D.6 Java RMI

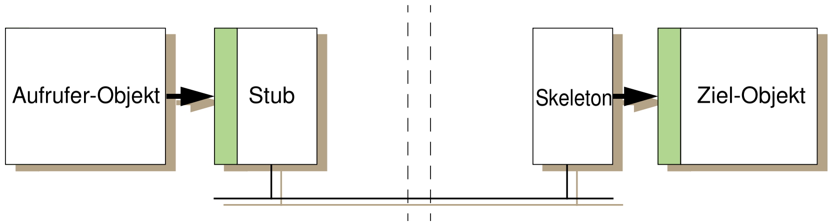
- Infrastruktur zur Unterstützung von verteilten Java-Anwendungen
  - Server-Anwendung
    - erzeugt Objekte
    - macht Objektreferenzen verfügbar
    - wartet auf Methodenaufrufe
  - Client-Anwendung
    - besorgt sich Remote-Referenz
    - ruft Methoden an entferntem Objekt auf
- Probleme
  - Entfernte Objekte finden
  - Methodenaufuf
  - Übergabe von Objekten

1 Entfernte Objekte finden

- Nameserver zur Umwandlung von Namen in Remote-Referenzen (rmiregistry)
  - Name = URL, bestehend aus Registry-Host[:Port] und Objektnamen
- Spezielle Klasse **`java.rmi.Naming`** für transparenten Zugriff auf Nameserver
  - Server meldet Objekt bei seiner Registry an
    - **`void Naming.bind(String name, Remote obj)`**
      - registriert ein Objekt unter einem eindeutigen Namen, falls das Objekt bereits registriert ist wird eine Exception ausgelöst
    - **`void Naming.rebind(String name, Remote obj)`**
      - registriert ein Objekt unter einem eindeutigen Namen, falls das Objekt bereits registriert ist wird der alte Eintrag gelöscht
  - Client bekommt Referenz von Registry
    - **`Remote Naming.lookup(String name)`**
      - liefert die Objektreferenz zu einem gegebenen Namen
- Alternative: ein Objekt erhält Remote-Referenz als Parameter oder als Ergebnis eines Methodenaufrufs

2 Methodenaufruf

- Klassische Stub-/Skeleton-Technik



- Stub und Skeleton werden aus der Implementierung des Zielobjekts generiert (`rmic`)
- Stub-Klasse wird bei Bedarf automatisch vom Server geladen (`RMIClassLoader`)
- Ausführungssemantik: "at most once"
  - ➤ bei Fehler wird `RemoteException` ausgelöst

3 Parameterübergabe

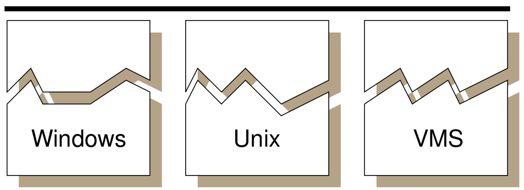
- keine einheitliche Parameterübergabesemantik
  - ◆ Problem: Übergabe von Referenzen auf Objekte, die kein Remote-Interface implementieren
    - ➤ keine Stub- und Skeleton-Klassen vorhanden
  - ◆ Ausweg: unterschiedliche Übergabesemantik
    - ➤ by reference: für alle Objekte von Klassen, die ein Remote-Interface implementieren
    - ➤ by value: für alle anderen Objekte
      - Objektzustand muss allerdings zumindest serialisiert werden können, Klasse kann über RMIClassLoader geholt werden.

4 Resumee

- einfacher, speziell auf Java abgestimmter Fernaufrufmechanismus
- implizite, nicht-orthogonale Interaktion (durch Remote-Referenzen)
- Verteilung nicht transparent

D.7 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.

D.8 CORBA — Architektur

1 Überblick

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

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: *Fundametals of Distributed Object Systems*. Wiley, 2001.

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

D.8 CORBA — Architektur

3 Motivation

- verschiedene Orte im verteilten System
  - ➤ Ortstransparenz
- Heterogenität im verteilten System
  - – unterschiedliche Hardware
  - – unterschiedliche Betriebssysteme und
  - – unterschiedliche Programmiersprachen
  - ➤ Transparenz der Heterogenität
- Dienste im verteilten System
  - ➤ Namensdienst
  - ➤ Zeitdienst
  - ➤ Transaktionsdienst
  - ➤ …

