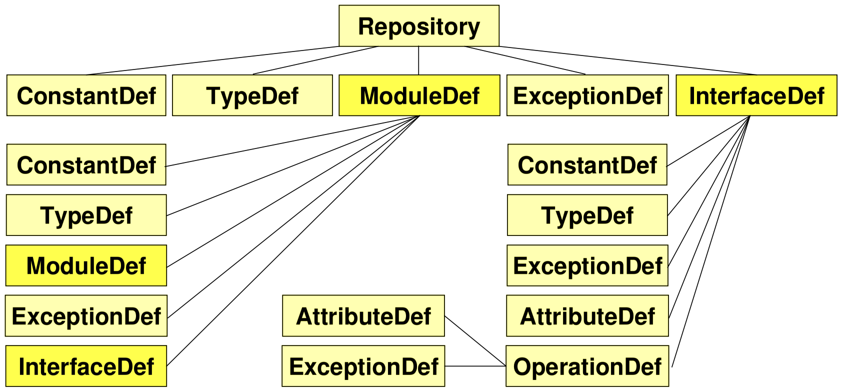
3 Interface Repository (2)

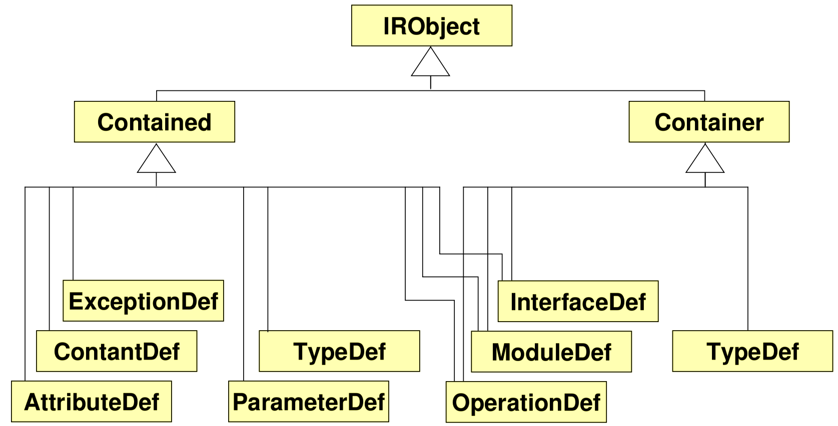
■ Gespeicherte Informationen / Aufbau von IDL Dateien



- ◆ Komponenten einer IDL Datei werden als Objekte die in IDL spezifiziert sind abgelegt
- ◆ Hierarchie der Komponenten 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.

3 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.

3 Interface 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
- ◆ Repräsentation von IDL basierten Objet Typen (via Interface Repository ID)

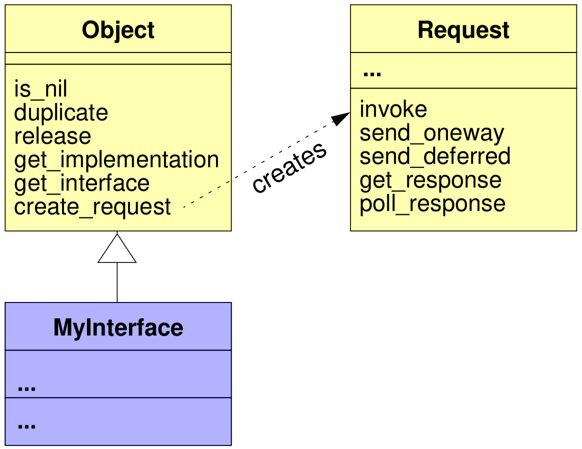
- Typecode repräsentiert den Typ bzw. die Struktur als Objekt mit IDL-Schnittstelle
  - Typecodes werden zur Typprüfung eingesetzt und ergänzen damit die beschreibenden Objekte aus dem Interface Repository. Alle Typen, die sich durch IDL beschreiben lassen werden durch ein entsprechendes Objekt aus dem Interface-Repository selbst repräsentiert, alle Standardtypen durch ein geeignetes Typecode-Objekt.
  - Methode zum Vergleich von Typecodes
  - Methode zur Abfrage einer Typbeschreibung

4 Dynamic Invocation Interface (DII)

- 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-dienst versandt wird. Das bedeutet, daß nicht sichergestellt ist, ob der Aufruf überhaupt durchgeführt wird.

