Design Patterns und  CORBA

Heiko Abraham
abraham@informatik.uni-freiburg.de

21. Januar 2001

Seminarvortrag “Softwarearchitekturen”

Universität Freiburg

Inhalt

1. Einleitung

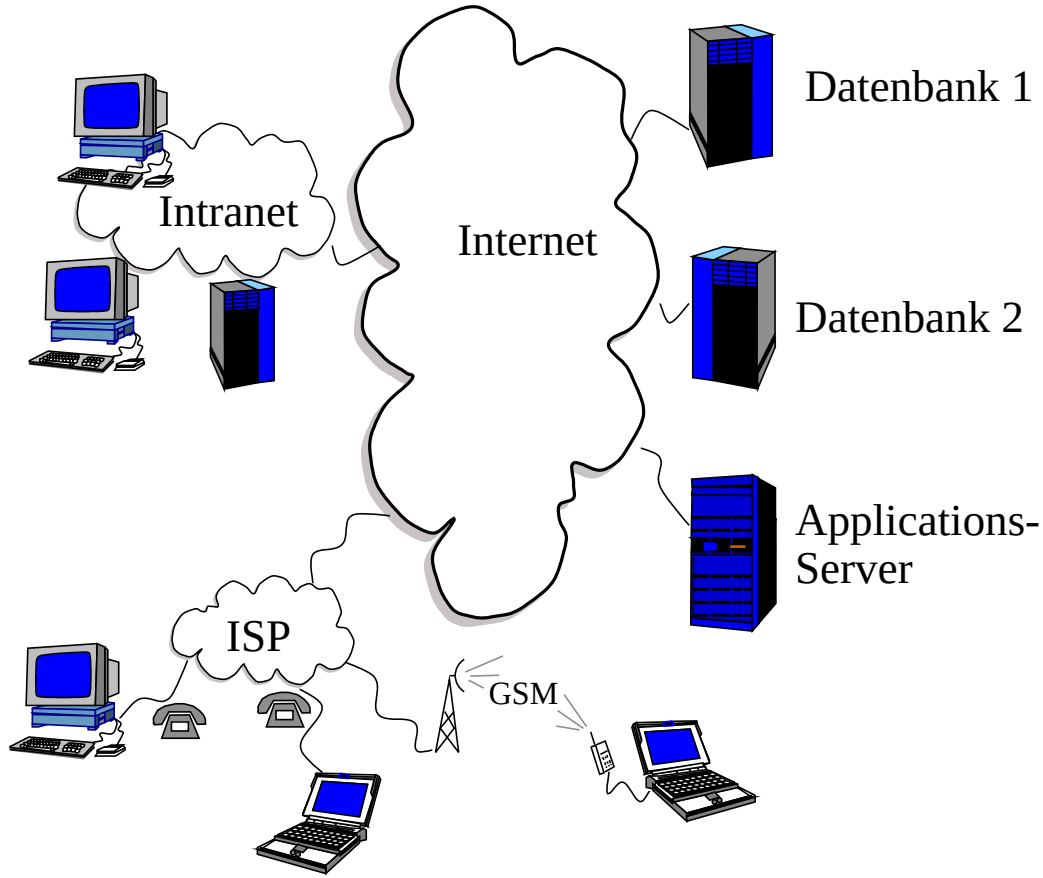
Beispiel: Informationsdienst

Situation:

- vernetzte und verteilte Systeme
- starke Heterogenität
- rechnerseitig:
  - Daten- und Funktionsverbund
  - Redundanz, Lastverteilung

Anforderungen:

- an die Softwarearchitektur
- Flexibilität
- Wiederverwertbarkeit/Wartbarkeit
- Interoperabilität

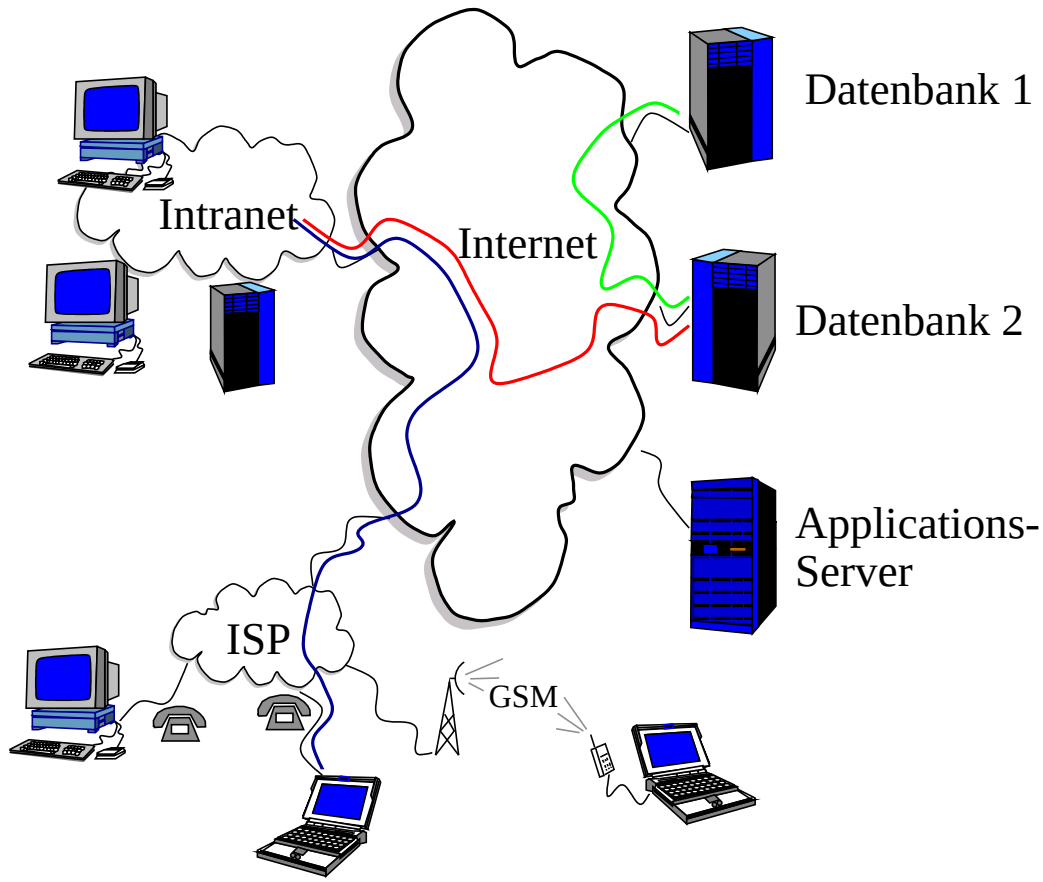


1. Einleitung

Beispiel: Informationsdienst

Situation:

- vernetzte und verteilte Systeme
- starke Heterogenität
- rechnerseitig:
  - Daten- und Funktionsverbund
  - Redundanz, Lastverteilung



Anforderungen:

- an die Softwarearchitektur
- Flexibilität
- Wiederverwertbarkeit/Wartbarkeit
- Interoperabilität

