

CORBA als Ordnung in verteilten Anwendungen

«Java Programming with CORBA» von Brose, ...
«Essentials CORBA-Buch» von Andreas Sayegh
und div. «Internetquellen»

Anton Böhm

anton.boehm@itServe.ch

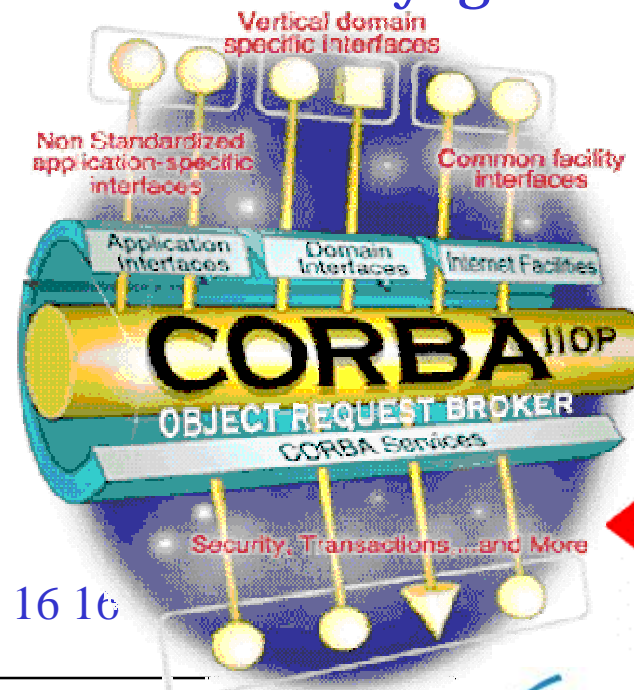
itServe AG

P.O.Box

Länggass-Str.26

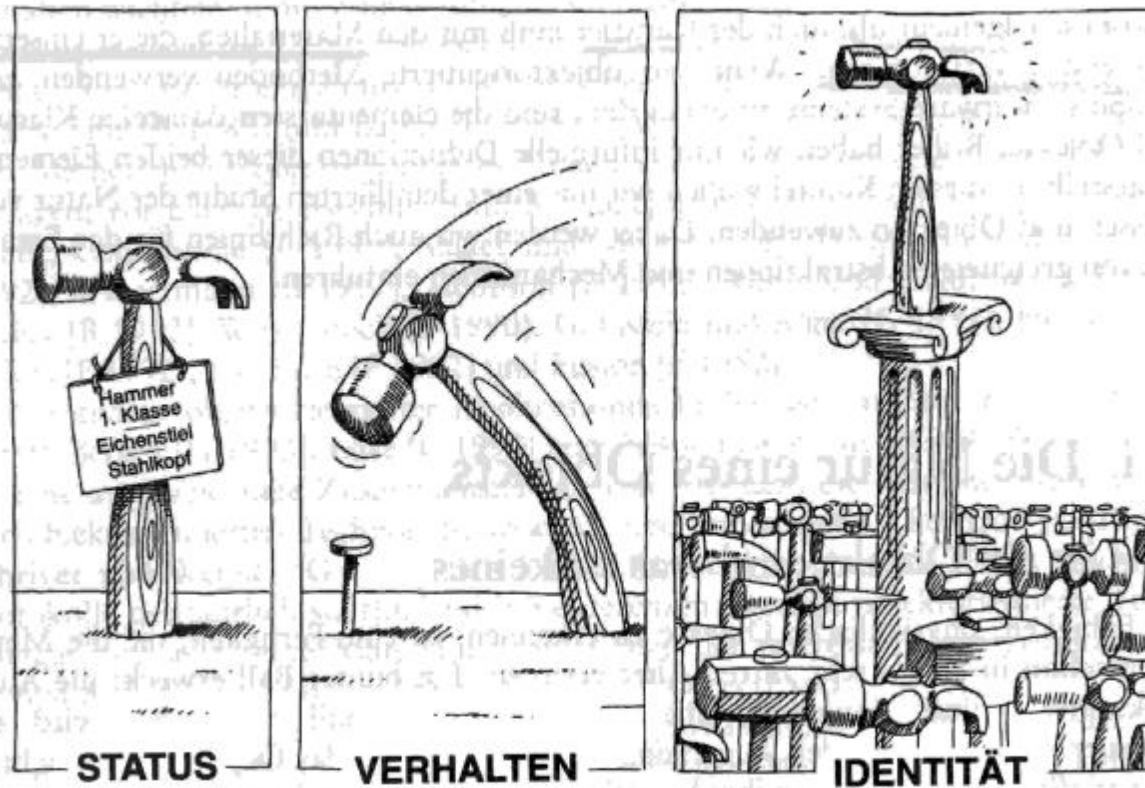
CH-3000 Bern 9

Tel. +41 31 305 16 16



itServe
tomorrow's technology now

Objekt



Ein Objekt hat einen Status, weist ein wohldefiniertes Verhalten auf und besitzt eine eindeutige Identität

CORBA Objektmodell

Objekt = eindeutig identifizierbare Entität

Typen (eingrenzen der Datenwerte)

einfache (float, ...) u. konstruierte (struct, ...) Typen

Objektreferenz zur Identifikation innerhalb ORB-Domäne

Interoperable Objektreferenz (IOR) CORBA-Definiert

Value, d.h. ORB kopiert Objekt vollständig (CORBA 2.3)
vom Client zum Server oder umgekehrt

Protokoll (Interface; «mehr» als die Summe aller Operationen)

Schnittstellen (static und dynamic Interfaces; IDL)

Operationen (Implementiert durch Methoden)