4 Dynamic Invocation Interface (2)

- 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

5 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
    - 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!

6 Object Adaptor

- Lokaler Repräsentant des ORB-Dienstes, direkter Ansprechpartner eines CORBA Objekts
  - ◆ Generiert Objektreferenzen (für neue Objekte)
  - ◆ Bildet Objektreferenzen auf Implementierungen ab
  - ◆ Bearbeitet ankommende 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 Portable Object Adaptor
  - ◆ Basis-Funktionalität, muss unterstützt werden
- weiteres Beispiel: OODB Adaptor
  - ◆ Anbindung einer objektorientierten Datenbank an CORBA
  - ◆ Der OODB Adaptor repräsentiert alle Objekte in der Datenbank als CORBA-Objekte und vermittelt Aufrufe

7 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
  - ◆ Object Adaptor 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 POA auf.

8 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.

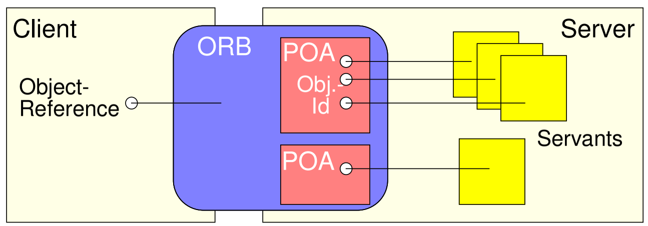
D.11 Portable Object Adaptor (POA)

1 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)
- Mehrere POA-Instanzen (mit unterschiedlichen Strategien) innerhalb eines Servers möglich

