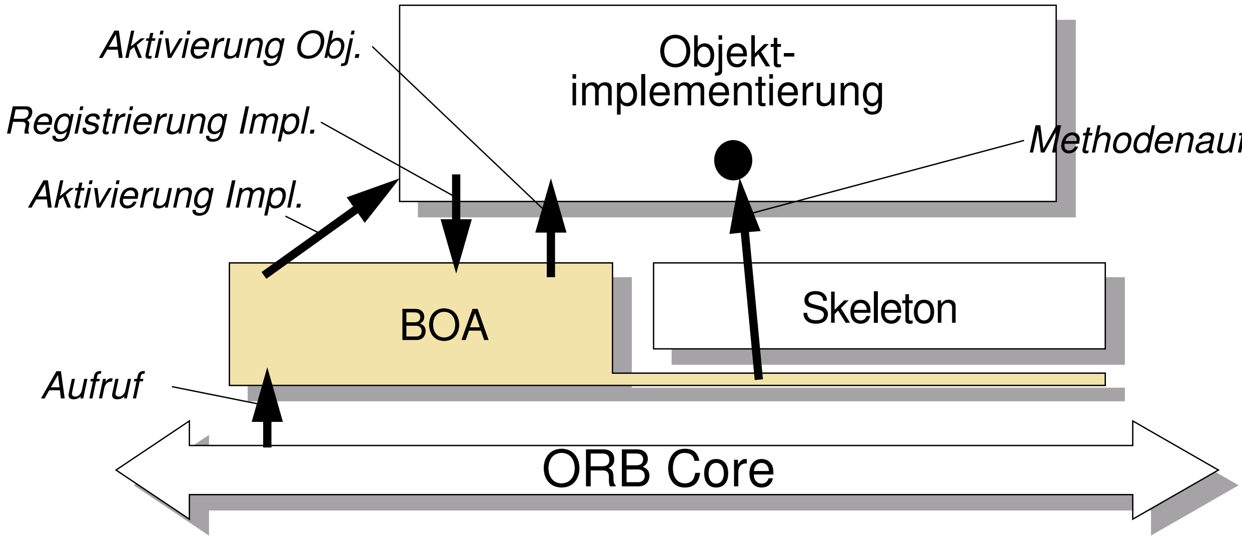
Beachte: DII und DSI sind jeweils von der Gegenseite nicht von statischen Stubs zu unterscheiden, d. h. voll interoperabel!

Object Adaptor

- ❏ Direkter Ansprechpartner eines CORBA Objekts zum Aufrufen von ORB Diensten
  - Generiert Objektreferenzen
  - Bildet Objektreferenzen auf Implementierungen ab
  - Bearbeitet einkommende Methodenaufrufe
  - Authentisierung des Aufrufers (Sicherheitsfunktionalität)
  - Aktiviert und deaktiviert Objekt bzw. dessen Implementierung
    - Ein Objekt ist eventuell zwar für das CORBA System präsent, aber noch nicht aktiv, so daß es direkt aufgerufen werden kann. Vor einem Aufruf wird der Object Adaptor dann das Objekt aktivieren.
  - Registriert Serverklassen im Implementation Repository
- ❏ CORBA definiert den Basic Object Adaptor
  - muß unterstützt werden
- ❏ Beispiel OODB Adaptor
  - Anbindung an eine objektorientierte Datenbank

BOA – Basic Object Adaptor

- ❑ Aktivierung und Deaktivierung von CORBA Servern
  - Implementierung eines Objekts ist nicht immer aktiv im Sinne des Betriebssystems



