Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2007/08

(Vertretung Herr Berger)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java und CORBA

- CORBA als Interoperations-Standard muss an konkrete Programmiersprachen **gebunden** werden
- Seit CORBA 2.2 (1998) gibt es OMG IDL to Java binding sowie Java to OMG IDL reverse binding
  - Java als die wichtigste CORBA-Referenzimplementierung
- Reverse binding interessant für Anbindung von Nicht-Java-Systemen an Java-Systeme über IIOP-Standard von CORBA
- Heute Koexistenz in fast allen Applikationsserver-Produkten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Entwicklung einer CORBA–Anwendung unter JAVA

(aus "Lehrbuch der Softwaretechnik" von Helmut Balzert)

4) Spezifikation der Schnittstelle in CORBA – IDL
5) Übersetzung mittels IDL–Compiler
   Stummel- und Skelettklassen werden erzeugt
6) Implementierung der Operationen der Schnittstelle
7) Rahmenanwendung entwickeln
   Erzeugt Objekt der Klasse
   Objekt wird für Clients zugreifbar gemacht
8) Entwickeln des Clients

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Definition der Schnittstelle mit OMG IDL

```
module SemOrg { // Schnittstelle der Klasse Firma
    interface Firma {
        attribute string Name;
        attribute float Umsatz;
        // Operationssignatur
        float berechneGewinn( in float Kosten );
    };
};
```

SemOrg.idl

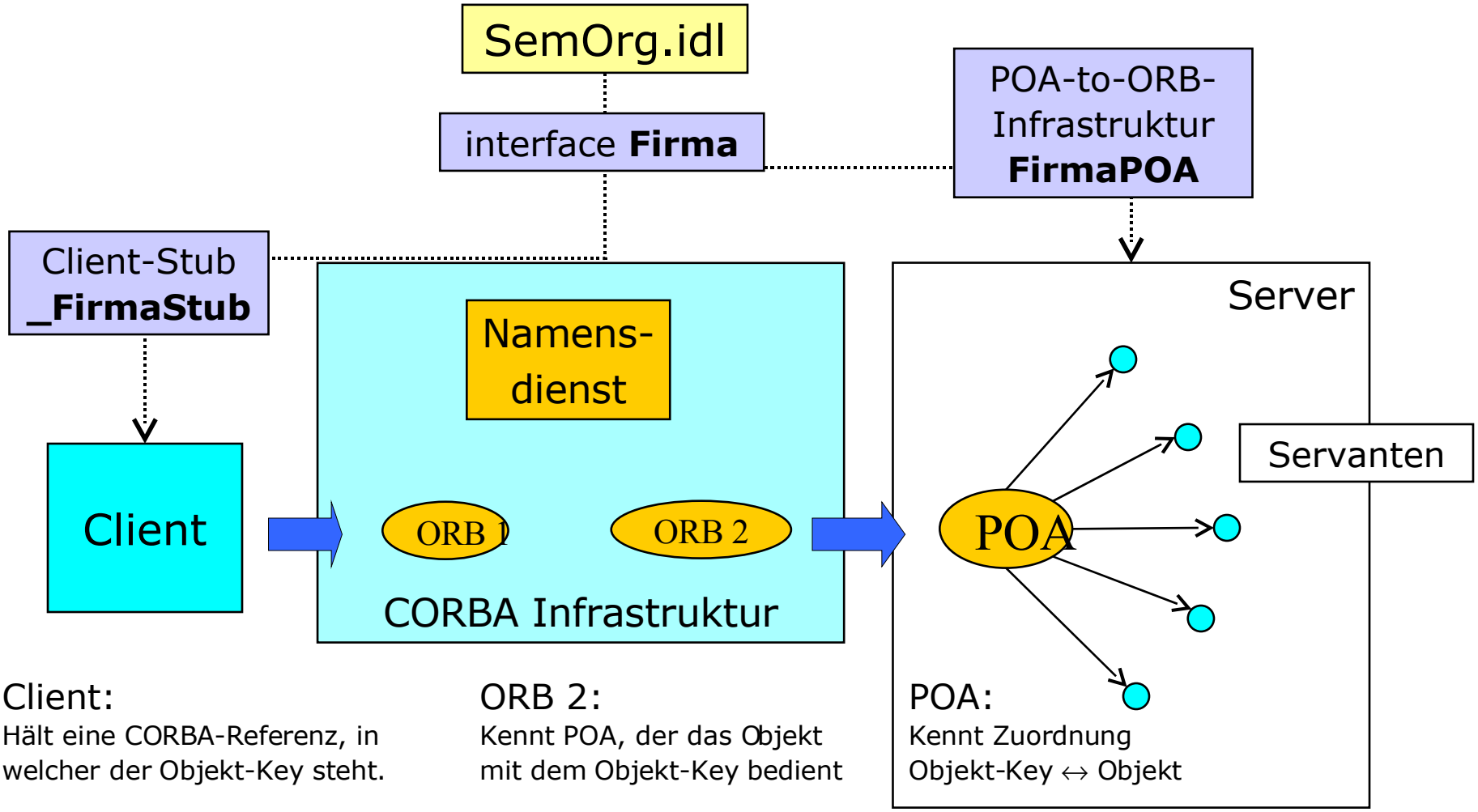
`idlj –f all SemOrg.idl` erzeugt daraus Java-Package **SemOrg**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vom IDL – Compiler erzeugte Klassen

- interface **FirmaOperations**
  - interface Firma als Java-Interface
- interface **Firma** extends FirmaOperations, ...
