CORBA
Einführung

WS 2004/05

CORBA - Literatur

- Bücher
  - Henning, Vinoski: *Advanced CORBA Programming with C++*
  - Brose et al: *Java Programming with CORBA*
  - Tari, Bukhres: *Fundamentals of Distributed Object Systems*
  - Siegel**:** *CORBA 3 Fundamentals and Programming*
- Links
  - Tutorials
    - http://www.iona.com/devcenter/orbacus/training_download.html
    - http://developer.java.sun.com/developer/onlineTraining/corba/
  - Allgemein
    - http://www.omg.org
    - http://www.cetus-links.org/oo_corba.html
  - ORBs: Vergleich und Download
    - http://www.middleware.org/object/orb.html
    - http://adams.patriot.net/~tvalesky/freecorba.html

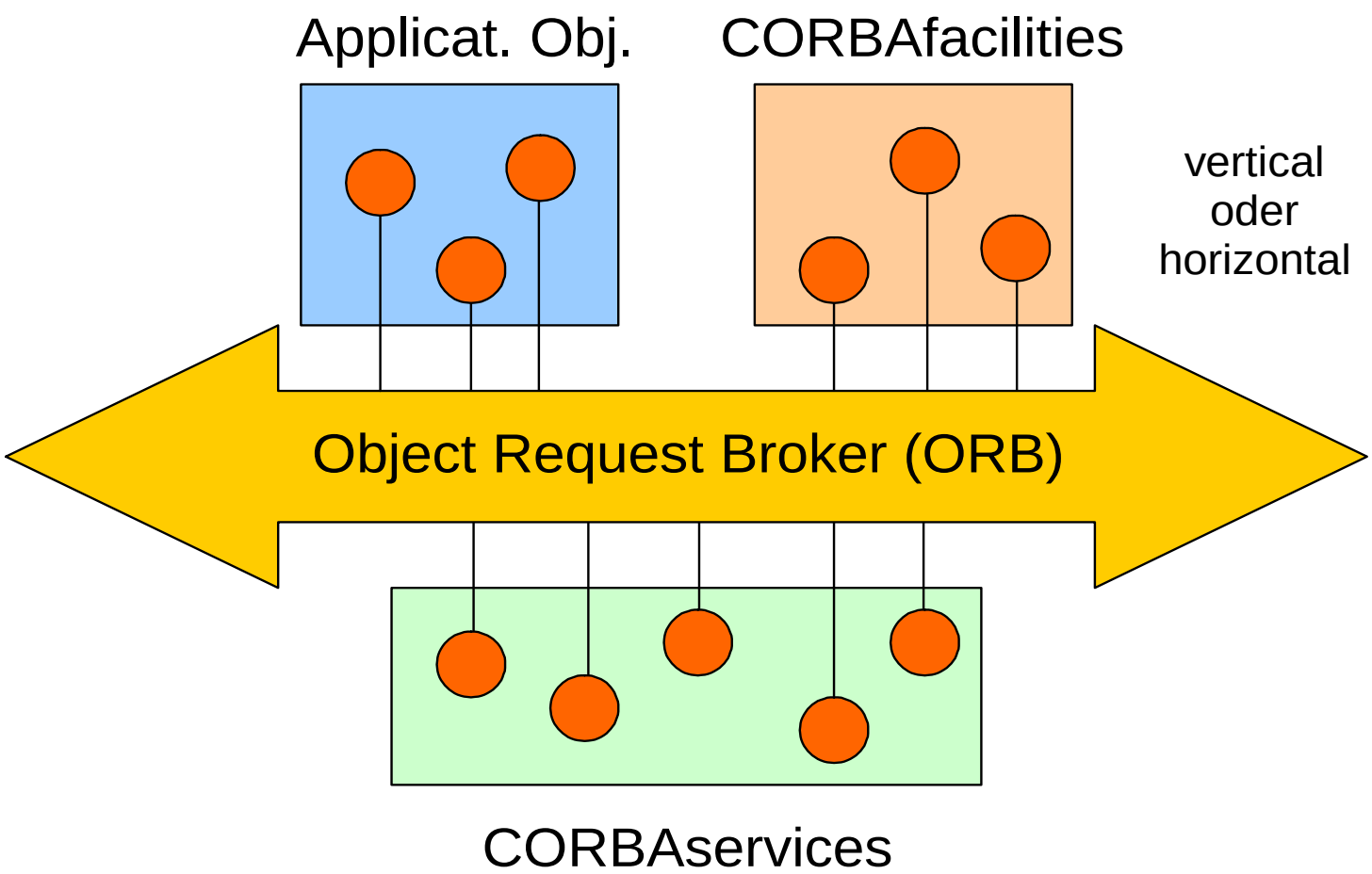
OMG – Object Management Group

- Geschichte
  - 1989 mit 8 Mitgliedern gegründet (3COM, Sun, HP, IBM,..)
  - Mittlerweile über 800 Mitglieder.
- Aufgaben
  - Bereitstellung organisatorischer Infrastruktur zur Erstellung von Spezifikationen.
  - Mitglieder produzieren kommerzielle Implementierungen auf Basis der Spezifikationen.
- Ziel
  - Förderung von OOP.
  - Entwicklung eines gemeinsamen Frameworks für verteilte OO-Anwendungen.

Object Management Architecture – OMA

- OMA definiert ein Framework für verteilte System-Architekturen.
- Core Object Model
  - Definition der *theoretischen Grundlagen* verteilter OO-Architekturen wie CORBA.
  - Hauptziele sind *Portabilität* und *Interoperabilität*.
  - Eher interessant für ORB-Implementierer.
- Reference Model
  - Definition des Zusammenwirkens der verschiedenen Komponenten eines verteilten (CORBA-basieten) OO-Systems.