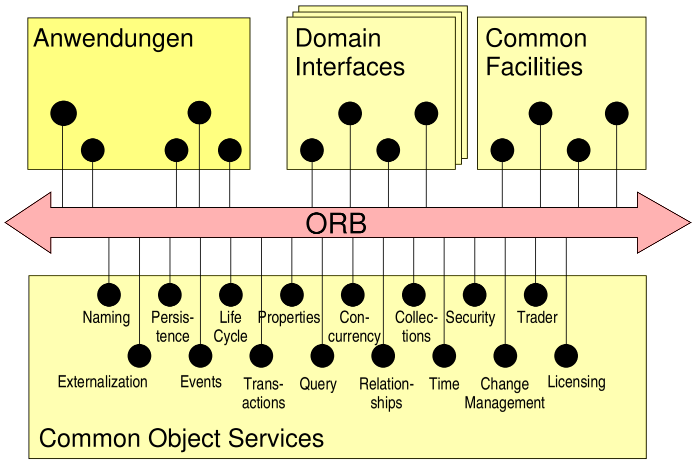
4 Entwurfsziele

- CORBA ist ein Standard
  - ◆ Standard-Dokumente
  - ◆ Definition von "CORBA compliance"
  - ◆ Hersteller realisieren Implementierungen des Standards (z. B., VisiBroker, Orbix, Orbacus, MICO, etc.)
- CORBA ist eine Middleware zur Abstraktion von
  - ◆ Hardware,
  - ◆ Betriebssystem und
  - ◆ Programmiersprache

4 Entwurfsziele (2)

- Interoperabilität
  - ◆ Eine Anwendung soll ohne Änderungen auf verschiedenen CORBA-Implementierungen ablaufen können
  - ◆ Anwendungen auf verschiedenen CORBA-Implementierungen sollen miteinander kommunizieren können
    - Mit Hilfe von CORBA soll nicht nur von Hardware, Betriebssystem und Programmiersprache abstrahiert werden, sondern auch von der Implementierung des CORBA Systems selbst.
- Einbindung von Altapplikationen ("Legacy-Anwendungen")
  - ◆ 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 miteinander in Kontakt stehen können.

5 OMA – Object Management Architecture



- 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 Anwendungsobjekte stehen untereinander in Client/Server-Beziehung.
- Die Common Object Services sind allgemeine, anwendungsunabhängige, standardisierte Systemdienste, die eine CORBA Implementierung anbieten kann bzw. muss, z. B. den Naming Service zum Auffinden von Objekten anhand eines Namens oder einen Transaktionsdienst zur Unterstützung von Transaktionen im verteilten System. CORBA-Services verhalten sich wie normale Objekte.
- Common Facilities sind ähnlich wie Object Services unabhängig von einem bestimmten Anwendungsbereich. Im Gegensatz zu Services legen sie aber nicht Schnittstellen für Systemdienste, sondern Schnittstellen zu Benutzeranwendungen (z. B. für Dokumentenbearbeitung oder Grafikschnittstellen) fest.
- Domain Interfaces legen ähnlich wie Facilities Anwendungsschnittstellen fest, die sich allerdings an bestimmten Anwendungsbereichen oder Branchen orientieren. Beispielsweise Product Data Management (PDM), Telekommunikation, Dienste des Gesundheitswesens oder Finanzanwendungen.

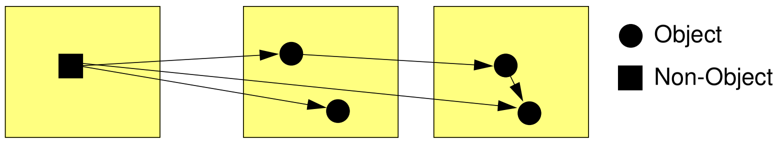
6 CORBA-Implementierungen

- Eine CORBA-Implementierung muss enthalten:
  - die Implementierung der Kern-Architektur
  - eine Sprachabbildung (z. B. für C++)
- Eine CORBA-Implementierung kann ausserdem enthalten:
  - eine beliebige Zahl von Services
  - Funktionalität für die Interoperabilität mit anderen CORBA-Implementierungen (GIOP, IIOP)
- **Wie** die Implementierung die Anforderungen des Standards realisiert bleibt freigestellt
  - unterschiedliche Implementierungen sind möglich
    - als Daemon
    - als Bibliothek
    - ...

D.9 CORBA-Anwendungsobjekte

1 Verteilte Objekte

- Identität
- Zustand
- Methoden
- CORBA-Objekte können aufgerufen werden (realisieren Server-Seite)
- CORBA-Objekte können als Client agieren



- Clients müssen keine Objekte sein (können auch Prozesse, etc. 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.

1 Verteilte Objekte (2)

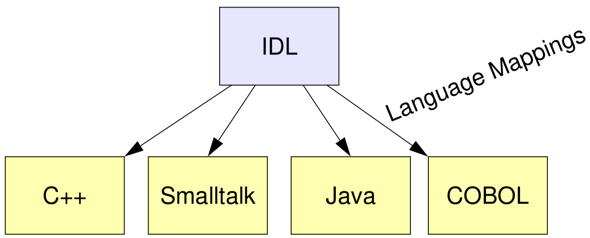
- Verteilte Objekte bilden eine Anwendung
  - Anwendungsobjekte liegen auf verschiedenen Rechnern
    - Die Anwendungsobjekte sind die Einheiten der Verteilung in CORBA. Objekte auf einem Rechner können auf eine oder mehrere Systemeinheiten verteilt sein, z. B. auf mehrere Prozesse eines UNIX Systems.
  - 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.