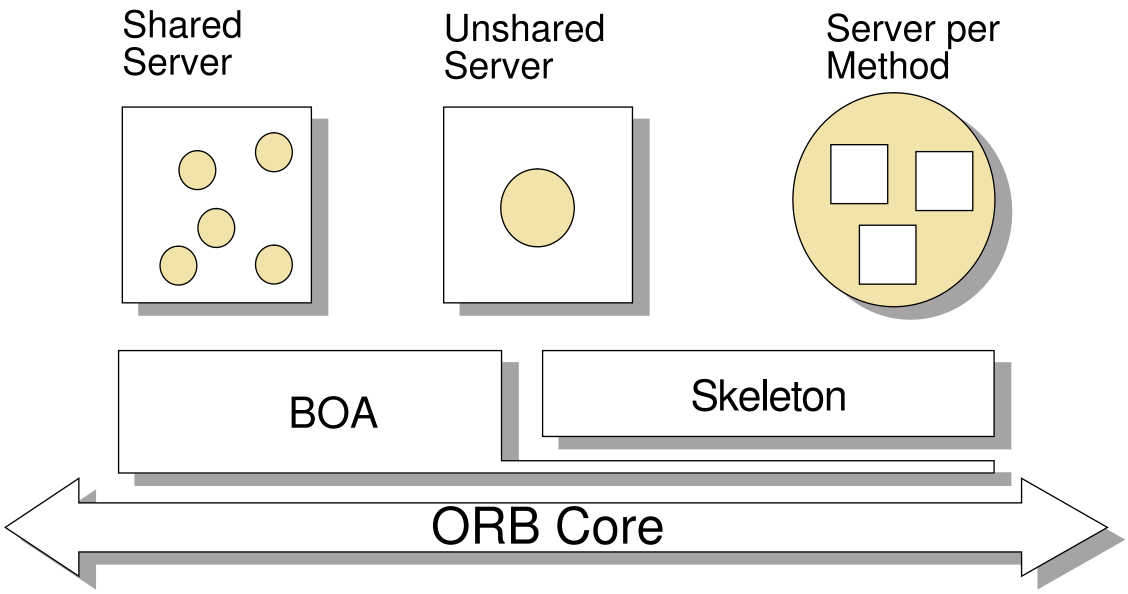
- Bei einem eingehenden Aufruf wird der BOA zunächst prüfen, ob eine Implementierung des Objekts bereits aktiv ist. Falls nicht wird diese aktiviert. Das bedeutet in einer CORBA Implementation beispielsweise das Starten eines entsprechenden Serverprozesses.
- Der Serverprozeß wird sich bei dem BOA registrieren und aktiv melden.
- Der BOA meldet dem Serverprozeß dann die Aktivierung des entsprechenden Objekts an und
- schickt dem Objekt mittels des statischen Sekeletons oder dem DSI Interface den Methodenaufruf.
- Ist das Objekt bereits aktiv, wird der Aufruf direkt über ein Skeleton oder das DSI durchgeführt.
- Der Serverprozeß kann sich beim BOA wieder abmelden/deaktivieren und somit aus dem Betriebssystem verschwinden.

BOA – Basic Object Adaptor (2)

- Aktivierungsstrategien
  - *Shared Server*
    ein Serverprozeß beherbergt alle Objekte einer Implementierung
  - *Persistent Server*
    Aktivierungsstrategie außerhalb des BOA (sonst wie Shared Server)
    - Der Persistent Server wird nicht vom BOA aktiviert, sondern ist ständig (persistent) vorhanden.
  - *Unshared Server*
    ein Serverprozeß pro Objektinstanz
  - *Server-per-Method*
    ein Serverprozeß pro Methodenaufruf

BOA – Basic Object Adaptor (3)

- ❑ Aktivierungsstragien im Bild



- Die Kreise stellen die Objekte dar, die Quadrate stellen Server dar
- Der Shared Server beherbergt eventuell mehrere Objekte einer Implementierung, der Unshared Server immer nur eins. Der Server-per-Method bearbeitet immer nur einen Methodenaufruf. Alle laufenden Server stellen mehr oder weniger ein Objekt dar.

Implementation Repository

- ❏ Datenbank, die Implementierungen speichert
  - Daten zur Lokalisierung und Aktivierung der Implementierung
  - Debugging- und Verwaltungsinformationen
  - etc.
- ❏ Abfrage der Datenbank zum
  - Implementierung der Methode *get_implementation* eines jeden Objekts
    - Dies entspricht der Methode get_interface beim Interface Repository.
- ❏ Schreiben der Datenbank
  - externe Tools
  - BOA schreibt beim Start eines Servers

- Das Implementation Repository ist implementationsabhängig und damit nicht über mehrere CORBA Implementierungen hinweg kompatibel. Vielfach wird es nicht implementiert.
- Meist nimmt es lediglich die Aktivierungsinformationen für den BOA auf.