Attribute

Wichtige Begriffe

«remote object»

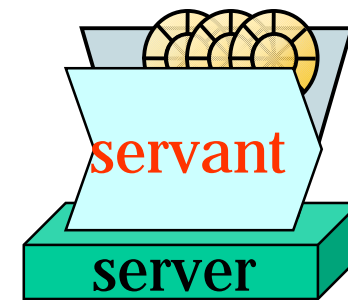
Entferntes Objekt; Bestandteil eines *Dienstes* (Server)
inklusive «Ablaufumgebung» (Prozess)

«servant»

Implementierung einer in IDL definierten Schnittstelle;
reale Ausprägung eines CORBA-Objekts

«server»

Laufzeit-/ Ablauf-umgebung
oder Prozess, in dem eine Servant existiert



(CORBA 2.2)

Object Request Broker (ORB)

⇒ «Herz» der Kommunikation

Findet Objektkomplementierungen

Vermittelt, sendet und empfängt Daten

ORB-Interface für Informationen und Aktionen
(Nutzung durch Client und Server)

⇒ ORB Domäne

Gesamtheit aller Objektkompl. im Zugriff

Innerhalb einer Domäne freie Realisierung

Domänen-Interaktionen klar definiert (GIOP)

«der ORB»

⇒ Ein ORB ist als Logik (Konzept) zu verstehen, die über alle Objekte einer ORB-Domäne verteilt ist. Es ist weder vorgeschrieben noch wahrscheinlich, dass er nur aus einer Komponente besteht:

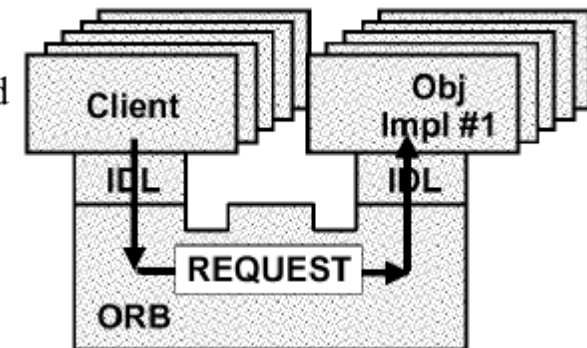


Er ist «abstrakt»

«der ORB» (2)