- implizite, nicht-orthogonale Interaktion
  - (Remote-)Methodenaufrufe
  - Client-Stub vermittelt Aufruf an Server-Objekt
  - **At-most-once** / **exactly-once** Semantik

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:
    C, C++, Smalltalk, COBOL, Ada und Java
  - IDL ist an C++ angelehnt



2 Interface Definition Language (2)

- Beispiel

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

IDL to C++

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

2 Interface Definition Language (3)

- Beispiel

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

IDL to Java

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

```
nam
    ...
};
```


2 Interface Definition Language (4)

★ Vorteile

- Unabhängigkeit/Transparenz von der Implementierungssprache
- Ermöglicht Interoperabilität über Sprachgrenzen

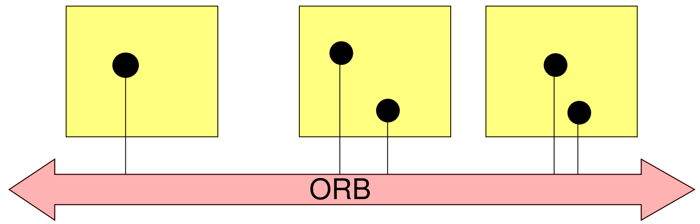
▲ Nachteile

- Objektschnittstellen müssen in der Zielsprache **und** in IDL spezifiziert werden
- IDL ist sehr ausdrucksstark
  - Language mapping für Sprachen, die nicht die entsprechenden Mechanismen anbieten (z. B. C) kann sehr komplex sein
- Wenn eine Sprache spezielle Eigenschaften anbietet, können diese nicht genutzt werden, weil sie von IDL nicht erfasst werden

3 Objekte Erzeugen und Binden

- Erzeugen eines Server-Objekts
  - 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

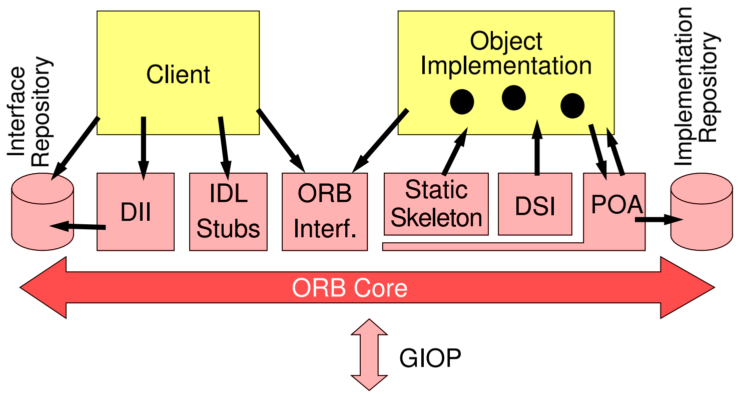
D.10 Object Request Broker – ORB



- Der ORB ist das Rückgrat einer CORBA-Implementierung
- Alle Kommunikation zwischen den Objekten läuft über den ORB
  - ... innerhalb eines Adressraums / Prozesses
  - ... zwischen Adressräumen / Prozessen
  - ... zwischen Adressräumen / Prozessen von verschiedenen ORBs
- Der ORB implementiert Ortstransparenz

1 Architektur

■ Zentrale Komponenten eines ORB



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.

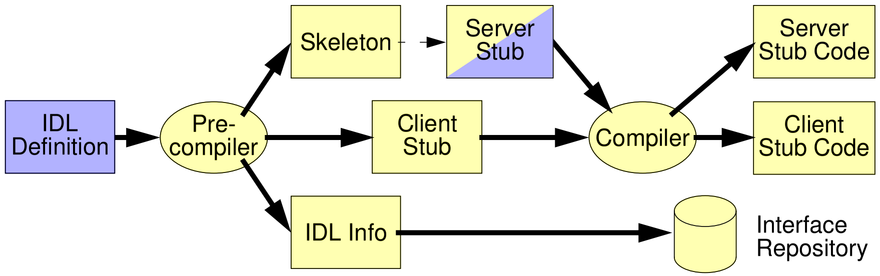
■ Aufgaben der Stubs

➤ Verpacken und Entpacken der Parameter (Marshalling)

➤ Abschicken bzw. Entgegennehmen von Methodenaufruf-Mitteilungen über den ORB-Core

2 Statische Stubs (2)

- Stub-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 das Serverobjekt. Es enthält bereits die Server-Stub-Funktionalität und muss um die Implementierung der Methoden des Server-Objekts ergänzt werden. Es 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.

3 Interface Repository

- Datenbank für Schnittstellendefinitionen in IDL
  - Alle IDL Schnittstellen werden dort abgelegt.
- Abfrage der Datenbank
  - ◆ für Typechecking
    - Der ORB kann prüfen, ob die Schnittstelle des aufgerufenen Objekts mit der Schnittstelle übereinstimmt, die der Client-Stub des Objekts repräsentiert.
    - ➤ bei Inter-ORB-Operationen: Hinterlegung in mehreren Repositories
  - ◆ zum Abruf von 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.