Inter-ORB Kommunikation

- GIOP – General Inter-ORB Protocol
  - Basisprotokoll zur Interaktion zweier ORBs
  - Common Data Representation (CDR) konvertiert IDL konforme Parameter in einen seriellen Bytestrom
  - IIOP – Internet Inter-ORB Protocol
    – GIOP über TCP/IP (Implementierung vorgeschrieben)
  - andere Implementierungen von GIOP möglich
- ESIOP – Environment Specific Inter-ORB Protocols, z.B. DCE/ESIOP
  - Inter-ORB Protokoll auf der Basis von DCE RPC
- IOR – Interoperable Object Reference
  Interoperable Repräsentation einer Objektreferenz
  - String
  - verschiedene Profiles, u.a. IIOP
    - Eine Objektreferenz besteht eventuell aus verschiedenen Profiles, d.h. aus verschiedenen gleichwertigen Beschreibungen einer Objektreferenz. Diese Beschreibungen sind jedoch meist für verschiedene Protokolle. Eine CORBA Implementierung kann so beispielsweise ein lokales Profile benutzen, in dem eine proprietäre Protokolladresse für ein Objekt kodiert ist, und gleichzeitig ein IIOP Profile angeben, das beschreibt, wie das gleiche Objekt über IIOP von außen erreichbar ist.

CORBA Services

❏ Basisdienste eines verteilten Systems – Erweiterungen des ORB
  - Namensdienst
  - Life Cycle Service

Namensdienst

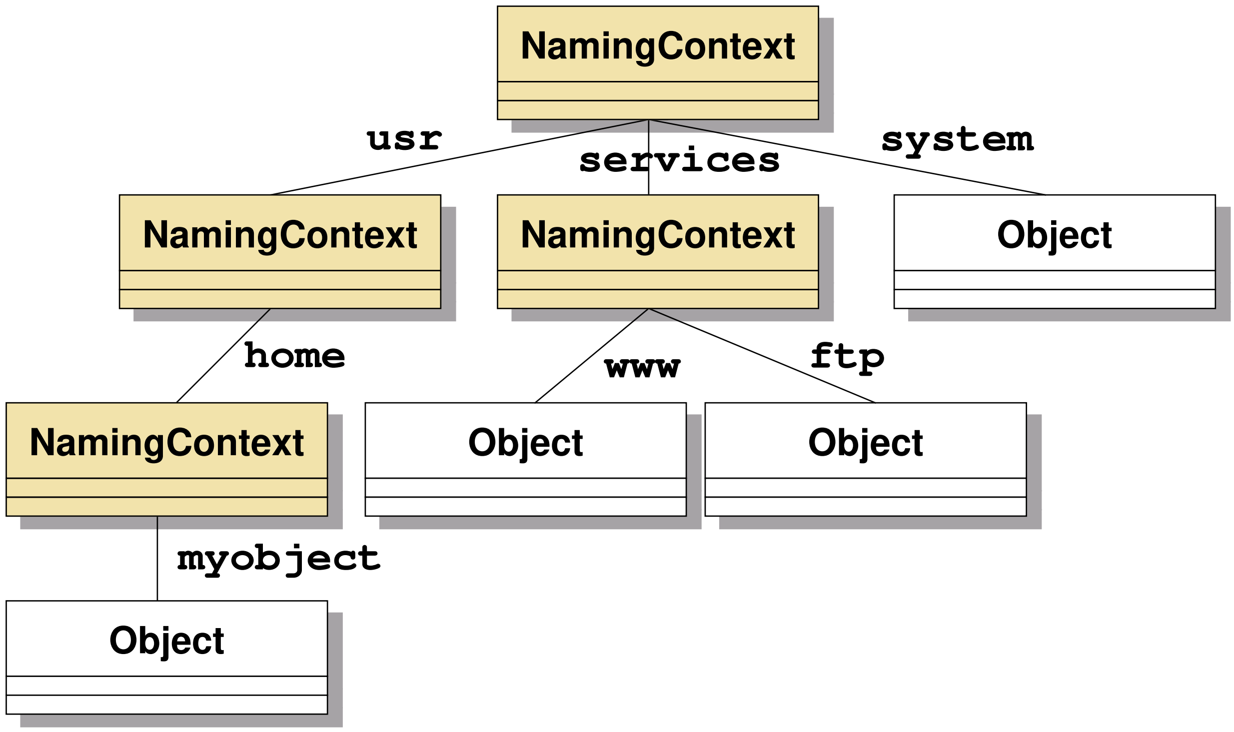
❏ CORBA definiert einen hierarchischen Namensdienst ähnlich dem UNIX Dateinamensdienst — Namensraum ist baumförmig