Möglichkeiten der ORB Realisierung

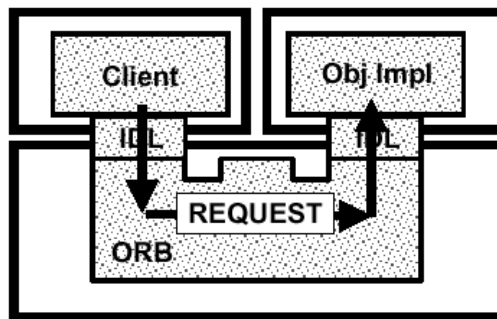
- ✗ Client- und Implementierungs-resident
 - ✗ ORB ist implementiert als Bibliothek
 - ✗ ORB ist resident in den Klienten und in der Objektimplementierungen
- ✗ Library-resident (single-process resident)
 - ✗ ORB und Objektimplementierungen sind als Bibliothek implementiert und Klienten-resident.
- ✗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



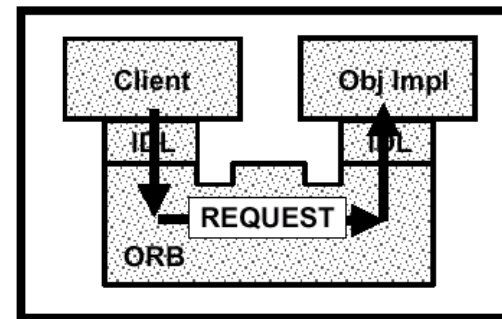
Michael Weber, Verteilte Systeme WS 1999/2000, Universität Ulm

«der ORB» (3)

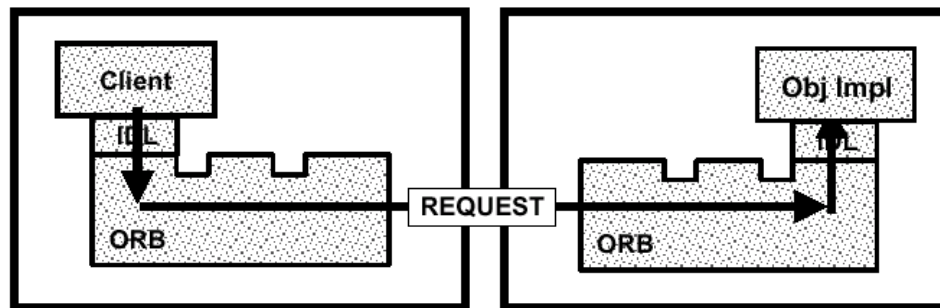
verschiedene ORB Typen



Server or Operating-
System Based



Single-Process
Library Resident



Client &
Implementation
Resident

Michael Weber, Verteilte Systeme WS 1999/2000, Universität Ulm

50

ORB Operationen

⇒ OMG definiert drei Bereiche

Operationen für alle ORB's gleich

Spezifisch Operationen für Bestimmte Arten von Objekten

Spezifisch Operationen für Bestimmte Arten von Implementierungen

⇒ IDL (*separater Kursteil*)

Objekt - Interface

«Pseudo» Objekte mit Interface (//PIDL)

«Definition ≠ Spezifikation»

In IDL werden Schnittstellen *definiert*, aber Objekte werden durch Schnittstellen *spezifiziert*.

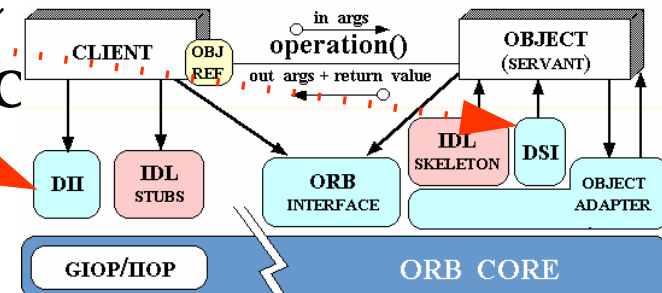
Dynamic Interfaces (DII, DSI)

Dynamic Skeleton Interface Dynamic Invocation Interface

⇒ «Warum?»