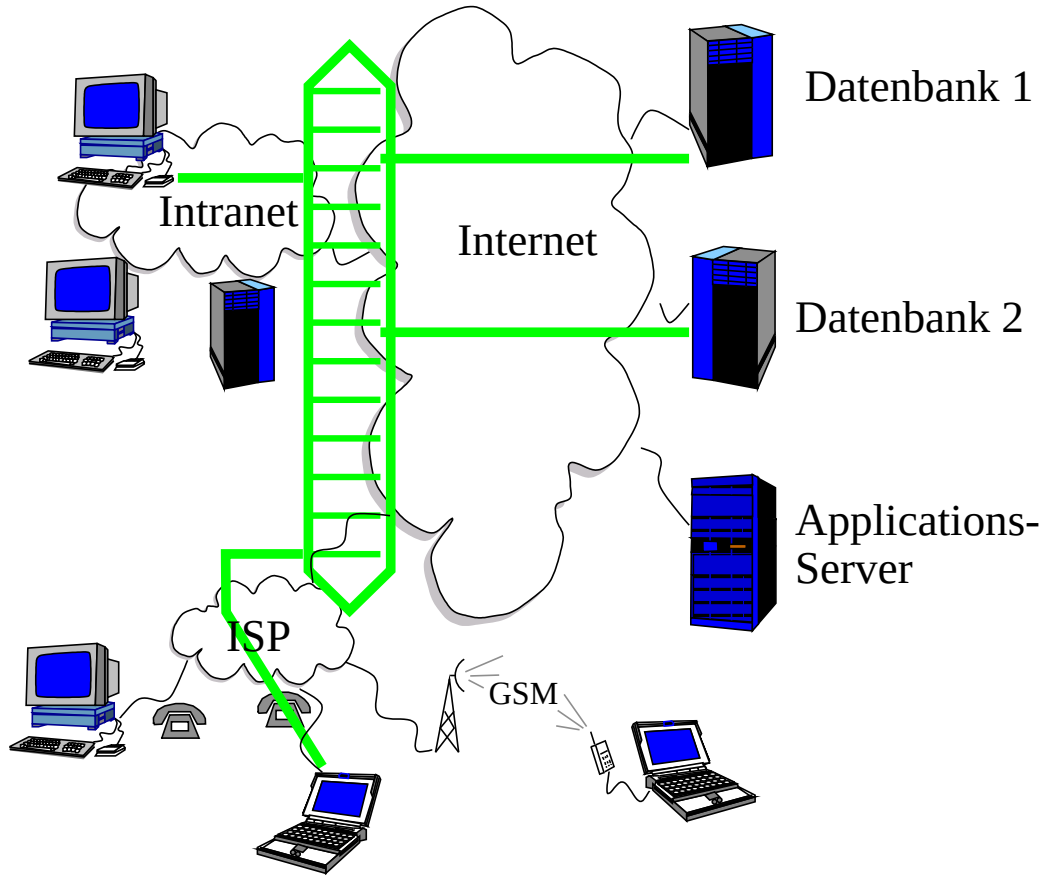
⇒ Middleware kann helfen

1. Einleitung

Beispiel: Informationsdienst

Situation:

- vernetzte und verteilte Systeme
- starke Heterogenität
- rechnerseitig:
  - Daten- und Funktionsverbund
  - Redundanz, Lastverteilung



Anforderungen:

- an die Softwarearchitektur
- Flexibilität
- Wiederverwertbarkeit/Wartbarkeit
- Interoperabilität

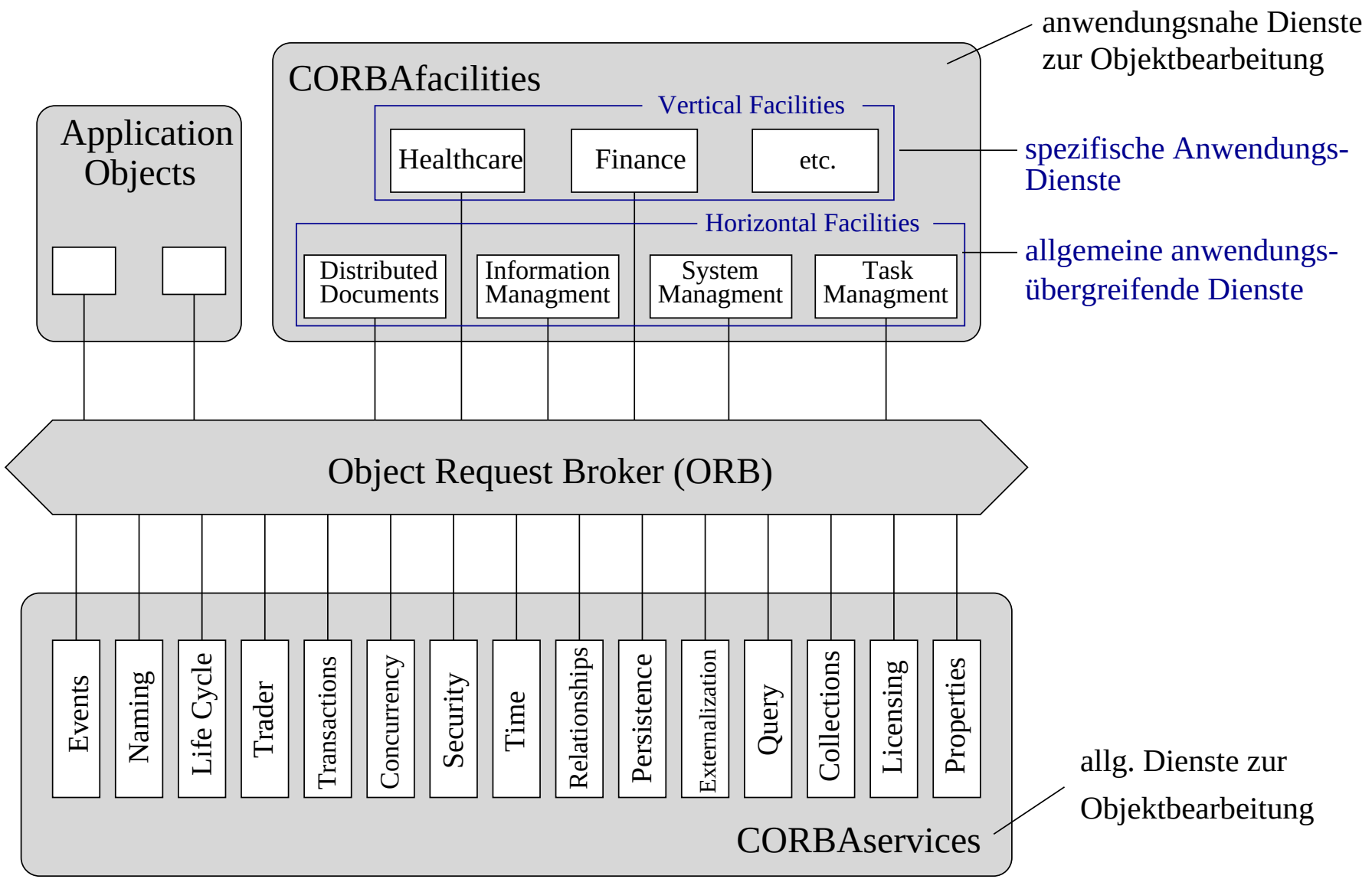
⇒ Middleware kann helfen ⇒ Objekt-Bus und bestehende Technologie verwenden

2. CORBA - Überblick

- CORBA = *Common Object Request Broker Architecture*
- CORBA = Industriestandard, sehr stabil und offen
- CORBA = Spezifikationsprodukt der *Object Managment Group* (OMG)
  - Festlegung der *Object Managment Architecture* (OMA)
  - Kommunikation von Objekten über *Object Request Broker* (ORB)
  - Definition der (Komponenten-) Schnittstellen, *Interface Definition Language* (IDL)
  - Definition von allgemeinen Diensten (*CORBAservices*)

- Bestehende Technologien verwenden
- Objektorientiertes Modell für Client-Server-Architekturen verwenden
- Implementierung in mehrere Programmiersprachen möglich
  (*Ada*, *Java*, *C++*, *C*, *Smalltalk*, *COBOL*, *Tcl*, *Perl*)