Das Referenz-Modell



Object Request Broker (ORB)

- ORB stellt *Basis-Infrastruktur* für verteilte OO-Anwendungen zur Verfügung.
- Objekte können über ORB *Nachrichten austauschen*.
- Wo sich Objekte befinden ist irrelevant.
- Zugriff auf ORB erfolgt über *abstraktes Interface*.
- ORB-Implementierung ist für Verwender nicht sichtbar.
- ORB kann *mehrere Implementierungs-Spachen* unterstützen.

CORBAservices

- Dienste, welche die *Verwaltung von Objekten* und die Interaktion zwischen Objekten erleichtern.
- *Domain-unabhängige* Bausteine zur Implementierung komplexerer Komponenten.
- OMG erstellt *Spezifikationen*.
- Kommerzielle Anbieter können Services implementieren.
- Wichtigste Services:
  - Naming/Trading
  - Event/Notification
  - Securtiy
  - Transaction

CORBAfacilities

- Dienste, die auf *Anwendungsebene* genutzt werden können.
- *Horizontale* Facilities: Domain-unabhängig
  - Datenaustausch,
  - Internationalisierung, ...
- *Vertikale* Facilities (*CORBAdomains*):
  - Produktion,
  - E-Commerce,
  - Finanz- und Versicherungswesen,
  - Gesundheitswesen (CORBAmed),
  - Transportwesen, ...

Historische Entwicklung (1)

- CORBA 1.1 (1992)
  - ORB-Kern, IDL
  - C-Anbindung
- CORBA 2.0 (seit Ende 1995)
  - Inter-ORB-Protokolle
  - C++ und Smalltalk
- CORBA 2.1 (August 1997)
  - Ada und COBOL
- CORBA 2.2 (Februar 1998)
  - IDL-to-Java, Java-to-IDL (RMI/IIOP), Portable Object Adapter (POA)
- CORBA 2.3 (Februar 1999)
  - Objects-by-Value (Objekt-Serialisierung)
- CORBA 2.4 (Oktober 2000)
  - Messaging, Realtime CORBA, Minimum CORBA ...
  - Naming Service