2 rfp (request for proposal)

Keine vorkompilierten Stubs/Skeletons



⇒ «Wozu?»

rfp
1 *sun* = *stat.*
2 *dec* = *dyn.*
-> Common
Object Request Broker
Architecture

Einsatz (Usage Pattern)

Client ruft ServerObjekt häufig	Statische Lösung (einfach und stabil !!)
ServerObjekt ist stabil	
Client ruft ServerObjekt selten	DII Lösung
Client entdeckt ServerObjekt zur Laufzeit	DII Lösung
Generischen Bridges	DSI Lösung
Interpreter und Script-Sprachen	DSI Lösung

Aufruftechnik

Dynamic Invocation Interface (DII)

⇒ Static Invocation Interface (SII)

Durch IDL-Compiler generiert Proxy-Stub

⇒ Dynamic Invocation Interface (DII)

Steht parallel zur Verfügung

Gleiches Protokoll zur Kommunikation

SII verwendet oft intern das DII

Wesentlicher Teil: *CORBA::Request*

- «oneway» Operation
- synchrone Kommunikation durch: *invoke(..)*;
- asynchrone Kommunikation durch: *send(..)*; *get_response(..)*



DII

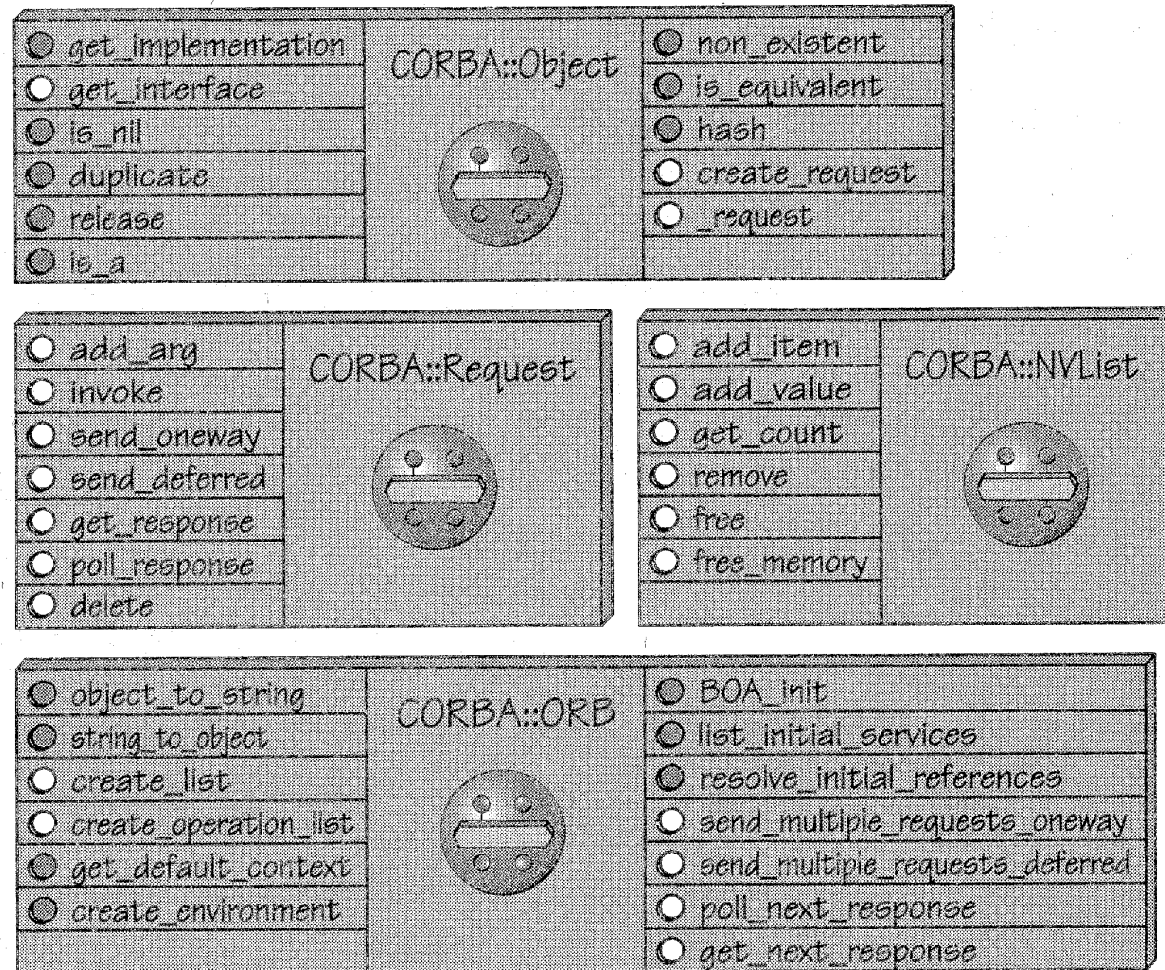


Bild 5.2. Die Schnittstellen der dynamischen Aufrufe

DII

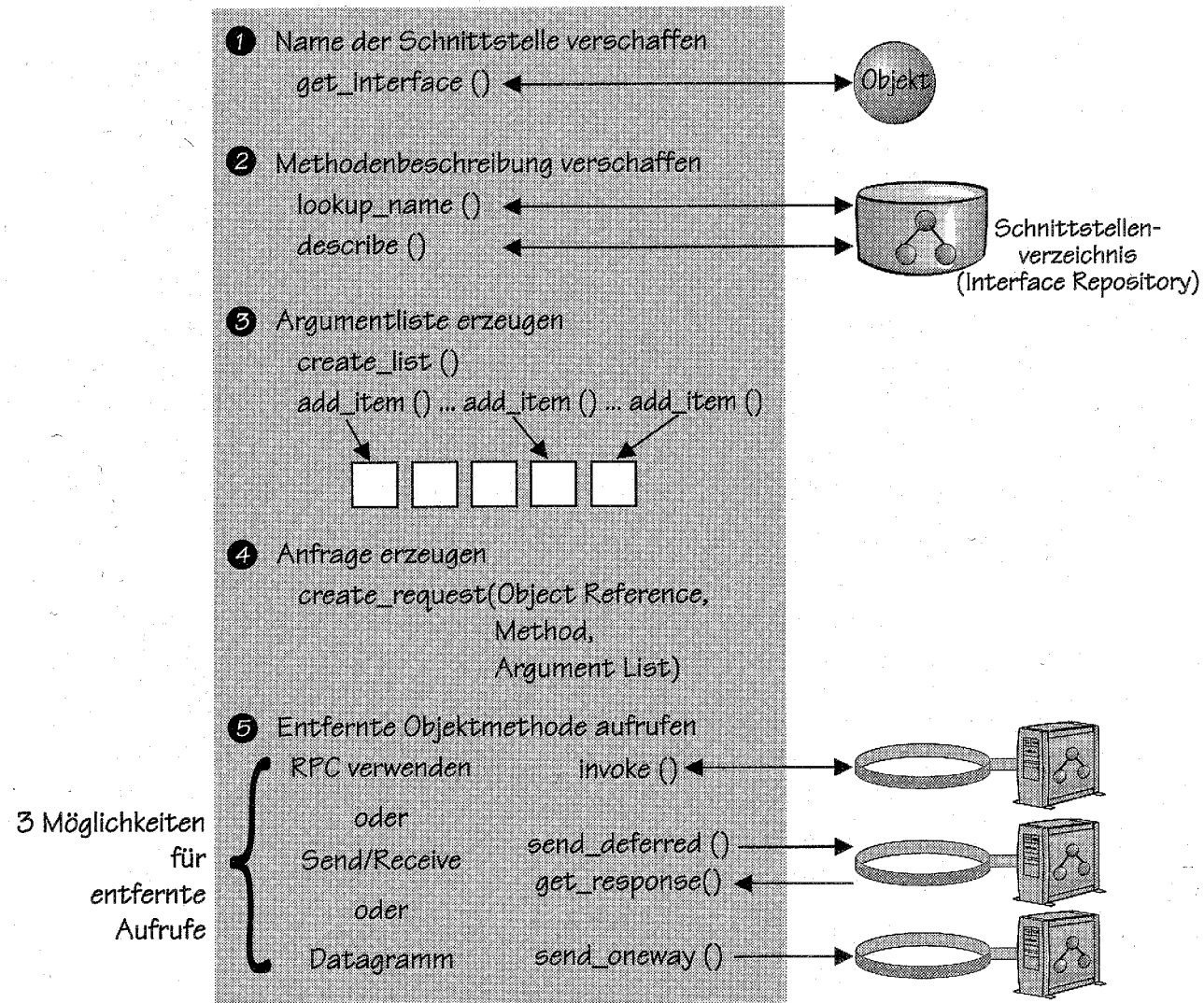


Bild 5.1. Der Prozeß eines dynamischen Aufrufs mit CORBA

Dynamic Skeleton Interface (DSI)