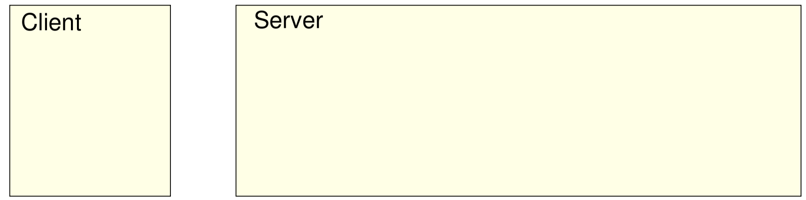
2 Terminologie

- ***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



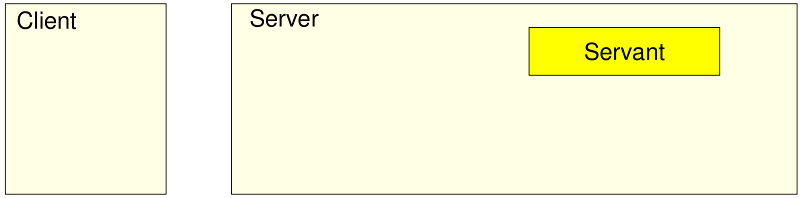
3 Erzeugung eines CORBA-Objekts

- Anwendung erzeugt ein Objekt



3 Erzeugung eines CORBA-Objekts

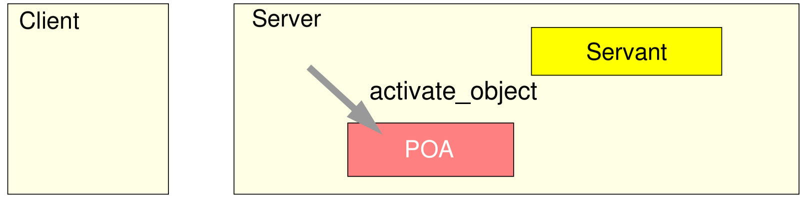
- Anwendung erzeugt ein Objekt



- ◆ 1. Servant wird erzeugt

3 Erzeugung eines CORBA-Objekts

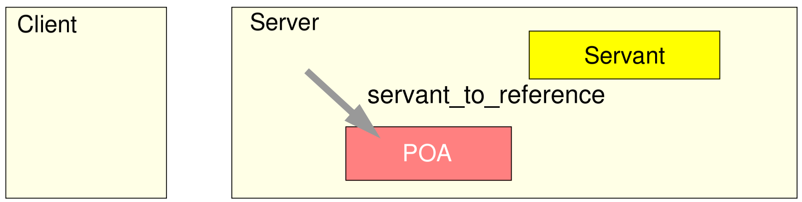
- Anwendung erzeugt ein Objekt



- ◆ 1. Servant wird erzeugt
- ◆ 2. Servant wird mit Object-Id versehen

3 Erzeugung eines CORBA-Objekts

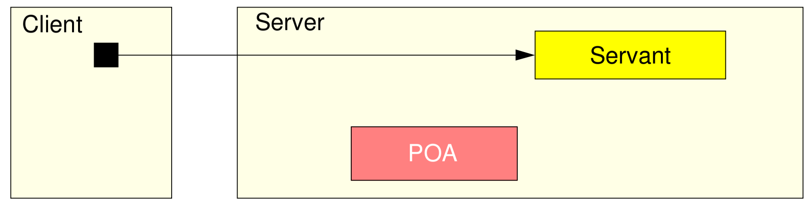
- Anwendung erzeugt ein Objekt



- 1. Servant wird erzeugt
- 2. Servant erhält eine Object-Id
- 3. Servant wird mit Object-Reference versehen (dadurch wird er zum CORBA-Objekt)

3 Erzeugung eines CORBA-Objekts

- Anwendung erzeugt ein Objekt



- 1. Servant wird erzeugt
- 2. Servant erhält eine Object-Id
- 3. Servant wird mit Object-Reference versehen (dadurch wird er zum CORBA-Objekt)
- 4. Reference kann nun über Name Service verfügbar gemacht werden oder als Ergebnis eines Aufrufs an einen Client übergeben werden. Dort wird dann ein Client-Stub zur Weiterleitung der Aufrufe an den Server erzeugt.

➥ eine Object-Reference beim Client steht für ein bestimmtes CORBA-Objekt — nicht unbedingt für einen bestimmten Servant!

4 Alternativen zur Aktivierung von CORBA-Objekten

- Implizite Aktivierung
  - POA erzeugt automatisch eine Object -Reference wenn ein (nicht aktivierter) Servant als Parameter oder Ergebnis "nach aussen" übergeben wird
- Aktivierung "on demand"
  - Object-Reference wird erzeugt ohne dass ein Servant existiert
    - damit entsteht ein abstraktes CORBA-Objekt, zunächst ohne Implementierung
  - Verschiedene Möglichkeiten, ankommende Aufruf zu bearbeiten:
    - Ein registrierter "Default Servant" bearbeitet den Aufruf (z. B. wenn Datenbank-Einträge als CORBA-Objekte exportiert werden)
    - Ein "Servant Manager" (vom Benutzer zu implementieren) liefert einen Servant (der Servant Manager kann den Servant ggf. dynamisch erzeugen)

5 Deaktivierung und Aktivierung

- Objekte können eine POA-Instanz überleben: Persistente Objekte
  - Servant kann dekativiert werden
  - POA kann deaktiviert werden
  - Server kann gestoppt werden
- POA kooperiert mit "Location Forwarding Service" und dem Implementation Repository
  - Bei der Deaktivierung werden die für die Aktivierung notwendigen Informationen im Implementation Repository aufgehoben
  - Aufrufe an deaktivierte Objekte gehen zunächst an den Forwarding Service. Dieser startet einen neuen Server.
  - Der Server erhält den weitergeleiteten Aufruf und aktiviert einen POA.
  - Der POA aktiviert einen Servant und übergibt ihm den Aufruf.