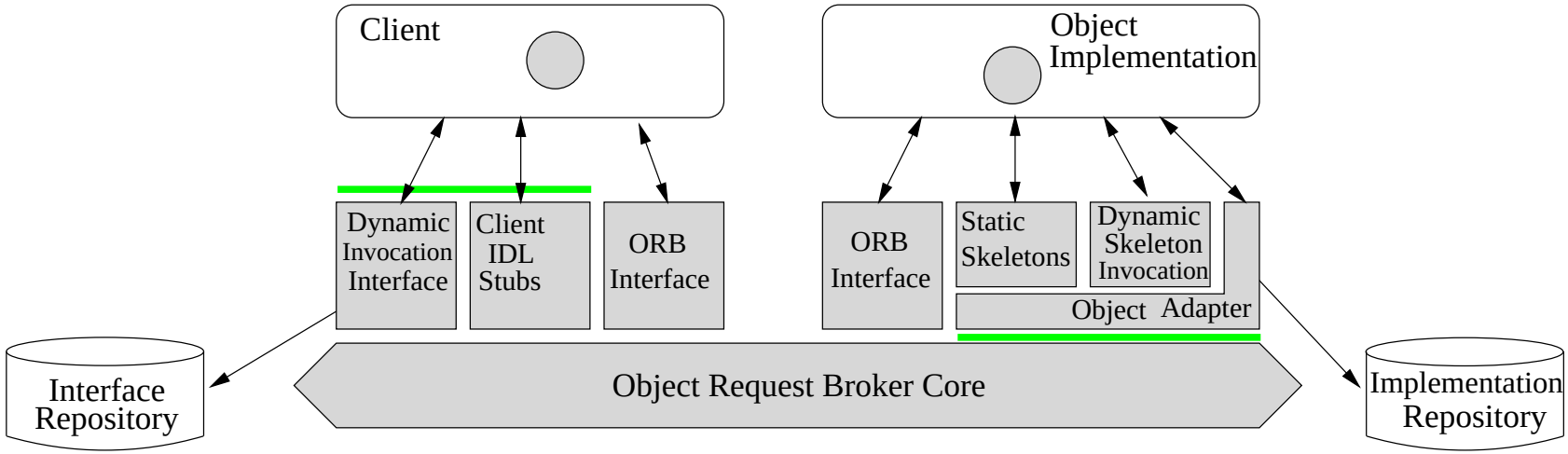
Object Managment Architecture (OMA)



Kurz und kleingedruckt : die CORBAservices

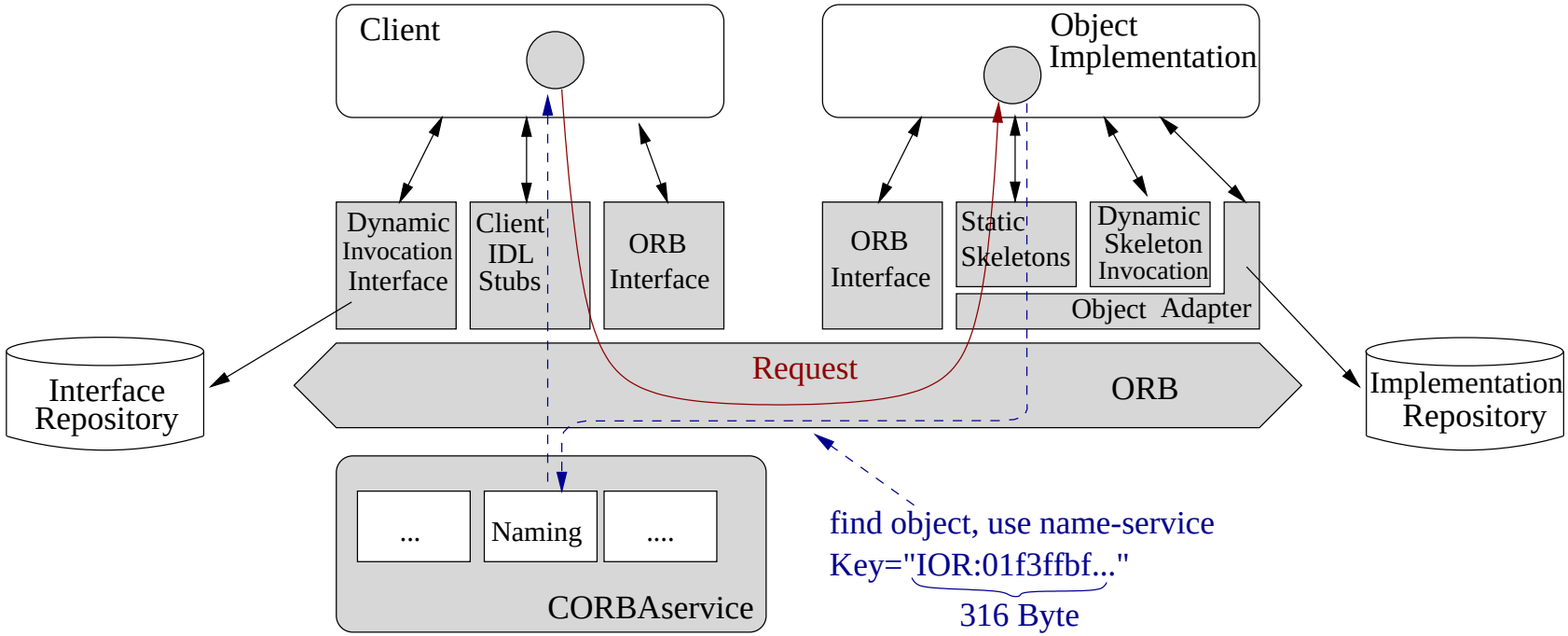
1. *Event-Service* - Komponenten registrieren ihr Interesse an Ereignissen, die in einem Event-Channel gesammelt werden.
2. *Naming-Service* - ermöglicht es Objekten, andere Objekte über ihren Namen zu finden.
3. *Life-Cycle-Service* - Operationen zum kopieren, verschieben oder löschen von Objekten.
4. *Trader-Service* - stellt die "Gelben Seiten" für Objekte bereit.
5. *Transaction-Service* - stellt ein zweistufiges Bestätigungsprotokoll zur Verfügung, (flache / geschachtelte Transaktionen).
6. *Concurrency-Control-Service* - umfaßt einen Lock-Manager, der im Auftrag Sperrungen vornehmen kann.
7. *Security-Service* - Framework für Sicherheit ; (Vertraulichkeit, Integrität, Unwiderlegbarkeit und Abrechenbarkeit).
8. *Time-Service* - für Uhrzeitsynchronisation oder zum Auslösen von zeitgetriggerten Ereignissen in verteilten Anwendungen.
9. *Relationship-Service* - erlaubt es, dynamische Verknüpfungen zwischen Komponenten zu erstellen.
10. *Persistence-Service* - zum Speichern von Objekten in Datenbanken über einheitliche Schnittstelle.
11. *Externalization-Service* - eine Standardmethode, um mittels Streaming Daten zwischen Komponenten auszutauschen.
12. *Query-Service* - stellt Abfrageoperationen für Objekte bereit.
13. *Collection-Service* - ermöglicht es, Schnittstellen von Objekten in Collections zu sammeln und zu manipulieren.
14. *Licensing-Service* - bietet Möglichkeiten, die Nutzung der Komponenten abrechnen zu können.
15. *Properties-Service* - erlaubt es, Name-Wert-Paare mit beliebigen Komponenten zu assoziieren.