  - Firma-Operations + CORBA.Object ...
- class **_FirmaStub** extends CORBA.portable.ObjectImpl
  implements SemOrg.Firma
  - voll generierte Stummel – Klasse
- final class **FirmaHolder** implements CORBA.portable.Streamable
- abstract class **FirmaHelper**
  - „Cast" von CORBA.Object zu Firma
- abstract class **FirmaPOA** extends PortableServer.Servant ....
  - portabler Objekt-Adapter
  - generierte Implementierung der ORB-seitigen Kommunikation

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Implementierung des Servanten

```
package SemOrg;
public class FirmaImpl extends FirmaPOA {
    private String derName;
    private float derUmsatz;
    public FirmaImpl() {} // Konstruktor
    public float berechneGewinn(float Kosten)
        { return this.Umsatz – Kosten; }
    // get-/set-Operation für Attribut Name
    public String Name() { return this.derName }
    public void Name(String neuerName)
        { this.derName = neuerName; }
    //analog für Umsatz ...
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Default-Server-Modell: Portable Servant Inheritance Model

Aufgaben des Servers

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- Instanz des Servanten **FirmaImpl** anlegen und in der CORBA-Struktur verankern: als POA registrieren und den POAManager aktivieren.
- CORBA-Objektreferenz des Servanten besorgen
- Objektreferenz auf einen Namens-Kontext von Namens-Server der ORB-Infrastruktur holen und den Servanten als „eineFirma" registrieren.
- Auf Anfragen warten

Aufgaben des Client

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- CORBA-Objektreferenz auf den Servanten „eineFirma" über den Namens-Kontext besorgen
- Dienste des Servanten in Anspruch nehmen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Serverkomponente

```
package SemOrg;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class SemOrgServer {
  public static void main(String[ ] args) {
    try {
      // Kontakt zur ORB-Infrastruktur herstellen: ORB-Referenz holen
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      // POA-Referenz vom ORB besorgen und POA aktivieren
      POA poa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      poa.the_POAManager().activate();
      // Servanten anlegen als reales Java-Objekt
      FirmaImpl impl = new FirmaImpl();
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
    // CORBA-Objektreferenz auf den Servanten besorgen und „casten“
    org.omg.CORBA.Object implObjServer = poa.servant_to_reference(impl);
    Firma eineFirmaS = FirmaHelper.narrow(implObjServer);
    // vom ORB Referenz auf Namensdienst-Server besorgen und „casten“
    org.omg.CORBA.Object ncObj =
          orb.resolve_initial_reference("NameService");
    NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
    // Servanten unter dem Namen "eineFirma" im Namensdienst bekannt
    // machen. Verbindung zwischen "eineFirma" und dem Java-Objekt impl
    // kennt nur poa
    ncRef.rebind(ncRef.to_name("eineFirma"), eineFirmaS);
    System.out.println(„SemorgServer gestartet ...“);
   } // end try
   catch (Exception e) { ... }
  } // end main
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Client-Komponente

```
package SemOrg;
import org.omg.CosNaming.*;

public class SemOrgClient {
  public static void main(String[ ] args) {
    try {
      // Verbindung zur ORB-Infrastruktur
       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

      // vom ORB Referenz auf Namensdienst besorgen
      org.omg.CORBA.Object ncObj = orb.resolve_initial_reference("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
    // CORBA-Objektreferenz auf das Servanten-Objekt „eineFirma“ besorgen
    org.omg.CORBA.Object firmaObj = ncRef.resolve_str("eineFirma");
    Firma eineFirmaC = FirmaHelper.narrow(firmaObj);

    System.out.println(“Handle auf das Server-Objekt”+eineFirmaC);

    //Aufrufe durchführen
    System.out.println(eineFirmaC.Name());
        ...