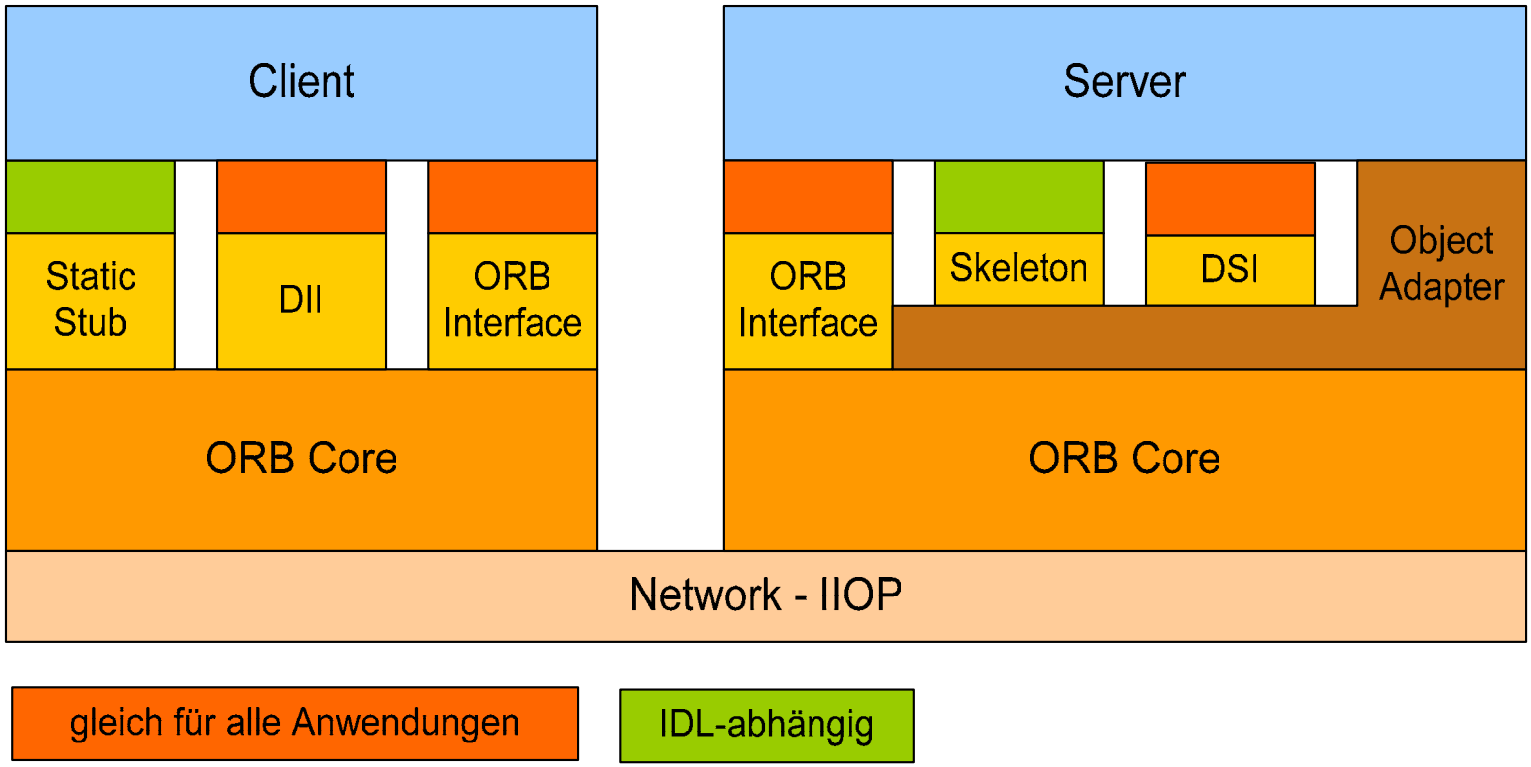
Historische Entwicklung (2)

- CORBA 2.5/2.6 (Ende 2001)
  - Fehlertoleranz, Sicherheit.
- CORBA 3 (Juli 2002)
  - Firewall Spezifikation
  - Quality of Service Control (QoS)
    - Asynchrone Aufrufe
    - CORBA für Embedded Systems
  - CORBA Component Model (CCM)
    - Container: Transaktionen, Sicherheit, Persistenz
    - Integration von EJB

Merkmale von CORBA

- CORBA ist ein Framework zu Entwicklung von verteilten OO-Anwendungen.
- Merkmale:
  - *Interface Definition Language (IDL):* Beschreibung der Objekt-Schnittstellen.
  - *Language Mappings:* Definition, wie auf Objekte zugegriffen werden kann und wie Objekte implementiert werden können.
  - *Entfernter Methodenaufruf:* Ortstransparenz
  - *Object Adapters:* ORB-unabhängige Implementierung von CORBA-Objekten.
  - *Inter-ORB Protokoll (IOP):* Kommunikation zwischen verschiedenen ORBs.

Common ORB Architecture – CORBA



Komponenten von CORBA (1)

- ORB-Core*:*
  - Implementiert Netzwerk-Funktionalität zum entfernten Methodenaufruf.
  - Anbieter-abhängige Implementierung.
- ORB-Interface:
  - API zur Kommunikation mit dem ORB.
  - Interface ist für Server und Client gleich.
  - Dient hauptsächlich zur Initialisierung.
- Static Stub*:*
  - Akzeptiert Requests von Clients und gibt sie an ORB weiter.
  - Stub wird für jedes Objekt aus IDL generiert.
  - Stub ist sprachabhängig, Interface ist aber für jeden ORB gleich.
- DII (Dynamic Invocation Interface):
  - Beliebige Requests können zusammengebaut und verschickt werden.
  - zur Laufzeit braucht Interface nicht bekannt zu sein.
  - Für Debugger bzw. Objekt-Browser.