Komponenten von CORBA



- *ORB-Interface* ist für alle ORBs identisch
- *static IDL-Stubs* und *static IDL Skeletons* existieren pro Objekt
- generell sind mehrere *Object-Adapter* möglich

Komponenten von CORBA - Operationsaufruf (mit Namensdienst)

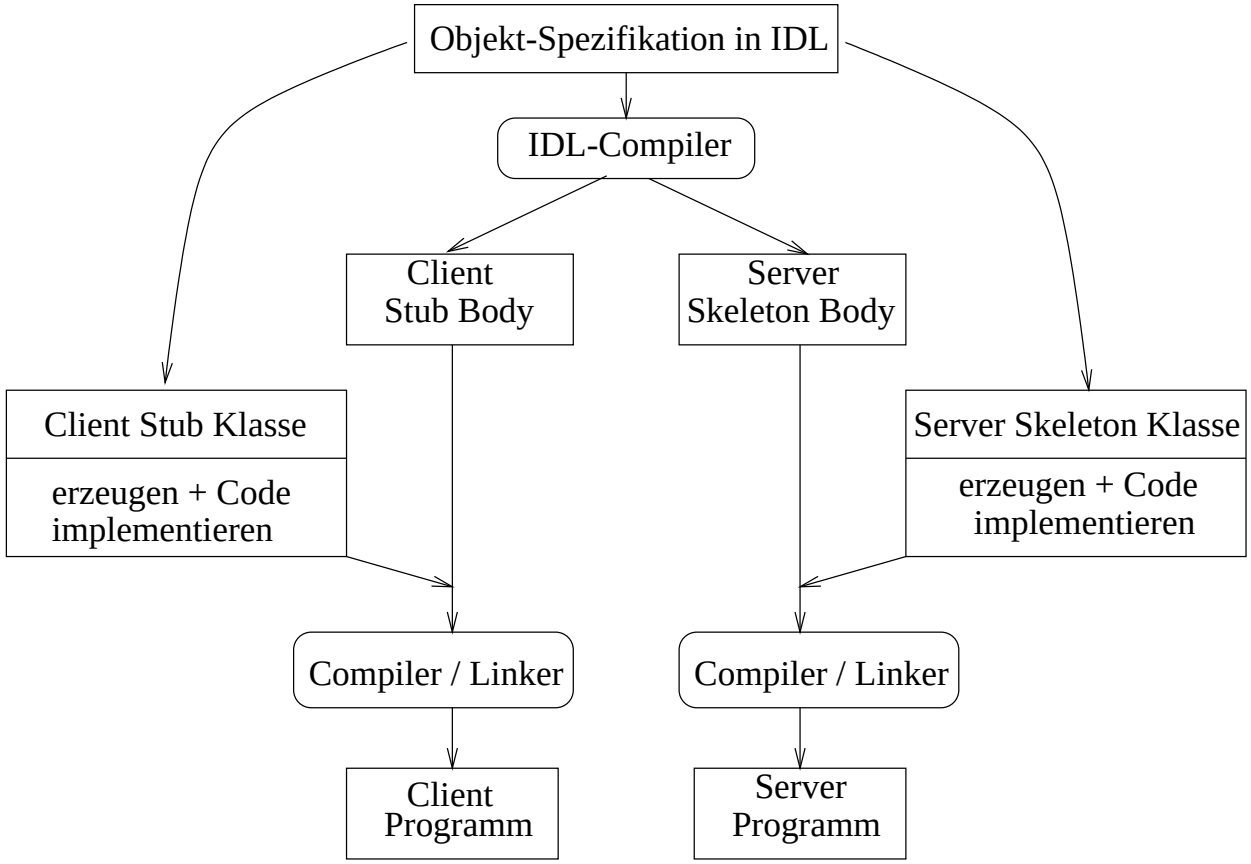


Möglichkeiten, die ein CORBA-ORB bietet

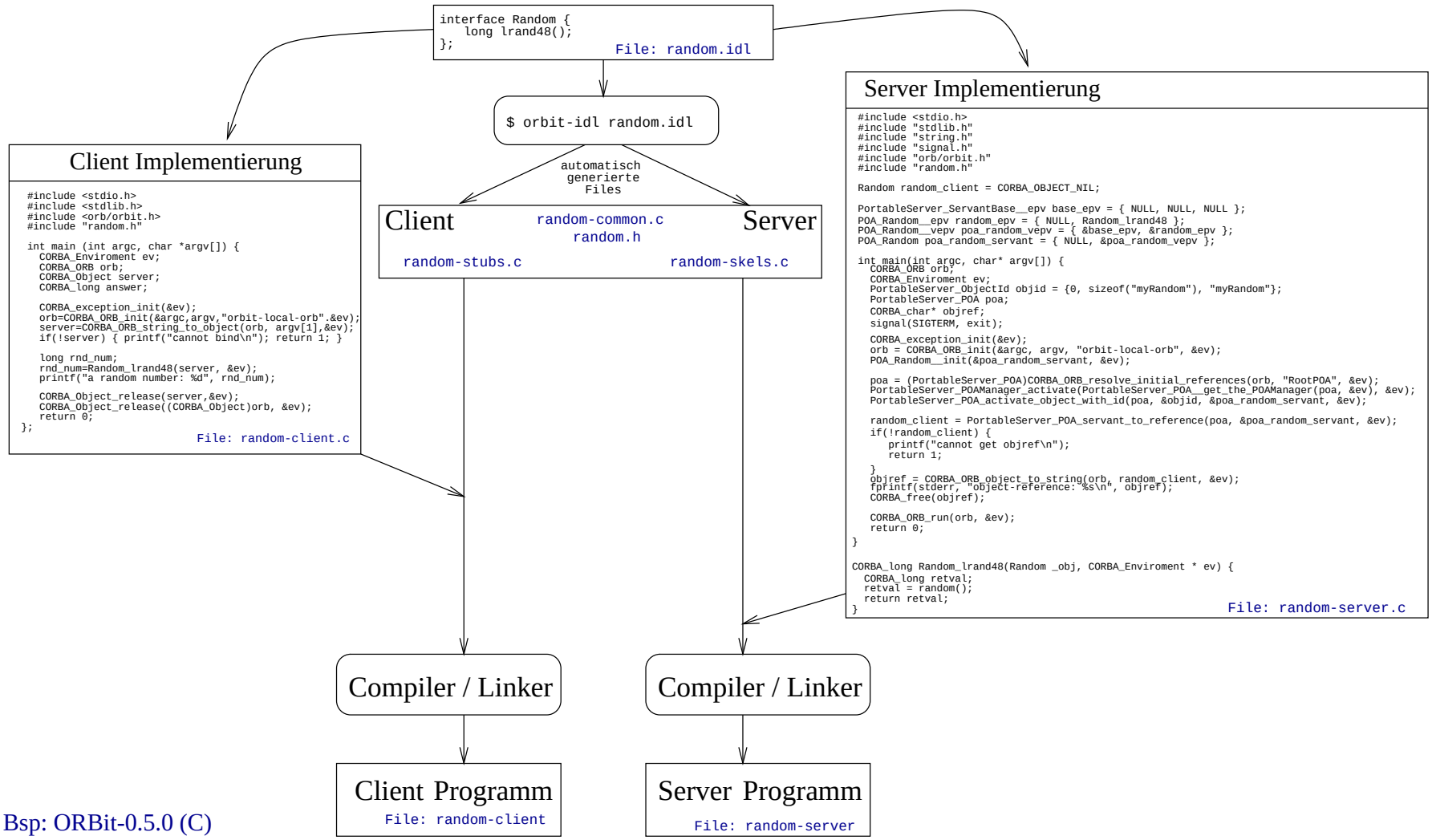
- statische und dynamische Methodenaufrufe
- Verknüpfungen auf Hochsprachenebene
- Ortstransparenz (ohne Programmieraufwand)
- Eingebaute Sicherheit (Dienste: Security, Transation, ...)
- Koexistenz mit existierenen Systemen (*RPC*, *DCOM* ...)