    }
    catch (Exception e) { ... }
  }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CORBA in der industriellen Anwendung

Hauptanwendungsfeld: Ersetzen von Sockets und RPC in Anwendungen, die über mehrere Server verteilt sind

höhere Abstraktionskonzepte vor CORBA 3 kaum bedient

- Kooperation beim Entwickeln verteilter Anwendungen über Teamgrenzen bisher kaum unterstützt

Weiter gehende Konzepte hat OMG schon lange im Visier

Object Management Architecture (OMA)

erste Standardisierungen mit CORBA 2, seit 1997 im Focus der OMG

mit CORBA 3 ins Zentrum gerückt

3 neue Standardisierungsfelder

- Spezifikation von (grundlegenden) Objektdiensten (CORBAservices)
- Spezifikation von (häufig benötigten) Anwendungsbestandteilen (CORBAfacilities)
- Spezifikation von Anwendungsobjekten

CORBA Komponentenmodell (CCM)

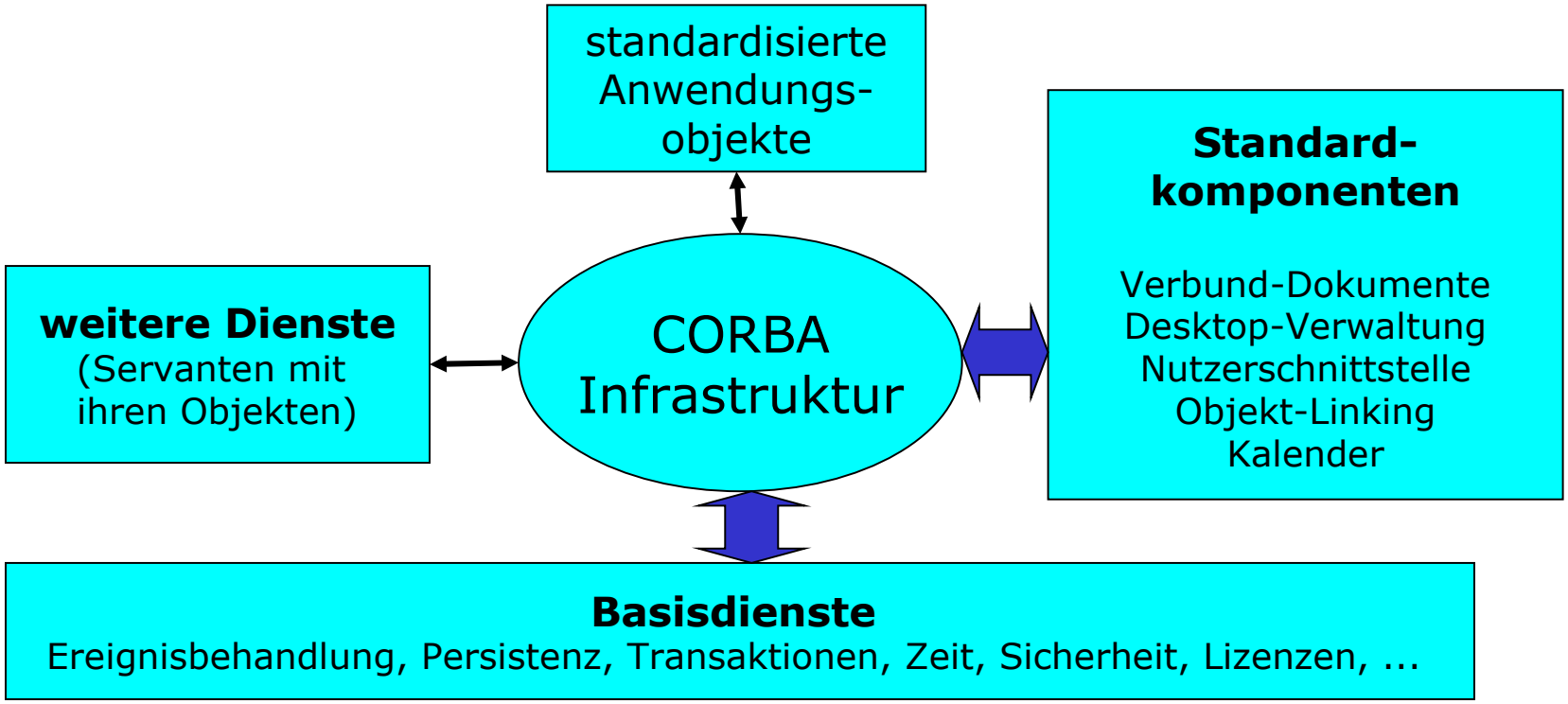
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Idee: Unterteilung der Dienste in mehr oder weniger wichtige

Bereitstellung von Dienst-Servanten für wichtige Funktionalitäten, die in einen **Komponentenrahmen** (component framework) „eingesteckt" werden können

Bsp: Geschäftsfeld-Objekte (business objects) = Objekte, die direkte Geschäftsprozessabstraktionen repräsentieren

steht erst ganz am Anfang der Entwicklung



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Auf dem Weg zu CORBA 3

Als Ganzes formal erst Ende 2002 freigegeben

Teilstandards Stück für Stück bereits in CORBA 2.3 ... 2.6 integriert

CORBA 2.3

- Objekte als Wertparameter
- XML-Abbildungen
- Java-RMI als Schnittstellenmodell in CORBA
  - Java-CORBA-Koevolution
  - Java als wichtigste Plattform für Implementierung der Standards

CORBA 2.4

- Objektreferenzen als URL
- asynchrone Botschaften
- Minimal- und Realzeit-CORBA

CORBA 2.5: Fehlertoleranz- und Abbruch-Standards

CORBA 2.6: Sicherheitsstandards

CORBA 3: **Meilenstein** ist Komponentenmodell (CCM)

- im Wesentlichen fertig seit Ende 2001

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Basisdienste (CORBAservices)

Fokus auf fundamentale Bausteine **jeder** Implementierung, welche die Komponenten-Infrastruktur zur Verfügung stellen sollte

- z.B. Ereignisbehandlung, Transaktionsabwicklung, Namensverwaltung
- derzeit 16 Basisdienste spezifiziert

2 Kategorien:

- Dienste zur Unterstützung unternehmensweiter verteilter Anwendungen
  - nutzen typischerweise CORBA-Objekte als Moduln und CORBA als Kommunikations-Middleware
  - grob granulare Ebene, CORBA Kommunikations-Infrastruktur als „Objektbus"
- Dienste zur Unterstützung fein granularer verteilter Anwendungen
  - Bedeutung nimmt ab, da zu hoher Komplexitätsgrad
  - Ausnahme: persistent state service (PSS) als einer der Pfeiler des CCM

große CORBA-Anwendungen nutzen oft nur wenige Basisdienste

- verfügbare CORBA-Plattformen bieten deshalb oft nur Implementierungen einiger Basisdienste an

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Dienste zur Unterstützung grob granularer verteilter Anwendungen

Namensdienst (name service)

- Abbildung intern verwendeter UUID auf externe Bezeichner
- Namens**-Kontexte** und Kontext-**Hierarchien**
  - vergleichbar zu Verzeichnisstrukturen

