2 Statische Stubs

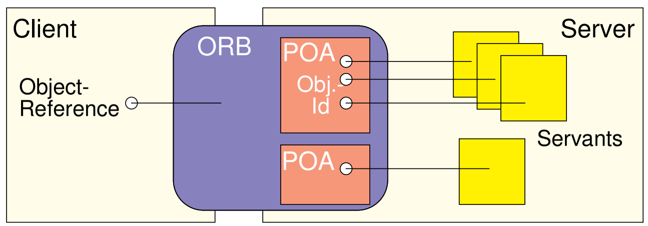
- Stellvertreter auf der Client- und Serverseite
  - ➤ sobald die Schnittstelle eines Objekts feststeht, können Stubs daraus erzeugt werden
  - ➤ Statische Stubs werden aus der IDL Beschreibung automatisch erzeugt
    - Mittels Precompiler und Compiler werden die Stubs aus der IDL Beschreibung erzeugt. Dabei helfen oft entsprechende Makefiles bei der Erstellung.
  - ➤ Auf Serverseite werden die Stubs (ausgefüllte) *Skeletons* genannt
    - Die Skeletons müssen mit der Implementierung des Objekts (vom Programmierer) ausgefüllt werden. Sie enthalten zunächst lediglich das Methodenskelett.
- Aufgaben der Stubs
  - ➤ Verpacken und Entpacken der Parameter (Marshalling)
  - ➤ Abschicken bzw. Entgegennehmen von Methodenaufruf-Mitteilungen über den ORB-Core

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

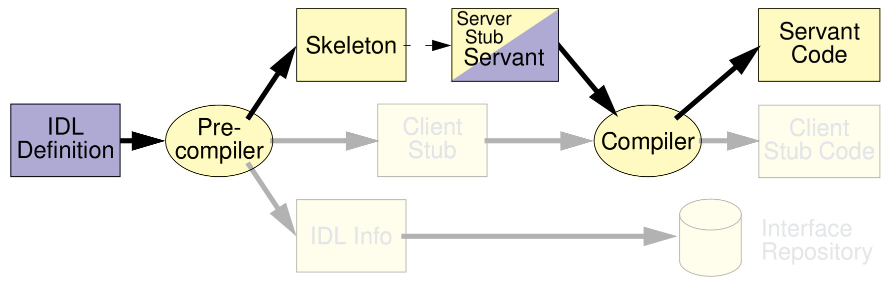
3 Portable Object Adaptor (POA): Terminologie

- ***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



4 Objekterzeugung

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

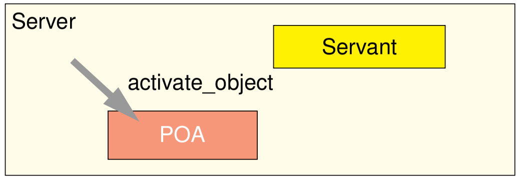


- Prinzipieller Ablauf bei der Definition eines CORBA Objekts
  - Aus der IDL-Definition der Objektschnittstelle werden mittels eines Precompilers mehrere Ausgabedateien erzeugt.
  - Das Skeleton enthält ein Skelett für den Servant. Es enthält bereits die Server-Stub-Funktionalität und muss um die Implementierung der eigentlichen Funktionalität des Servant-Objekts ergänzt werden. Den Servant bildet dann den Server-Stub inklusive der Implementierung des Servant-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-Stub und Servant müssen in der Regel noch von einem Compiler übersetzt werden. Die daraus entstehenden Codestücke werden dynamisch oder statisch in die Applikation eingebunden.

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*

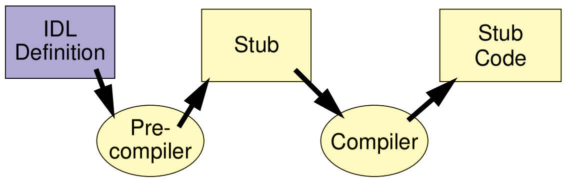


4 Objekterzeugung (3)

- 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 Cliebt wird dann ein Client-Stub zur Weiterleitung der Aufrufe an den Server installiert

5 Objektnutzung

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



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

5 Objektnutzung (3)

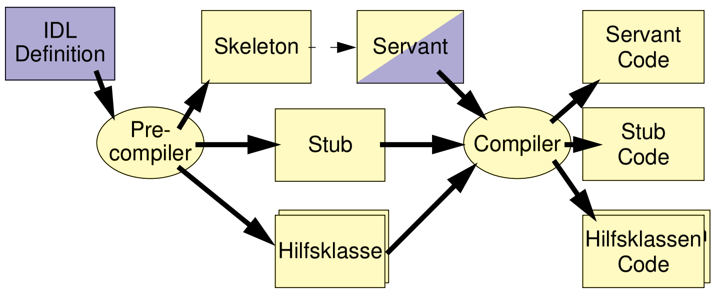
- Parameterübergabe (fortges.)
  - Übergabe bei **`out`** und **`inout`** Parametern sprachabhängig
    - ➤ Problem: Programmiersprachen kennen meist keine Möglichkeit mehrere Ergebnisse zurückzugeben
    - ➤ z. B. Java: Einführung von Holder-Klassen
      - – z. B. **`AccountHolder`**: kann eine Objektreferenz 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

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)

5 Objektnutzung (5)

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



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 ...

G.5 Object-Adaptor

- Aufgaben des Objektadapters
  - ◆ Generierung der Objektreferenzen
  - ◆ Entgegennahme eingehende Methodenaufruf-Anfragen
  - ◆ Weiterleitung von Methodenaufrufen an den entsprechenden Servant
  - ◆ Authentisierung des Aufrufers (Sicherheitsfunktionalität)
  - ◆ Aktivierung und Deaktivierung von Servants
    - 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
- 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

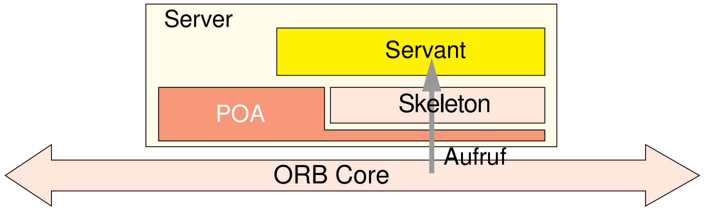
1 Portable-Object-Adaptor (POA): Ziele

- 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

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)



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