3. Entwurf mit CORBA

Motivation für Programmierer: *Wie komme ich zum Code?*



Beispiel - eine Zufallszahl vom Server erfragen:



Bsp: ORBit-0.5.0 (C)

4. CORBA und Design Patterns

Motivation:

- zuverlässige und “gute” Software-Entwicklung
- schnelle Entwicklungszyklen
- Wartbarkeit
- Technik um Softwareteile oder -komponenten wiederzuverwenden

mittels Design Pattern:

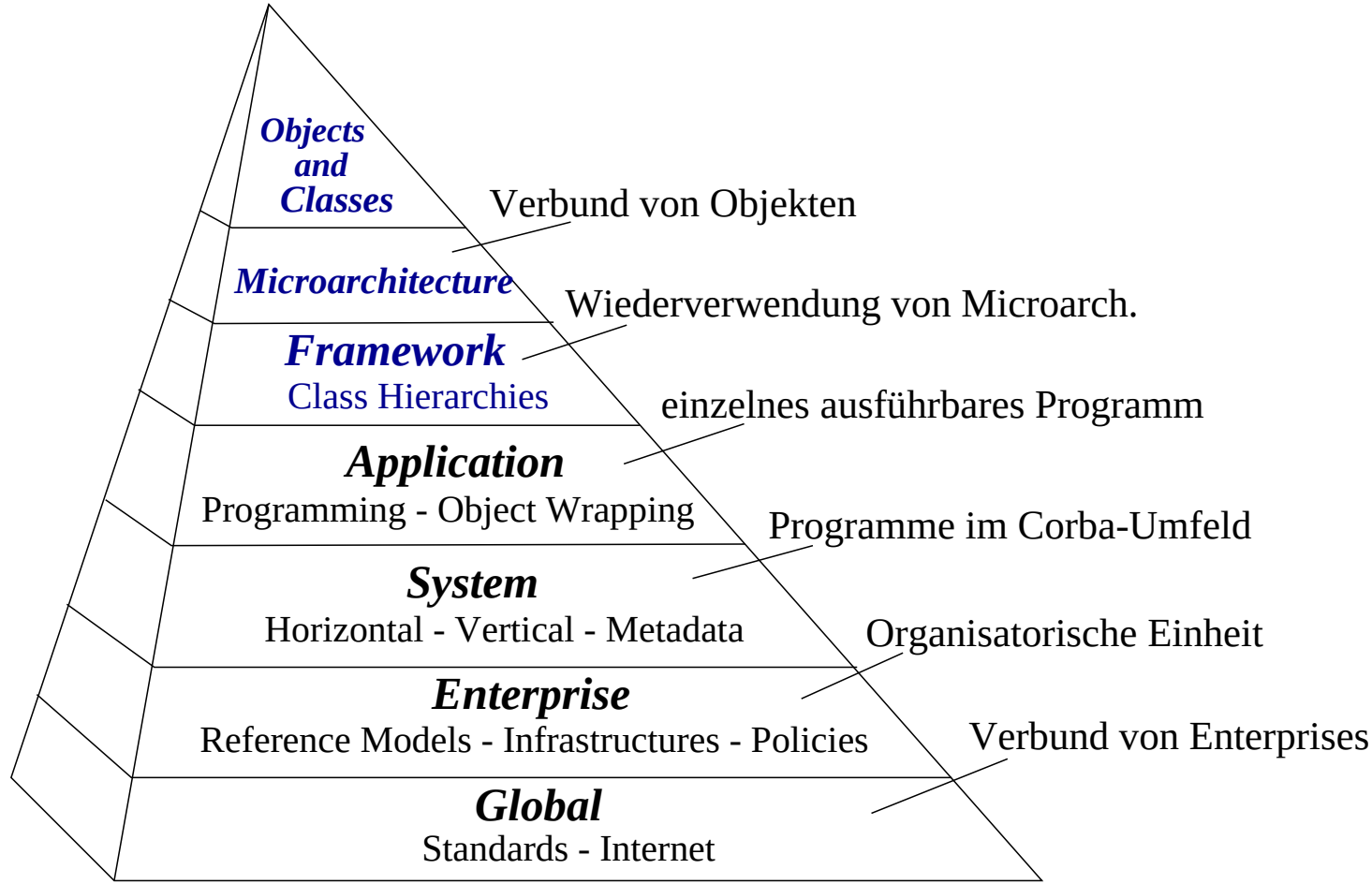
- Darstellung erfolgreicher (OO)-Lösungen zu bestimmten wiederkehrenden Problemen
- Lösungen nicht als Code, sondern in sprachunabhängiger Form

Wie Patterns beschreiben? *(nach Vortrag “Patterns”, Seite 11)*

- Name und Klassifikation
- Zweck — Was tut das Pattern?
- Motivation — anhand eines Szenarios
- Anwendbarkeit — In welchen Situationen diese Pattern benutzen?
- Struktur — Graphische Darstellung der Struktur des Pattern
- Mitwirkende (Pattern) — teilhabende Klassen und Objekte und ihre Aufgabe
- Folgen — aus der Anwendung des Pattern
- Implementation — Einige Tips, Sprachabhängigkeiten, Schwierigkeiten
- Beispiel-Code — zur Illustration
- Anwendungsbeispiel — aus echtem System
- Verwandte Patterns — Unterschiede, Kombinationsmöglichkeiten

Skallierungsmodell

Ansatz: [2] Design Patterns nach *Most Applicable Scale* *("hauptsächliche Anwendungsebene")* in einem Skalierungsmodell ordnen



(blau = nicht weiter beschieben)

Application Design Patterns

- Maximizing Parallism
- Improving Object Implementations
- Modifying Client Stubs and other Tricks

System Design Patterns

- Prinziples of Object-Oriented Architecture
- Fundamental Structural Patterns
- Advanced System Design Patterns
- Using OMG CORBAServices

Enterprise Design Patterns

- Building an Organizational Infrastructure

Global Design Patterns

- Role of Open Systems
- Internet Design Patterns

Skallierungsmodell (2)

Anmerkung:

- genannte Design Patterns orientiert an CORBA
- sind sehr allgemein gehalten

allgemeines Ziel:

- gekapseltes Design
- besseres Objekt-Managment
- Kommunikation und Kooperation mit bestehenden Standards
- Performance-Gewinn
- senken des Netzwerk-Traffic

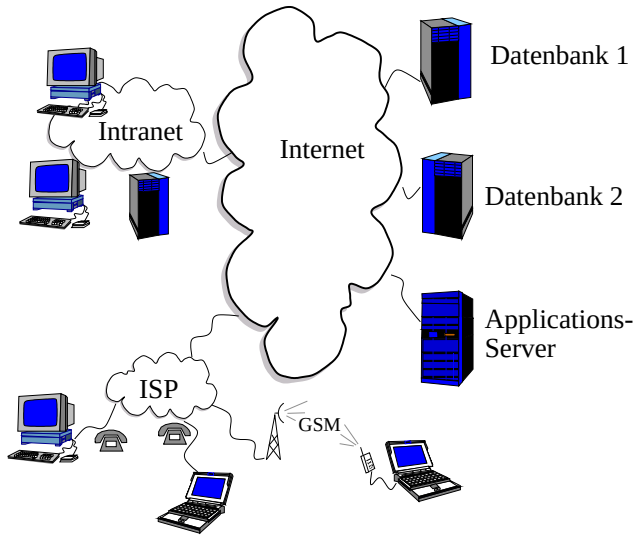
Pattern an folgende Beispiele (einfacht):