Händlerdienst (trader service)

- Verfeinerung des Namensdiensts (white vs. yellow pages)
- Anbieter veröffentlichen Dienstangebote per **Registrierung**
- Nutzer finden Angebote über Händlerdienst per **Beschreibung**
- Händlerdienste organisieren Angebote in Handels**-Kontexten**
- Standardisierte Methoden zur Suche in den Angeboten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ereignisdienst (event service)

- Verteilung der E.-**Objekte** von E.-**Erzeugern** (event supplier) an E.-**Konsumenten** (event consumer)
- E.-Objekte sind unveränderbar, wenn einmal erzeugt
  - strikt unidirektionaler Informationsfluss
- E.-**Kanäle** (event channel) entkoppeln Erzeuger und Konsument
- E. können getypt sein (OMG IDL)
- Kanäle können Ereignisse nach ihrem Typ filtern
- Push- und Pull-Methoden werden unterstützt

Benachrichtigungsdienst (notification service)

- Erweiterung des Ereignisdiensts um einige kritische Merkmale
  - Dienstqualität, Administration
  - dynamische E.-Filterung, Filterung auf verschiedenen Ebenen
- technisch kein Basisdienst, sondern Standardkomponente
  - gemeinsamer Standard mit Telecomm. Domain Task Force

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Transaktionsdienst (object transaction service, OTS)

- einer der wichtigsten Bausteine für verteilte Anwendungen
- standardisiert seit 1994
- wird von den meisten ORB-Produkten und J2EE-Servern unterstützt
- **Eingebettete Transaktionen** nur optional
  - Transaktionshülle um Folge von Operationen
  - erforderlich zur unabhängigen Entwicklung auf verschiedenen Hierarchie-Ebenen
  - (noch) nicht standardisiert, weil kaum eines der heute ex. Transaktionssysteme so etwas vorsieht
- Verwaltung eines (objektspez.) aktuellen Tr.-**kontexts** durch OTS
  - Objekte müssen dazu die Schnittstelle *TransactionalObject* implementieren
  - Methoden *begin*, *commit*, *rollback* operieren auf dem Kontext
- Objekte unter Transaktionskontrolle registrieren sich beim OTS-Koordinator-Objekt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Transaktionsdienst (Fortsetzung)

- Ressourcen müssen die Schnittstelle *Resources* implementieren
  - Koordinator wickelt darüber 2-Phasen-commit-Protokoll ab
  - bekanntes Problem der Deadlock-Gefahr
    - 3-Phasen-Protokoll vermeidet diese, ist aber teurer
- heute weit verbreitet: Transaktionskontrolle nicht als separater Dienst, sondern als Kontextkontrolle **innerhalb** eines Anwendungsservers
  - Diese Abstraktion wird vom CCM abgedeckt

Sicherheitsdienst (security service)

- erforderlich, wenn sich verteilte Anwendung über mehrere Vertrauensbereiche (trusted domains) erstreckt
- spezifiziert in **CORBAsecurity**
- Authentifizierung, sichere Kommunikation, Zertifizierung
- volles Spektrum wird derzeit von kaum einem Produkt unterstützt
  - meist nur SSL-basierte Sicherheit
  - unterstützt einfache Sicherheit, aber keine Zertifikate

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Dienste zur Unterstützung fein granularer verteilter Anwendungen

Nebenläufigkeits-Kontrolldienst (concurrency control service)

- Synchronisierung nebenläufiger Zugriffe auf Ressourcen
- *read* (nicht exklusiv lesen), *upgrade* (exklusiv lesen), *write*