⇒ Konzept:

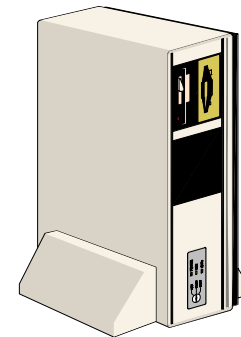
Server erhält Operationsname und Parameter
DSI ist nicht notwendig bei Client-DII Nutzung

⇒ Ablauf

Anmeldung einer Instanz der Klasse
DynamicImplementation beim ORB

Empfängt der ORB eine Anfrage, kann er von dieser
Instanz die Operation invoke() aufrufen

Die Instanz erhält ein Pseudo-Objekt ServerRequest



Interface-Repository (IR)

- ⇒ IR dient zur persistenten Erfassung von Schnittstellen
- ⇒ IDL zu IR;
d.h. IDL-Elemente werden durch Objekte repräsentiert

- ⇒ Wozu dient ein IR?

Typcheck für Methodensignatur

Hilfe beim Verbinden der ORBs

Metadaten für Clients u. Tools (Methodenaufruf «on-the-fly»)

Selbstbeschreibende Objekte

```

module <identifier> ←
(
  <type declarations>;
  <constant declarations>;
  <exception declarations>;

  interface <identifier> [[:<inheritance>]] ←
  (
    <type declarations>;
    <constant declarations>;
    <attribute declarations>;
    <exception declarations>;

    [<op type>]<identifier>(<parameters>)←
    [<raises exception>]<context>;

```

Definiert einen Namenskontext

Definiert eine CORBA-Klasse

Definiert eine Methode

Repository-ID mit « a : b: c »

a = Zeichenkette «IDL»
 b = Liste von Bezeichnern getrennt durch «/»
 c = Hauptversion. Unterversion
 z.B. IDL:itServe/corba/gugus:1.0

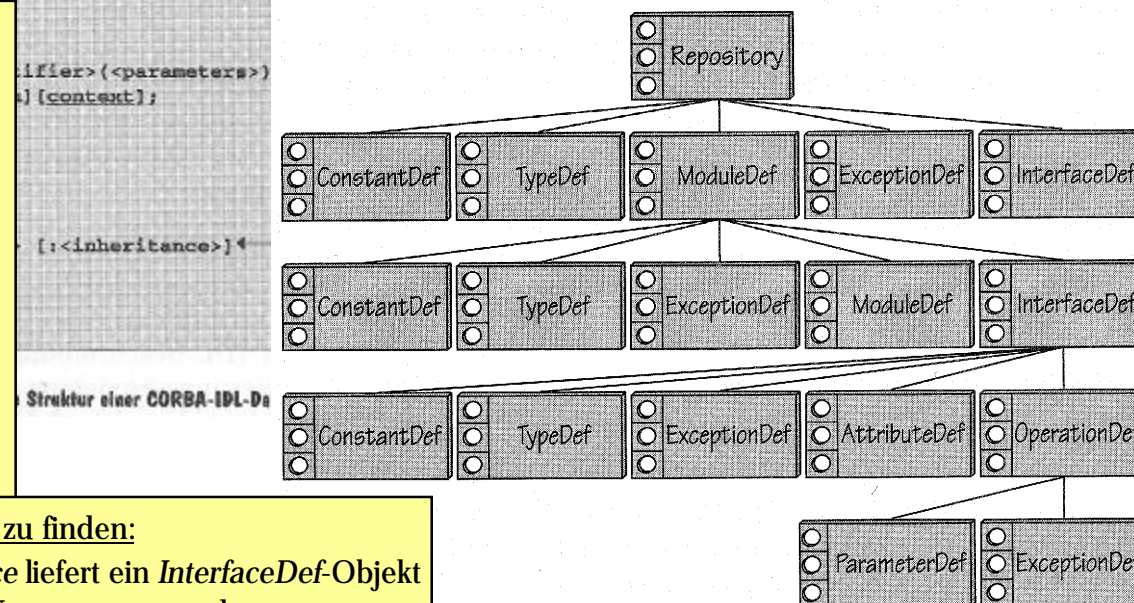
8 IDL Strukturen:

- ModulDef
- InterfaceDef
- OperationDef
- ParameterDef
- AttributeDef
- ConstantDef
- ExceptionDef
- TypeDef

und ein Root Repository

3 Varianten Objekt-IF zu finden:

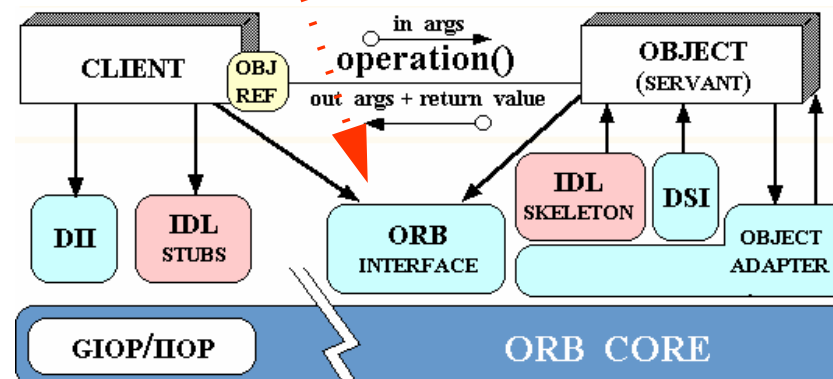
- *Object::get_interface* liefert ein *InterfaceDef*-Objekt
- Navigieren durch Namensraum und dann *InterfaceDef::describe_interface*
- Repository-ID mit *Repository::lookup_id*



Die Container-Hierarchie für die Klassen des Schnittstellenverzeichnisses

ORB-Interface

- ⇒ Policies (in CORBA 2.2)
«Verfahren/Verhalten» Parameter für ORB
z.B. request parallel abarbeiten, ...
- ⇒ CORBA::ORB Interface
- ⇒ CORBA::Object Interface



CORBA::ORB Interface

Operation	Beschreibung
<pre>ORB ORB_init (inout arg_list argv, in ORBid orb_identifizier);</pre>	initialisiert den ORB und liefert eine program- miersprachliche Referenz zurück. Der Parameter argv kann zur Übergabe von Kommandozeilen- parametern benutzt werden; die Funktionalität des orb_identifizier wird von CORBA nicht fest- gelegt.
<pre>Object resolve_initial_references (in ObjectId identifizier) raises (InvalidName);</pre>	liefert die Objektreferenz eines Standarddienstes des ORBs, dessen Name als identifizier angege- ben wurde. Gemäß CORBA 2.3 können fol- gende ORB-Objekte ermittelt werden: RootPOA, POACurrent, InterfaceRepository, NameService, TradingService, SecurityCurrent und Transac- tionCurrent.
<pre>ObjectIdList list_initial_services();</pre>	liefert eine Liste der Kennungen aller verfügba- ren Standarddienste des ORBs, die durch resolve_initial_references() aufgelöst wer- den können.

Tabelle 3-1: Operationen der Pseudo-Schnittstelle CORBA::ORB

CORBA ::ORB

Operation	Beschreibung
<pre>string object_to_string (in Object obj);</pre>	<p>Die Methode <code>object_to_string()</code> konvertiert eine Objektreferenz in eine ASCII-Zeichenkette. Solche Zeichenketten können dann über gemeinsamen Dateizugriff, WWW oder auf eine andere geeignete Weise ausgetauscht werden. Der Name der Methode ließe auch vermuten, daß hier ganze Objekte in Zeichenketten umgewandelt werden – dem ist nicht so. Mit der Wandlung ganzer Objekte in einen Datenstrom befaßt sich der Externalisation-Service, der in Kapitel 7.5 beschrieben ist.</p>
<pre>Object string_to_object (in string str);</pre>	<p>formt eine textuelle Objektreferenz in eine reguläre um und bildet somit das analoge Gegenstück zu <code>object_to_string()</code>.</p>
<pre>Status create_list (in long count, out NVList new_list);</pre>	<p>wird für die Erzeugung der in CORBA oft (z. B. für die in Kapitel 5 beschriebenen Kontexte) verwendeten Name/Wert-Listen benötigt, unter anderem von <code>create_operation_list()</code>.</p>
<pre>Status create_operation_list (in OperationDef oper, out NVList new_list);</pre>	<p>erzeugt eine Name/Wert-Liste für die Parameter einer bestimmten Operation, die durch die Bezeichnung <code>oper</code> festgelegt wurde. Der Bezeichner wird zur Ermittlung der Parametertypen aus dem Interface-Repository verwendet.</p>
<pre>Status get_default_context (out Context ctx);</pre>	<p>Eine umfassende Erläuterung zu Kontexten finden Sie auf Seite 88.</p>