❏ Name besteht aus mehreren Komponenten

❏ Schnittstellen festgelegt durch IDL Beschreibungen

Namensdienst (2)

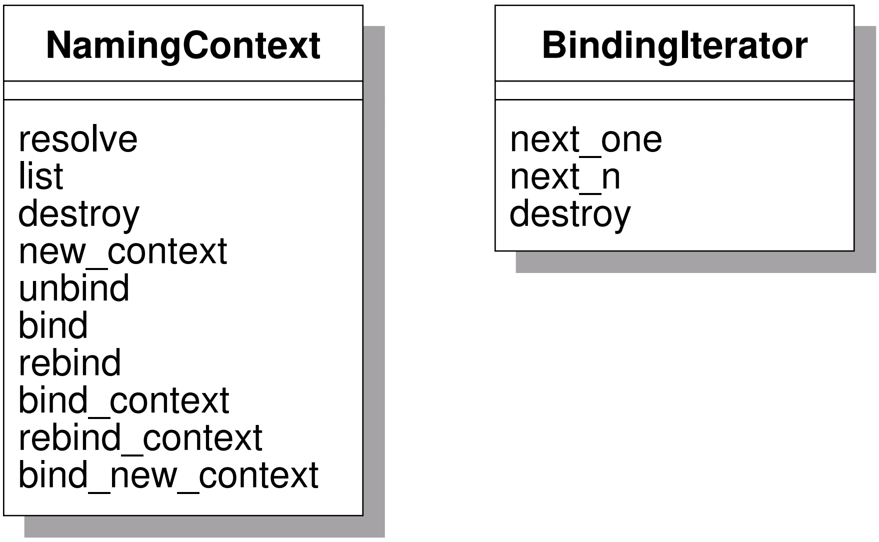
❑ Beispielbaum



- In diesem Baum gibt es beispielsweise den Namen
  < "usr" ; "home" ; "myobject" >
- Dieser Name besteht aus drei Komponenten. In unserer Notation wurden sie mit Strichpunkten abgetrennt. Es ist zu beachten, daß CORBA keine solche Notation definiert. Es ist lediglich festgelegt, daß der Name aus Komponenten besteht. Eine andere Notation könnte sein
  /usr/home/myobject
  wie aus UNIX bekannt.
- Bei der Auflösung dieses Namens erlangt man die Objektreferenz auf das entsprechende Objekt.

Namensdienst (3)

- ❑ Kontext- und Iteratorschnittstelle



- Die Klasse NamingContext stellt soetwas wie ein Verzeichnis oder Directory dar. Man kann Namenskomponenten definieren und diese entweder mit einem Objekt oder einem anderen Kontext verbinden. Im Falle eines Kontexts entsteht der besagte Baum.
- Beim Auflisten eines Kontexts kann eine maximale Anzahl von Namenseinträgen angegeben werden, die der Aufrufer haben möchte. Überbleibende Einträge werden dann in einen BindingIterator gepackt, der zusätzlich zurückgegeben wird. Mit ihm lassen sich dann die folgenden Einträge zugreifen.

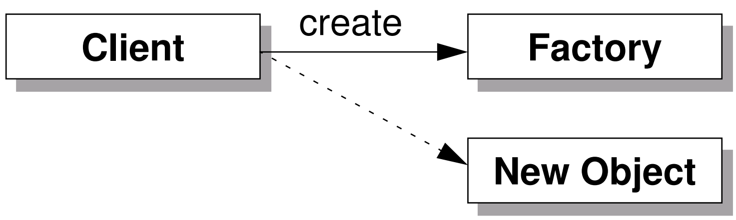
Life Cycle Service

❏ Lebenszyklus eines Objekts
  - Erzeugung
  - Kopieren, Verlagern
  - Löschen

❏ einheitliche Schnittstelle für Operationen während des Lebenszyklus

Modell des Lebenszyklus

❏ Erzeugung:

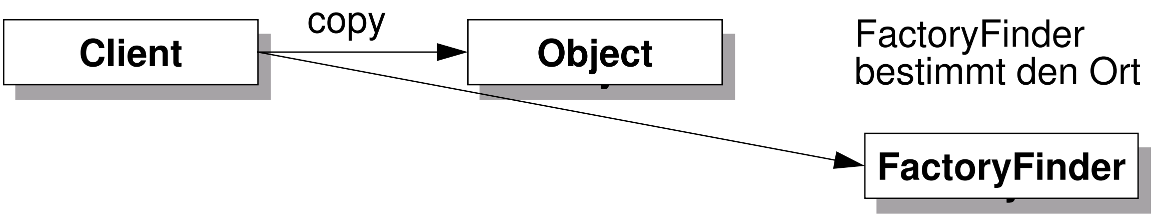


Aufruf einer Erzeugunsmethode

- Die Factory wird aufgerufen, erzeugt ein neues Objekt und gibt dessen Objektreferenz an den Aufrufer zurück.
- Das Protokoll bzw. das Interface der Factory ist in der Regel abhängig vom zu erzeugenden Objekt und nicht festgelegt.

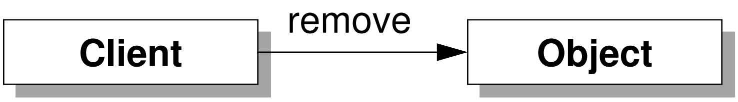
Life Cycle Service (2)

❏ Kopieren und Verlagern:



- Der FactoryFinder bestimmt den Ort und oder die Region, in der die Kopie erzeugt wird bzw. in die das Objekt verlagert werden soll.
- Orte könnten sein: ein bestimmter Rechner, ein bestimmte Rechnergruppe etc.

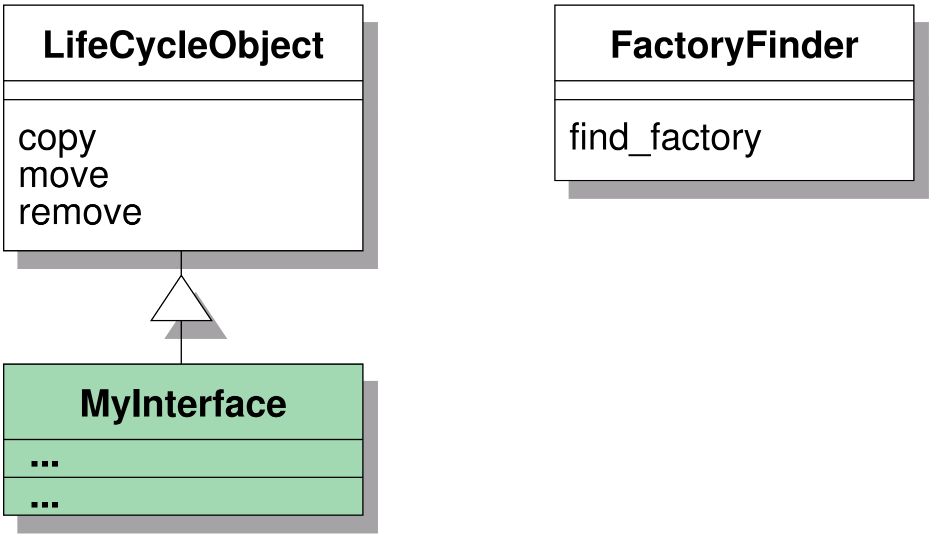
❏ Löschen:



- Mit dem Aufruf von remove löscht sich das Objekt.

Life Cycle Service (3)

- ❏ Objekte müssen das Interface LifeCycleObject implementieren



- ❏ copy and move benötigen ein FactoryFinder Objekt zur Bestimmung des Ortes
- ❏ remove löscht ein Objekt

Life Cycle Service (4)

- ❑ Implementierung am Beispiel copy
    - Client ruft copy Methode auf und übergibt einen FactoryFinder
    - Die copy Methode stellt entweder der Programmierer des Objekts oder die CORBA Implementierung bereit
    - Die copy Methode ruft den FactoryFinder auf und sucht eine passende Factory aus und erzeugt mit ihrer Hilfe ein neues Objekt.
    - Das neue Objekt wird initialisiert mit den Daten des bestehenden Objekts.
- ❑ Nach außen klare Schnittstelle
- ❑ Nach innen offen und implementationsabhängig, z.B. Protokoll zwischen copy Methode und Factory
- ❑ In CORBA 2.0 Berücksichtigung von Teil-Ganze-Beziehungen und ähnlichen (Deep Copy und Shallow Copy)
    - Die Beziehungen zwischen Objekten können registriert werden. Bei einer Kopier- oder Löscheoperation wird dann ein ganzer Objektgraph kopiert oder gelöscht.