- ORBlet
- ORB
- Operating Enviroment
- Naming Service
- Distributed Callback

Global Design Pattern

Application Design Pattern

Bezug zum einleitenden Beispiel wird gesucht ⇒



Global Design Pattern - ORBlet

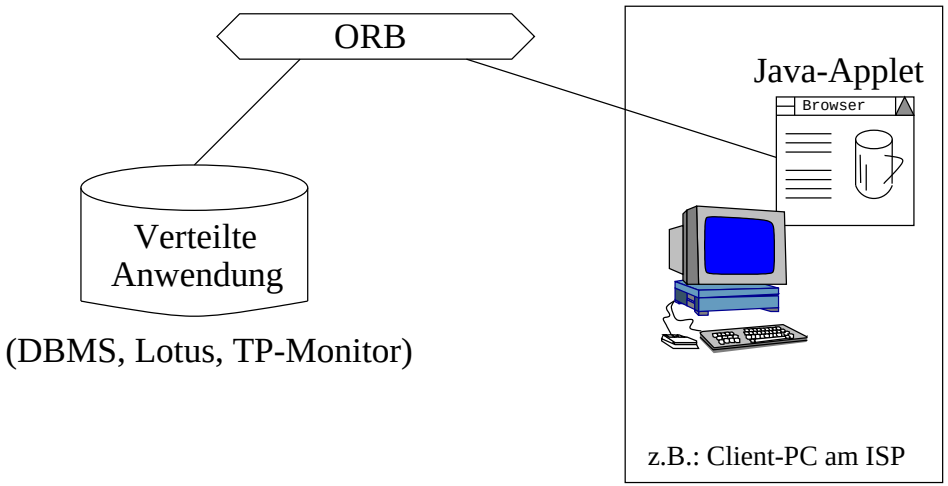
Zweck:

- Zugriff auf *CORBA*-basierende Anwendungen und *CORBAservices* über WebBrowser

Realisierung:

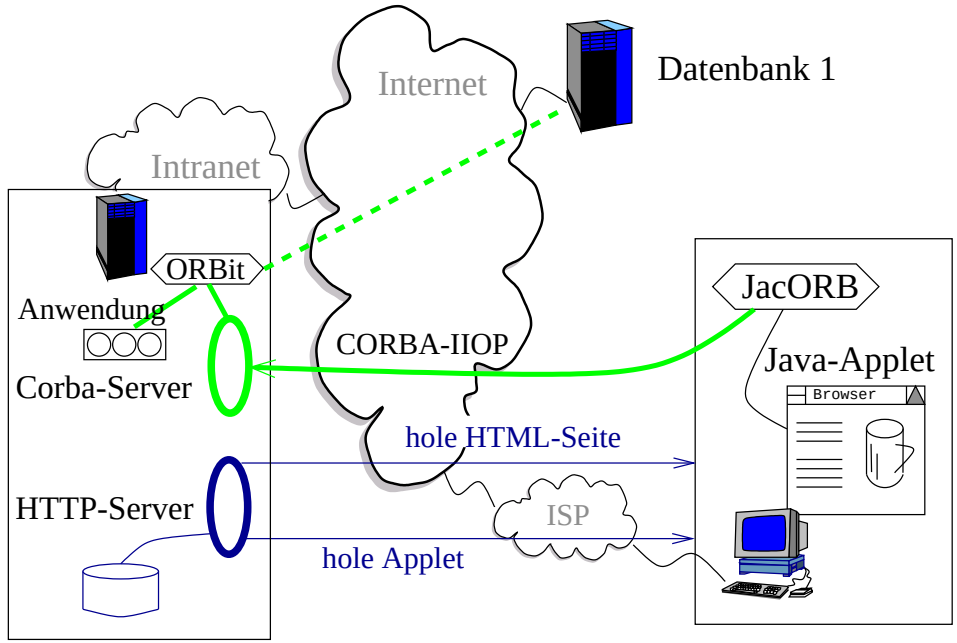
- benutze Java-Applet (enthält *CORBA client stubs*)
- ORB-Kommunikation über *Internet Interoperability Protokoll* (IIOP)
- asynchrone Aufrufe verwenden

Struktur:



Global Design Pattern - ORBlet (2)

Beispiel:



- hole HTML-Seite (benutze Pattern *Standards*, z.B. HTML3.2)
- hole Applet (*Applet* ist ein Pattern, das auf dem Pattern *Java* aufbaut)
- starte Applet (im WebBrowser)
- CORBA-IIOP

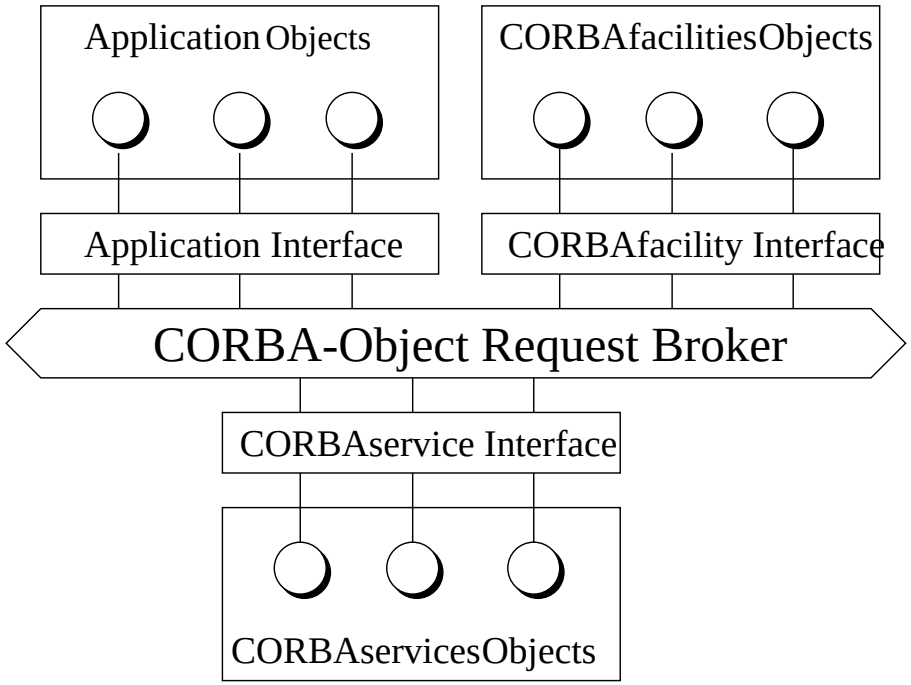
Enterprise Design Pattern - ORB

Zweck:

- (transparentes) verteiltes Arbeiten (mit Objekten) in heterogener Umgebung.
- unabhängig von Betriebssystem, Implementierungssprache und Lokalität

Realisierung:

- ORB als vermittelnder Agent zwischen Client und Server $\rightarrow$ *OMA*



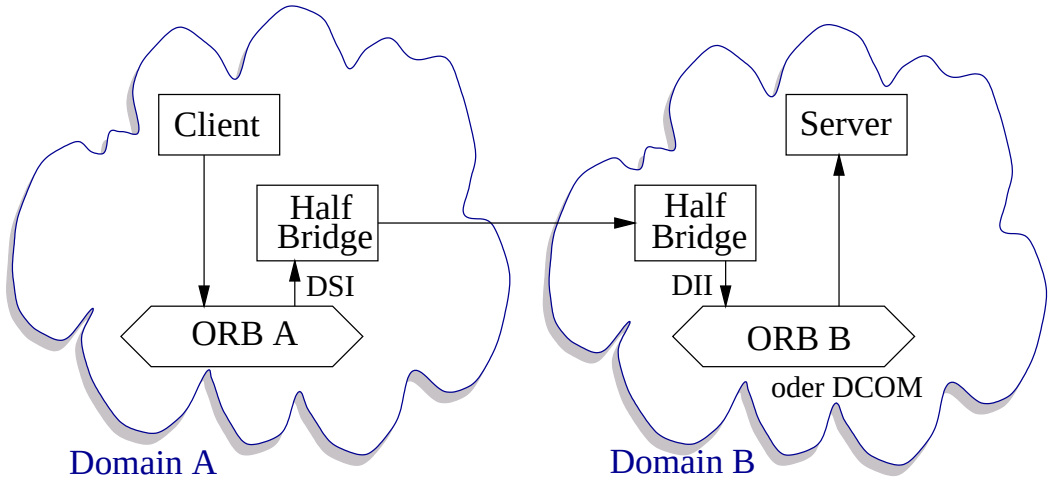
Enterprise Design Pattern - ORB (2)

Folgen:

- separate Kompilation notwendig
- reduzieren der Komplexität der Anwendung
- separieren der Anwendung von der Infrastruktur
- etwas Overhead

Ausblick:

- über Half-Bridge die ORBs transparent verbinden
- damit Lösung: administrativer Bedürfnisse, Diensttypen, Sicherheitsaspekte...



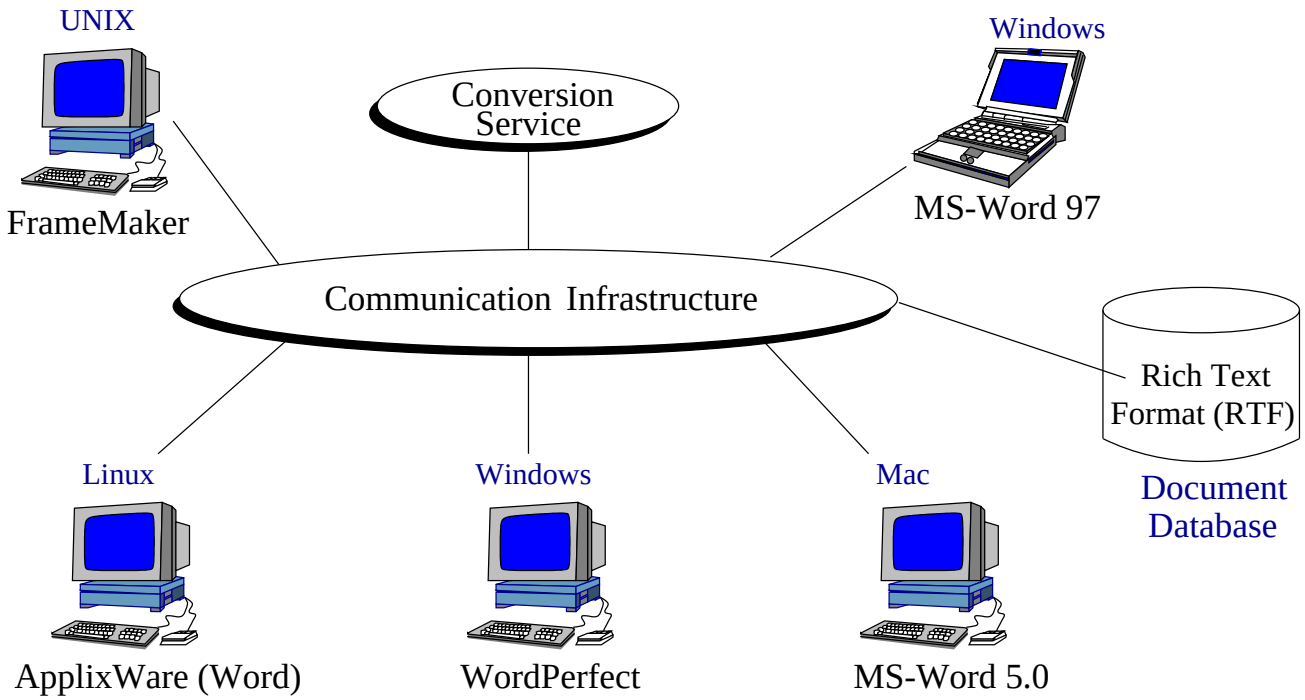
Enterprise Design Pattern - Operating Enviroment

Zweck:

- Interoperabilität zwischen existierenden Produkten

Realisierung:

- benutze Konvertierungs-Service und zentrale Datenbank



Enterprise Design Pattern - Operating Enviroment (2)

Folgen:

- Unabhängigkeit von Hersteller und Produkt
- genereller Informations-Import realisierbar
- Änderungen in Daten verifizierbar
- jede Einheit mit mehr Autonomie
- Unterstützung von speziellen Tools ($\rightarrow$ höhere Produktivität)

Beispiel : einleitender Informationsdienst:

- "Kunde" kann angeben, welches Datenformat er wünscht, oder
- kann Konvertierungs-Service nutzen
- Kommunikation über ORB

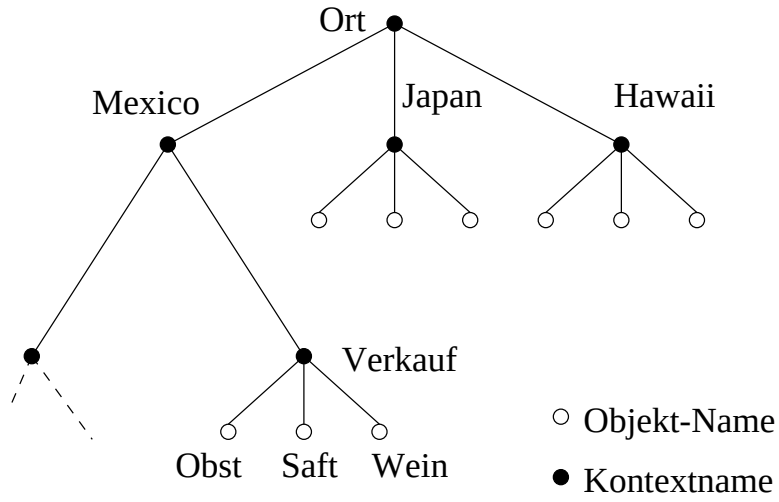
System Design Pattern - Naming Service

Zweck:

- Objekt anhand eines Namens im Namensraum finden bestimmen

Realisierung:

- nutze *repository* zum speichern von *Namen* und *Objekt-Referenz*
- Namensbindung relativ
- zwei Grundoperatoren: *store(bind)* und *retrieve(resolve)*



Ort.Mexico.Verkauf.Saft

zusammengesetzter Name
(compound name)

System Design Pattern - Naming Service (2)

Beispiel:

- implementierter ORB kennt einige "initial services"

Beispiel - Code-Fragment (ORBit-C)

```
int main (int argc, char *argv[]) {
    CORBA_Environment ev;
    CORBA_ORB orb;
    ...
    CosNaming_NamingContext name_srv;   /* name server */
    CosNaming_Name* serv_name;          /* This holds the name of our server */
    CosNaming_Name* serv2_name;         /* This holds the name of our server2 */
    factory fac, fac2;                  /* the server-side factory object */
    ...
    orb = CORBA_ORB_init(&argc, argv, "orbit-local-orb", &ev);
    ...
    /* get the name server from the ORB */
    name_srv = CORBA_ORB_resolve_initial_references(orb,"NameService", &ev);
    ...
    /* store(bind) the name of the object implementation */
    CosNaming_NamingContext_bind(name_srv,serv_name, fac, &ev);
    ...
    /* retrieve(resolve) the reference of the object implementation */
    serv2_name=create_name("Ort.Mexico.Verkauf.Saft");
    fac2 = CosNamingContext_resolve(name_srv, serv2_name, &ev);
    ...
}
```