Tabelle 3-1: Operationen der Pseudo-Schnittstelle CORBA::ORB (Fortsetzung)

CORBA ::Object

Operation	Beschreibung
InterfaceDef get_interface ();	liefert ein Objekt des Interface-Repository, dem Informationen über die IDL-Spezifikation des Objekts entnommen werden können.
boolean is_nil ();	liefert einen Wahrheitswert, ob das Objekt ein Null-Objekt ist. Ähnlich wie NIL in Pascal oder NULL in C/C++ stellt CORBA die Definition eines Null-Objekts bereit.
Object duplicate ();	dupliziert eine Objektreferenz, da einer Anwendung selbst diese Aktion nicht erlaubt ist. Referenzen sind transparente Daten, deren Aufbau ORB-spezifisch ist. Eine semantisch äquivalente Kopie kann daher auch nur vom ORB hergestellt werden.
void release ();	gibt eine Objektreferenz frei. Auch hier handelt es sich um eine ORB-interne Aktion, daher darf eine Anwendung eine Objektreferenz nicht selbst löschen.
boolean is_a (in string logical_type_id);	liefert einen Wahrheitswert, ob ein Objekt eine bestimmte IDL-Schnittstelle implementiert.
boolean non_existent ();	prüft, ob ein entferntes Objekt noch existiert und liefert darüber einen Wahrheitswert. Diese Operation kann der Überprüfung und Aktualisierung gepufferter Objektlisten dienen.
boolean is_equivalent (in Object other_object);	liefert einen Wahrheitswert, ob ein Objekt wertgleich mit einem anderen Objekt ist.
unsigned long hash (in unsigned long maximum);	Bildung eines eindeutigen numerischen Wertes, der einem Objekt zugeordnet werden kann.

Tabelle 3-2: Operationen der Schnittstelle CORBA::Object

CORBA ::Object

Operation	Beschreibung
<pre>Status create_request (in Context ctx, in Identifier operation, in NVList arg_list, inout NamedValue result, out Request request, in Flags req_flags);</pre>	<p>erzeugt ein Request-Objekt für einen speziellen Objekttyp. Diese Operation wird für das Dynamic Invocation Interface (DII) benötigt, da Anfragen stets schnittstellenspezifisch sind.</p>
<pre>Policy get_policy (in PolicyType policy_type);</pre>	<p>ermittelt eine Policy zu einem bestimmten Objekt. Der Parameter <code>policy_type</code> bestimmt, um welche Art Policy es sich handelt. Durch den Policy Type <code>LIFESPAN_POLICY_ID</code> könnte man beispielsweise ermitteln, ob das Objekt transient oder persistent ist (näheres dazu im folgenden Kapitel).</p>
<pre>DomainManagersList get_domain_managers ();</pre>	<p>liefert die Liste aller Domain-Manager, die mit dem jeweiligen Objekt assoziiert wurden. Diese Domain-Manager bieten Informationen über Policies, die auf das Objekt zutreffen.</p>
<pre>ImplementationDef get_implementation ();</pre>	<p>liefert eine plattformabhängige Beschreibung über die Implementierung eines Objekts. Die OMG hat diese Operation in CORBA 2.2 als »deprecated« (mißbilligt) eingestuft; ab CORBA 2.3 gibt es diese Operation nicht mehr.</p>

Tabelle 3-2: Operationen der Schnittstelle CORBA::Object (Fortsetzung)