- Geschützte Ressourcen verwalten dazu Schlossmengen (lock sets), die von einem Koordinator-Objekt erzeugt und verteilt werden

Lizenzdienst (licensing service)

- Verwaltung von Objektlizenzen, Abrechnung von Gebrauchsgebühren für Objekte
- Unterstützung verschiedener Lizenzmodelle
- 2 Schnittstellen: *Lizenzdienst-Manager* (LDM) und *Lizenzdienst*
- Objekt unter Lizenz (OL) erfährt über LDM, unter welchen Bedingungen seine Nutzung legitimiert ist
  - OL fordert vom LDM Referenz auf entsprechendes (hersteller-spezifisches) Lizenzdienst-Objekt (LDO) an

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Lizenzdienst (Fortsetzung)

- OL informiert LDO über **Kontext** der Lizenzanforderung
- LDO prüft, welche Nutzung des OL in dem Kontext legitim ist
  - LDO veranlasst Übergang von OL in erlaubten Zustand (ggf. Demo-Modus, Probe-Modus)
- OL informiert LDO über Ende der Nutzung

aktuelle Lizenzgestaltung also gekapselt zwischen OL und LDO

zwischen beiden kann auch statistisch relevante Information ausgetauscht werden

- Nutzerprofile, Lizenzdauer und –ablauftermine

Zeitdienst (time service)

Synchronisierung der Uhren in verteilten Systemen

Korrelation innerhalb sinnvoller Fehlerschranken, um zeitliche Kausalitäten über Systemgrenzen hinweg zu erhalten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Basisdienste (Fortsetzung)

Lebenszyklusdienst (lifecycle service)

- Verwaltung von Objekten (Erzeugen, Kopieren, Löschen, Verschieben) oder Gruppen von Objekten
- unterstützt Objekterzeugung durch Factory-Objekte
  - Registrierung, Wiederverwendung letzterer
- Objektverwaltung mit Referenzzählern in verteilten Anwendungen oder mit verteiltem garbage collection wird nicht unterstützt
  - Grund: verteiltes garbage collection in fehleranfälliger Umgebung (Maschinen- oder Netzwerkausfall) ist sehr kompliziert, braucht Transaktionskontext
  - kein Problem beim Einsatz von CORBA als Kommunikations-Middleware, da dort Objekte gewöhnlich Serverobjekte mit unbegrenzter Lebensdauer

Beziehungsdienst (relationship service)

- Erzeugen, Löschen und Verwalten von Beziehungen zwischen Objekten, Navigation über Beziehungen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz-Dienst (persistent state service, PPS)

- Persistenz = Eigenschaft eines Objekts, das Programmende zu überleben
- CORBA 2: Persistenzobjekt-Dienst (persistence object service, POS)
  - seit 1994, erste Implementierungen 1996
  - unterspezifiziert: konkrete Speicheranforderung war anwendungsspezifisch gelöst
- CORBA 3: Ablösung durch Persistenzzustands-Dienst
- Grundlegender Ansatz: Trennung von persistentem Objekt und Persistenzmechanismus
  - Dateien, Datenbanken
  - strukturierte Speicher (Containerdokumente)
- sehr einfache Schnittstelle: Speichern und Laden eines Objekts
- drei problematische Objekteigenschaften:
1. Objekte haben Identität, sind nicht referenziell transparent
   - Problem beim mehrfachen Speichern / Laden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz-Dienst (Fortsetzung)

1. Objekte können sich aufeinander beziehen (Objekt-Web)
   - Beziehungen müssen mit gespeichert werden
   - wesentliche und flüchtige Beziehungen