CORBA-Komponenten (2)

- Skeleton*:*
  - Server-seitiges Interface, über das Methoden des Servers aufgerufen werden.
  - Server muss Methoden des Skeleton-Interfaces implementieren.
  - Ebenfalls aus IDL generiert.
- DSI (Dynamic Skeleton Interface):
  - Gegenstück zu DII.
  - Server kann Objekte implementieren, deren Interface er nicht kennt.
  - Für Protokoll-Brücken.
- Object Adapter:
  - Vermittler zwischen ORB und Server.
  - Leitet Request an Servants (Objekte, Prozeduren) weiter.
  - Portable Object Adapter (POA) ist standardisiert.
- IIOP (Internet InterORB Protokoll)
  - Standardisiertes Protokoll zum Transport von Requests.

Interface Definition Language – IDL

- Vertrag zwischen Client und Server, wie auf Objekte zugegriffen werden kann.
- Programmiersprachen-unabhängige *Schnittstellen-Beschreibungssprache.*
- *Keine Implementierungssprache* wie C++ oder Java.
- C-ähnliche Syntax.
- Beispiel:

```
interface Printer {
  enum ColorMode { BlackAndWhite, FullColor };
  void setColorMode(in ColorMode mode);
}
```

Language Mappings

Language Mapping ist eine Spezifikation, die definiert, wie IDL in eine bestimmte Programmiersprache übersetzt wird.

- OMG
  - C
  - C++
  - Smalltalk
  - Java
  - Ada
  - Cobol
  - Python
  - COM/CORBA Internetworking
- Diverse Hersteller
  - Eiffel
  - Modula 3
  - Tcl
  - PL/1
  - LISP
  - Perl
- Interoperabilität mit .NET
  - IIOP.NET (ETH)

Vorteile und Nachteile von CORBA

- Voreile:
  - Offener *Hersteller-unabhängiger* Standard.
  - Es existiert eine Vielzahl von Implementierungen.
  - Plattform- und Programmiersprachen-unabhängig.
  - Besonders für den Einsatz in *heterogenen Umgebungen* geeignet.
- Nachteile:
  - Kommerzielle ORBs sind relativ *teuer*.
  - Großer *Implementierungs-Overhead* bei kleinen Anwendungen.
  - Relativ großer *Einarbeitungsaufwand* (besonders bei C++).
  - Großer *Protokoll-Overhead* (besonders bei IIOP).

Eine minimale CORBA-Anwendung

- Interface-Definition: `ServerInfo.idl`

```
interface ServerInfo {
  boolean getServerEnv(
            in  string varName,
            out string varValue);
};
```

- IDL-File kompilieren: `idl ServerInfo.idl`
- Folgende C++-Files werden generiert:
  - `ServerInfo.cpp`
  - `ServerInfo.h`
  - `ServerInfo_skel.cpp`
  - `ServerInfo_skel.h`

für Client

für Server

Implementierung des Servants (1)

- Header-File `ServerInfo_Impl.h`

```
class ServerInfo_impl : virtual public POA_ServerInfo {
  virtual CORBA::Boolean getServerEnv(
        const char* varName, CORBA::String_out varValue)
        throw(CORBA::SystemException);
};
```

  - Skeleton beinhaltet abstrakte Klasse **`POA_ServerInfo`**.
  - Server muss virtuelle Methoden der Skeleton-Klasse implementieren (Servant).
  - Object-Adapter (POA) ruft virtuelle Methode **`getServerEnv()`** im Skeleton auf.
  - Aufgrund dynamischer Bindung wird **`getServerEnv()`** im Servant ausgeführt.

JH1

JH1 Johann Heinzelreiter; 02.11.2001

Implementierung des Servants (2)

- Implementierung der virtuellen Funktion (`ServerInfo_Impl.cpp`):

```
CORBA::Boolean ServerInfo_impl::getServerEnv(
      const char* varName, CORBA::String_out varValue)
      throw(CORBA::SystemException) {
   char* val = getenv(varName);
   if (val)
      varValue = CORBA::string_dup(val);
   else
      varValue = CORBA::string_dup("");
   return val != 0;
}
```

Implementierung des Servers (1)

- Main-Funktion des Serverprozesses

```
int main(int argc, char* argv[]) {
  CORBA::ORB_var orb;
  try {
```