6 POA-Strategien (Policies)

- Eine POA-Strategie ist jeweils einer POA-Instanz zugeordnet
  - Innerhalb eines Servers kann es mehrere POA-Instanzen, jeweils mit einer anderen Strategie geben.
- Aktivierungs-Methode ist in solch einer Strategie festgelegt
  - Servant retention policy
  - Request processing policy
  - Implicit activation policy
- Weitere Strategien:
  - Thread policy: multi-threaded oder single-threaded
  - ID uniqueness policy: ob eindeutige IDs benutzt werden
  - ID assignment policy: IDs werden durch Benutzer bzw. System erzeugt
  - Object lifespan policy: transient bzw. persistent

D.12 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`	

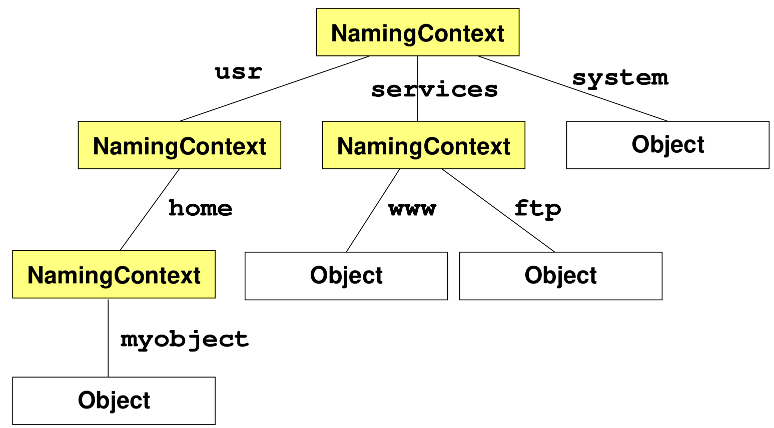
- Services sind über IDL-Schnittstellen aufrufbar
  - Keine zusätzlichen Konstrukte erforderlich; Methodenaufruf reicht aus

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" >
    (UNIX: "/usr/home/myobject")
  - ◆ Schnittstellen festgelegt durch IDL-Beschreibungen

1 Namensdienst (2)

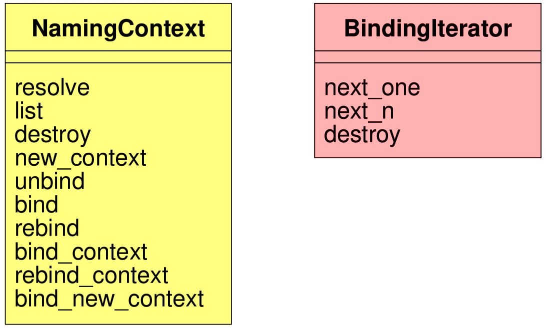
- Beispielbaum



- 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.
- Die Referenz auf den root-NamingContext bekommt man über eine Abfrage beim ORB
  `orb.resolve_initial_references("NameService");`

1 Namensdienst (3)

- Kontext- und Iteratorschnittstelle



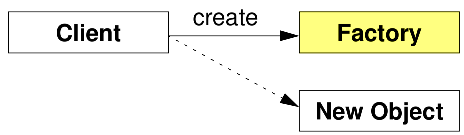
- Die Klasse NamingContext stellt so etwas wie ein Verzeichnis oder Directory dar. Man kann Namenskomponenten definieren und diese entweder mit einem Objekt oder einem anderen Kontext verbinden. Im Falle eines Kontexts entsteht der besagte Baum.
- Beim Auflisten eines Kontexts kann eine maximale Anzahl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

2 Life Cycle Service

- Lebenszyklus eines Objekts
  - Erzeugung
  - Kopieren, Verlagern
  - Löschen
- Life Cycle Service definiert eine gemeinsame Schnittstelle für Operationen während des Lebenszyklus

★ Modell des Lebenszyklus

- Erzeugung eines Objekts:

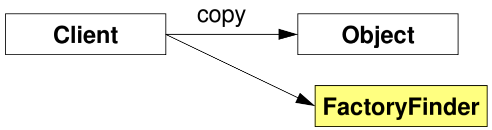


Aufruf einer create-Methode an einem Factory-Objekt

- 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.

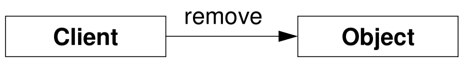
2 Life Cycle Service (2)

◆ Kopieren und Verlagern:



- Client kennt einen FactoryFinder
- 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, eine bestimmte Rechnergruppe etc.