   - Probleme beim Mehrfachspeichern (RAID etc.)
2. Objekte sind Einheiten der Datenkapselung
   - Sicherung der Integrität von Objekten auf dem Speichermedium
   - Schutz vor Manipulation unter Umgehung der Objekt-Schnittstelle

POS löste Probleme durch Kooperation zwischen Objekt und Persistenzdienst über ein Protokoll

PSS: explizite Deklaration, welche Objektteile wie zu speichern sind

neue OMG Beschreibungssprache für solche Deklarationen (**persistent state description language, PSDL**)

- Spezifikation verschiedener abstrakter und konkreter Speichertypen (analog Schnittstellen und Klassen in Java)
- Spezifikation entsprechender Factories

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Auslagerungsdienst (externalization service)

- Linearisierung / Delinearisierung von Objekten
  - zueinander invers (erzeugt Objektkopie)
  - keine referenzielle Integrität
    - Wertkopie von Teilobjekten
    - Referenzen **nur** über ORB Referenzmechanismus
- zum Datenexport von Objekten in Dateien und Streams
- Schnittstelle *Streamable* des auszulagernden Objekts (AO)
- wird von Strom-Objekt gerufen, das selbst Schnittstelle *Stream* implementiert
  - über *externalize_to_stream* Methode des AO
  - erzeugt daraus ein lineares Objekt (LO), das Schnittstelle *StreamIO* implementiert
- es können ganze Graphen von Objekten ausgelagert werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eigenschaftendienst (properties service)

- dynamisches Binden von Eigenschaften an Objekte
- keine semantische Interpretation der Eigenschaften
- Schnittstelle *PropertySet* mit Methoden *add*, *modify*, *delete*
- diese können normal, read-only (löschbar, schreibgeschützt), fixed-normal (nicht löschbar) oder fixed-read-only sein

Anfragedienst (object query service)

- Dienst zum Auffinden von Objekten nach Attributen
- ähnlich Händlerdienst, sucht aber Objektinstanzen
- Unterstützt Object Query Language (OQL-93) der Object Database Management Group) sowie SQL mit Objekterweiterungen
- Definiert Schnittstelle eigener *Sammeldienst*-Objekte
  - Semantik geordneter Mengen (add, remove, enumerate)
  - spezielle Schnittstelle *Iterator* zur Auswertung solcher Objekte
- Anfrage-Objekt kapselt die Anfrage, welche in zwei Schritten beantwortet wird: Vorbereitung und Abarbeitung der Anfrage

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Anfragedienst (Fortsetzung)

- Vier Objekttypen:
  - Anfrage-Objekte (query object, QO) und Sammelanfragen (querable collections, QC)
  - Anfrage-Auswerter (query evaluator, QE) wertet QO oder QC aus und erzeugt Ergebnis-Sammelobjekt
  - Anfrage-Manager (query manager, QM) erzeugt QO oder QC und schickt sie an QE zur Beantwortung
- Das Objekt, das Anfrage generiert, benutzt *Iterator*-Schnittstelle zur Auswertung der Antwort

Sammeldienst (object collections service)

- Möglichkeit zum Bilden von Sammeltypen verschiedener Topologien, z.B. Mengen (bags, sets), Schlangen (queues), Listen (lists) oder Bäume (trees), entsprechend der Smalltalk-Klassifikation
- unklar, ob das nicht lieber auf Objektebene realisiert sein sollte
  - existieren effiziente Implementierungen dieser Datentypen auf Bibliotheksebene