- Initialisierung des ORBs

```
    orb = CORBA::ORB_init(argc, argv);
```

- Initialisierung des Object-Adapters (POA)

```
    CORBA::Object_var poaObj =
      orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var rootPOA =
      PortableServer::POA::_narrow(poaObj);
```

- Aktivierung des POA-Managers

```
    PortableServer::POAManager_var manager =
      rootPOA->the_POAManager();
    manager->activate();
```

Implementierung des Servers (2)

- Instanzierung und Registrierung des Servants

```
ServerInfo_impl sInfoImpl(rootPOA);
ServerInfo_var sInfo = sInfoImpl._this();
```

- Speichern der serialisierten IOR

```
CORBA::String_var ior =
   orb->object_to_string(sInfo);
ofstream out("ServerInfo.ref");
out << ior << endl;
```

- Starten des ORBs

```
orb->run();
```

- Abfangen von CORBA System-Exceptions

```
  } catch(const CORBA::Exception& ex) { … }
  return 0;
}
```

Imlementierung des C++-Clients

```
int main(int argc, char* argv[]) {
  CORBA::ORB_var orb;
  try {
```

- Initialisierung des ORBs

```
    orb = CORBA::ORB_init(argc, argv);
```

- Lesen der Objektreferenz

```
    ifstream in("ServerInfo.ref");
    string ior; in >> ior;
    CORBA::Object_var obj =
      orb->string_to_object(ior.c_str());
    ServerInfo_var sInfo = ServerInfo::_narrow(obj);
```

- Zugriff auf (entferntes) CORBA-Objekt

```
    CORBA::String_var value;
    bool defined = sInfo->getServerEnv("OS", value);
  }
  catch(const CORBA::Exception& ex) {}
  return 0;
}
```

Implementierung des Java-Clients

```
public static void main(String[] args) {
  try {
```

- ORB initialisieren:

```
    orb = ORB.init(args, null);
```

- Lesen der Objektreferenz

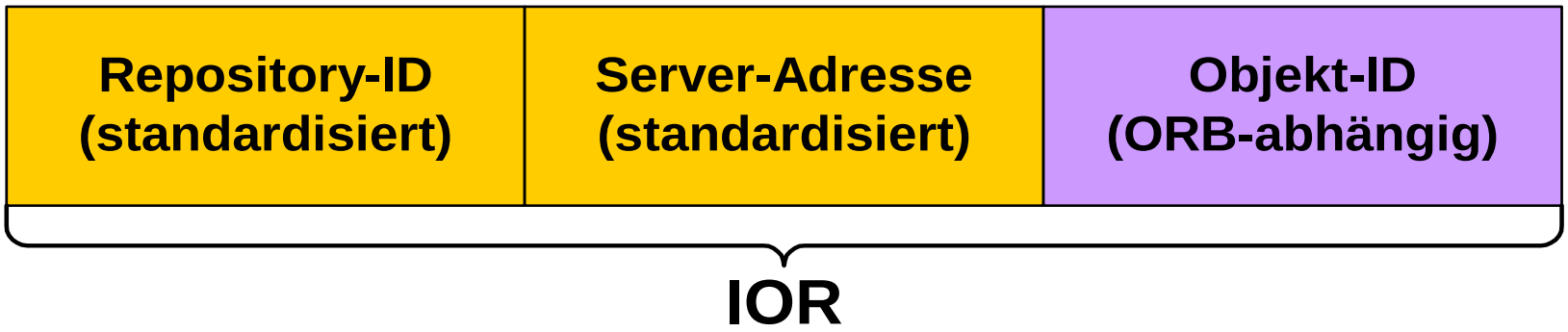
```
    BufferedReader in = new BufferedReader(
       new FileReader("ServerInfo.ref"));
    org.omg.CORBA.Object obj =
       orb.string_to_object(in.readLine());
    ServerInfo sInfo = ServerInfoHelper.narrow(obj);
```

- Zugriff auf (entferntes) CORBA-Objekt

```
    StringHolder varValue = new StringHolder();
    boolean defined = sInfo.getServerEnv("OS",
                                         varValue));
  }
  catch (org.omg.CORBA.SystemException ex) { … }
}
```

Interoperable Object Reference (IOR)

ORBs, die IIOP unterstützen, müssen Standard-format für Objektreferenzen unterstützen.



- Repository-ID
  - Verweis auf Interface-Repository (Metadaten)
  - für Typsicherheit
- Server-Adresse
  - Host/Port oder
  - Referenz auf Implementation Repository (IMR)