Application Design Patterns - Distributed Callback

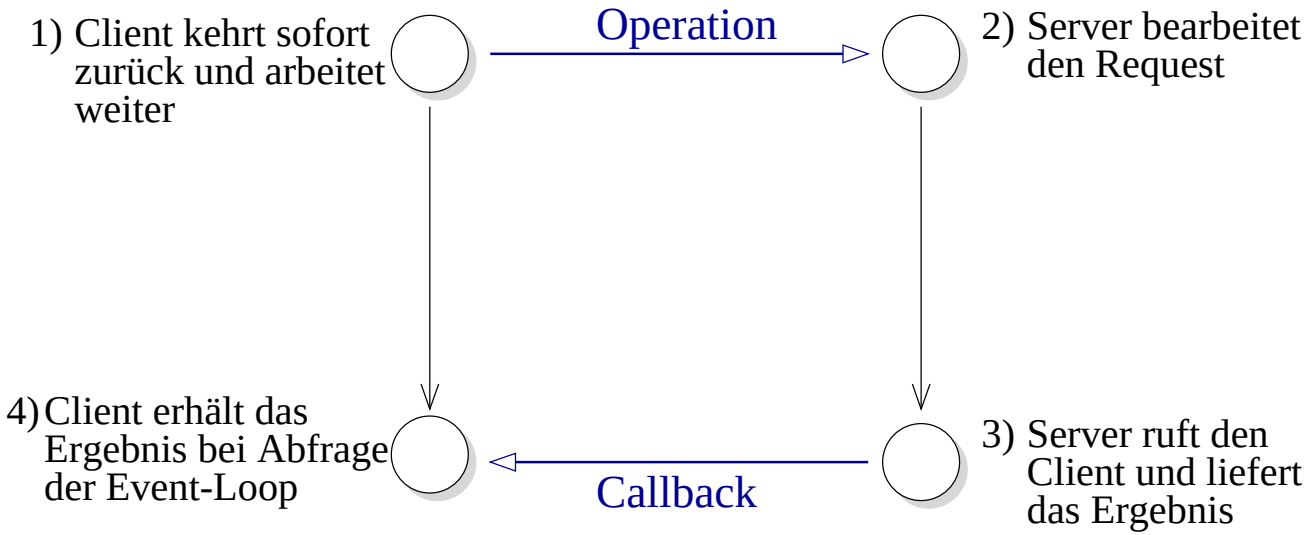
Zweck:

- Client soll, während er auf einen Request wartet nicht blockieren

Realisierung:

- synchrone Operatoren in (asynchrone) oneway-Operatoren umwandeln
- Client mit Callback-Interface ausstatten

Struktur:



Application Design Patterns - Distributed Callback (2)

Folgen:

- Client-Callbacks in verschieder Reihenfolgen
- nur primitive Fehlerbehandlung (keine Exceptions)
- Zustellung (oneway) nicht garantiert

Implementierung:

- Ausgangsoperation (IDL):

```
interface app {
    string get_info( in string my_data);
};
```

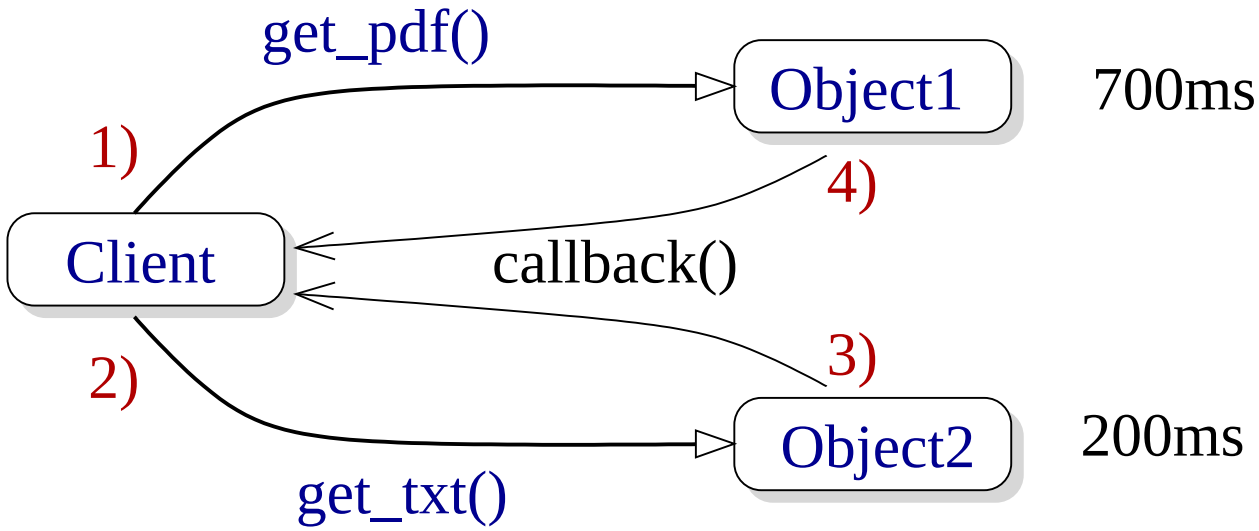
- konvertieren zu:

```
interface app_callback;     /* forward */
interface app { /* this interface defines the server object */
    oneway void get_info( in string my_data, in app_callback my_callback);
};
interface app_callback { /* this interface must be implemented by the client */
    oneway void callback( in string return_value, in long status);
};
```

Application Design Patterns - Distributed Callback (3)

Beispiel:

- Konvertierungs-Service vom Informationsdienst



5. Zusammenfassung

Zu CORBA:

- für viele Betriebssysteme / Hochsprachen (frei) verfügbar
- IDL-Schnittstelle nur mit Syntax

Design Pattern und CORBA:

- Design Pattern nach Anwendungsebene skalliert
- Buch: leider bei gewisse Pattern zu abstrakten Formulierungen
  bzw. mageren Beispiele - damit kein erwünschter Praxisbezug
- Ziel von viele Pattern: Performance / Netzwerk-Traffic verbessern
- Gut: System-unabhängige Dienste vorhanden: (*Naming*, *Event*, *Transaction*, *Lock*...)
  - werden aber nur gering beachtet
- Fraglich: Sicherheitskonzept auf Ebene des verwendeten ORB (Speicherraum)

Quellen

[1] R. Orfali, D. Harkey, J. Edwards, *Instant CORBA*, Addison-Wesley, 1998, ISBN 3-8273-1325-2

[2] T. J. Mowbray und R. C. Malveau, *CORBA Design Patterns*, Wiley Computer Publishing, 1997, ISBN 0-471-15882-8

[3] John Siegel, *CORBA - Fundamentals and Programming*, Wiley Computer Publishing, 1996, ISBN 0-471-12148-7

[4] Folien “Patterns”, Seminar “Softwarearchitektur”, Nov. 2000, www.informatik.uni-freiburg.de

Ein Überblick über einige ORBs gibt http://www.vex.net/~ben/corba/index.html

Abkürzungen

CORBA = Common Object Request Broker Architecture

DII = Dynamic Invocation Interface

DSI = Dynamic Skeleton Invocation

OMA = Object Managment Architecture

OMG = Object Managment Group

ORB = Object Request Broker

RPC = Remote Procedure Call

IDL = Interface Definition Language