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Standardkomponenten (CORBAfacilities)

Standardisierung von häufig benötigten Anwendungsbestandteilen

Komponentenrahmen zur einfachen Integration von Anbieterlösungen

Abgrenzung von Bereichen horizontal oder vertikal

horizontal: Fokus auf generellem Anwendungsmodell

Standards für Nutzerschnittstellen, System- und Aufgabenverwaltung

vertikal: Focus auf bereichsspezifischen Einsatzfeldern

im Rahmen von OMG SIG‘s oder Domain Task Forces

Standards zur Integration häufig benötigter Dienste als „Plugins“ in bestehende Komponentenrahmen (component frameworks)

- vereinfacht und standardisiert das Vorgehen bei der Integration von Komponenten verschiedener Anwender

Einteilung der Rahmen nach horizontalen (allgemeinen) oder vertikalen (bereichsspezifischen) Gesichtspunkten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Horizontale (generale) Standardkomponenten

Fokus auf generellem Anwendungsmodell

- Standards für Nutzerschnittstellen, System- und Aufgabenverwaltung

OMG hatte hier ursprünglich folgende Rahmen im Auge

- **Benutzerschnittstelle** (user interface)
- **Informationsverwaltung** (information management)
- **Systemverwaltung** (system management)
- **Aufgabenverwaltung** (task management)

wird heute nur noch wenig vorangetrieben und stärker auf übergreifende Dienste konzentriert, die aus branchenspezifischen Standards herrühren

- Internationalisierung, mobile Agenten, Zeit- und Druckdienst-Standards
- keine Standard-Komponenten, sondern Komponenten-Standards

Standardisierungsbemühungen der ursprünglichen Bereiche spielen praktisch so gut wie keine Rolle

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vertikale Standardkomponenten

ursprünglich Fokus auf Basisfunktionalität für unterschiedliche Marktsegmente

Ergebnisse bekommen zunehmend segmentüberschreitende Bedeutung

- Komponenten-Standards statt Standardkomponenten

Ausgehandelt in Aktivitäten verschiedener Domain Task Forces

- business enterprise integration
- command, control, communications
- Finanzbereich
- Bereich Gesundheitsvorsorge
- Lebenswissenschaften
- Produktionsstrukturen
- Telekommunikation usw.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Sun und Java

- Java: Geschichte und Konzepte
- Wichtige von Java unterstützte Grundkonzepte
- Die J2EE-Architektur
- Java Komponentenmodelle
- Java Servlets / Java Server Pages (JSP)
- Enterprise Java Beans
- Ein Beispiel

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java: Geschichte und Konzepte

große Erfolgsstory

- 1995/96 erste Anfänge (Sun Microsystems)
- heute (2003) eines der am häufigsten gebrauchten Schlagworte
- objektorientierte Programmiersprache, aber Magnet war Konzept von Applets, Mini-Applikationen und Funktionalität innerhalb von Webseiten

2 Ansätze, womit Java wirklich zur Killerapplikation wurde

- **Sicherheitskonzept**
  - Applet wird doppelt geprüft (Übersetzungszeit und Ladezeit)
  - strenge Sicherheitsregeln (security policies), die mit keiner anderen Programmiersprache erreicht werden
  - Sicherheit auch im compilierten Code eines JIT-Compilers
  - Sicherheitsaussagen durch Erzeugerzertifikate möglich