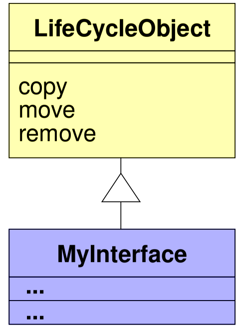
◆ Löschen:

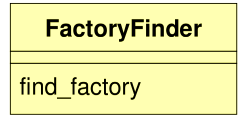


- Mit dem Aufruf von remove löscht sich das Objekt.

2 Life Cycle Service (3)

■ 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

2 Life Cycle Service (4)

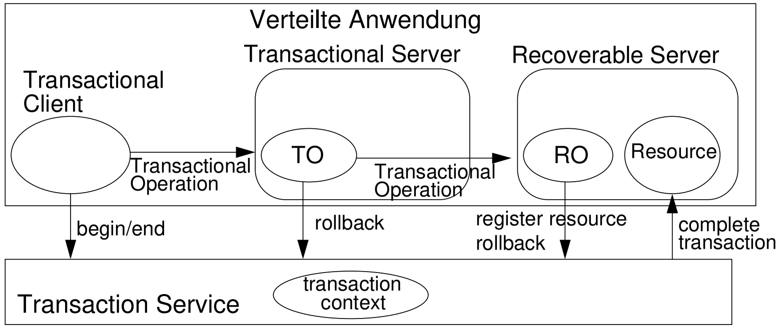
- Implementierung am Beispiel copy
  - ◆ Client ruft copy-Methode auf und übergibt einen FactoryFinder
  - ◆ Die copy-Methode stellt entweder der Programmierer des Objekts oder die CORBA-Implementierung bereit
  - ◆ Die copy-Methode ruft den FactoryFinder auf, sucht eine passende Factory aus und erzeugt mit ihrer Hilfe ein neues Objekt.
  - ◆ Das neue Objekt wird mit den Daten des bestehenden Objekts initialisiert.
- ▲ 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.

3 Object Transaction Service (OTS)

- Transaktionen: "ACID":
  - atomic (alles oder nichts)
  - consistent (Neuer Zustand erfüllt Konsistenzbedingungen)
  - isolated (Isolierung: keine Zwischenzustände sichtbar)
  - durable (Persistenz)
- Eine Transaktion => mehrere Objekte, mehrere Requests:
  Bindung an einen *"Transaction context"*
  - ◆ wird normalerweise *implizit* an alle Objekte weitergereicht
  - ◆ auch explizite Weitergabe durch Client möglich (Spez. in IDL)
- Typischer Ablauf:
  - ◆ Beginn der Transaktion erzeugt *Transaction context* (an den Client-Thread gebunden)
  - ◆ Ausführung von Methoden (implizit an die Transaktion gebunden)
  - ◆ Schliesslich: Client beendet die Transaktion (commit/roll back)

3 Object Transaction Service (OTS) (2)

- Bestandteile einer Anwendung, die vom OTS unterstützt wird:
  - Transactional Client
  - Transactional Objects (TO)
  - Recoverable Objects (RO)
  - Transactional Servers
  - Recoverable Servers



3 Object Transaction Service (OTS) (3)

- Zugriff auf Transaction Service über **`Current`**-Objekt
  - **`orb.resolve_initial_reference("TransactionCurrent");`**

```
interface Current : CORBA::Current {
   void begin() raises(SubtransactionsUnavailable);
   void commit(in boolean report_heuristics)
      raises(NoTransaction,HeuristicMixed,HeuristicHazard);
   void rollback() raises(NoTransaction);
   void rollback_only() raises(NoTransaction);

   Status get_status();
   string get_transaction_name();
   void set_timeout(in unsigned long seconds);

   Control get_control();
   Control suspend();
   void resume(in Control which) raises(InvalidControl);
};
```

3 Object Transaction Service (OTS) (4)

- *Transaction Context*: `Control`-Objekt

```
interface TransactionFactory {
   Control create(in unsigned long time_out);
   Control recreate(in PropagationContext ctx);
};

interface Control {
   Terminator get_terminator() raises(Unavailable);
   Coordinator get_coordinator() raises(Unavailable);
};

interface Terminator {
   void commit(in boolean report_heuristics) raises(...);
   void rollback();
};
```


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