    - „signed applets“ (unter Nutzerkontrolle)

- **Java virtual machine**
  Plattformunabhängigkeit
  großer Vorteil für Applikationen, die übers Web verteilt werden
  Vorteil vor allem im Standard
    - Java class-File Format und Java JAR-Archiv-Format
- beides nicht neu, jedoch in der Kombination und zu diesem Zeitpunkt durchschlagend

Java 2 (seit 1998)

- Fokus auf Applets aufgegeben
  Applets heute nur noch marginal
- Plattform-Editionen
  Funktionalitätsbündel für verschiedene Klassen von Nutzern
    - J2SE mit *JavaBeans* als Plattform für Einzelanwendungen
    - J2EE mit *Enterprise JavaBeans* (EJB) als Serverplattform (seit Ende 1999)
    - J2ME für mobile und eingebettete Anwendungen
- Formalisierung der Bezeichnungen Laufzeitumgebung (JRE), Entwicklungsumgebung (JDK) und Referenzimplementierung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java 2 (Fortsetzung)

Referenzimplementierung der J2SE von Sun auf der Basis der HotSpot-JVM

- J2EE als Standard mit Implementierungen von verschiedenen (unabhängigen) Anbietern
  - (nicht laufzeitoptimierte) Referenzimplementierung von Sun als Beispiel-Implementierung im Quellcode verfügbar
- Prüfung der Unterstützung von Standards durch Kompatibilitätstest-Reihen
- Java BluePrints als Sammlung von Design-Richtlinien und Mustern, um spezielle Technologien zu unterstützen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige von Java unterstützte Grundkonzepte

- **Methoden** (Verhalten, behavior) und **Attribute** (Status, state)
- **Schnittstellen:** Es können Interfaces und abstrakte Klassen definiert werden, die später (mittels ‚*implements*') implementiert werden sollen
  - Mehrfachvererbung von Schnittstellen (ohne Status und Verhalten)
- **Klassen:** Implementierungen von Schnittstellen. Es können von bestehenden Klassen spezielle Unterklassen (mittels ‚*extends*') abgeleitet werden.
  - Einfachvererbung von Implementierungen
  - vermeidet das Diamant-Problem
  - unveränderbare Implementierungen (final class, method, attribute)
- **Pakete** und **Pakethierarchien** als Modularisierungskonzept jenseits von Klassen
  - Namensgebung und Namensräume auf dieser Basis
  - keine Unterstützung von Mehrfachversionen
  - company-name.productname Präfix als Standard
  - Namensraum-Importe

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Sichtbarkeitsklassen** von Attributen und Methoden (default, public, protected, private)
- **Ausnahmebehandlung** (exception handling): Es besteht die Möglichkeit, über Ausnahmen (mittels ‚try-catch'-Blöcken) vom Standard-Kontrollfluss abzuweichen
- **Threads und Synchronisation:** Nebenläufige Programmabläufe lassen sich mit Threads erzeugen und synchronisieren
- **Garbage Collection:** Nicht mehr referenzierte Objekte werden automatisch auf kontrollierbare Weise (‚*finalize*') zerstört

  Anwender hat darauf aus Sicherheitserwägungen keinen Einfluss
- **Objektserialisierung:** Objekte, welche die Schnittstelle *Serializable* implementieren, können in einen Datenstrom geschrieben oder aus einem solchen aufgebaut werden (z.B. Speichern in eine Datei)
- **Ereignisse (events):** werden einige Folien später genauer besprochen