

Vorlesung Software aus Komponenten

3. Komponenten-Modelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2008/09

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

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

Persistenz-Dienst (Fortsetzung)

2. Objekte können sich aufeinander beziehen (Objekt-Web)
 - Beziehungen müssen mit gespeichert werden
 - wesentliche und flüchtige Beziehungen
 - Probleme beim Mehrfachspeichern (RAID etc.)
 3. 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

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.

Zeitdienst (time service)

- Synchronisierung der Uhren in verteilten Systemen
- Korrelation innerhalb sinnvoller Fehlerschranken, um zeitliche Kausalitäten über Systemgrenzen hinweg zu erhalten

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 (löschar, schreibgeschützt), fixed-normal (nicht löschar) 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

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

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

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

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.

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

3.4. Java

Java: Geschichte und Konzepte

Java: Geschichte und Konzepte

große Erfolgsstory

- 1995/96 erste Anfänge (Sun Microsystems)
- Um 2000 eines der am häufigsten gebrauchten Schlagworte
- objektorientierte Programmiersprache, aber Magnet war Konzept von Applets, Mini-Applikationen und Funktionalität innerhalb von Webseiten

Zwei 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)

3.4. Java

Java: Geschichte und Konzepte

- **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 als Serverplattform (seit Ende 1999)
 - J2ME für mobile und eingebettete Anwendungen

3.4. Java

Java: Geschichte und Konzepte

Java 2 (Fortsetzung)

- Formalisierung der Bezeichnungen Laufzeitumgebung (JRE), Entwicklungsumgebung (JDK) und Referenzimplementierung
- 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

3.4. Java

Unterstützte Grundkonzepte

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)

3.4. Java

Unterstützte Grundkonzepte

- **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
- **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

3.4. Java


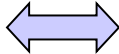
Unterstützte Grundkonzepte

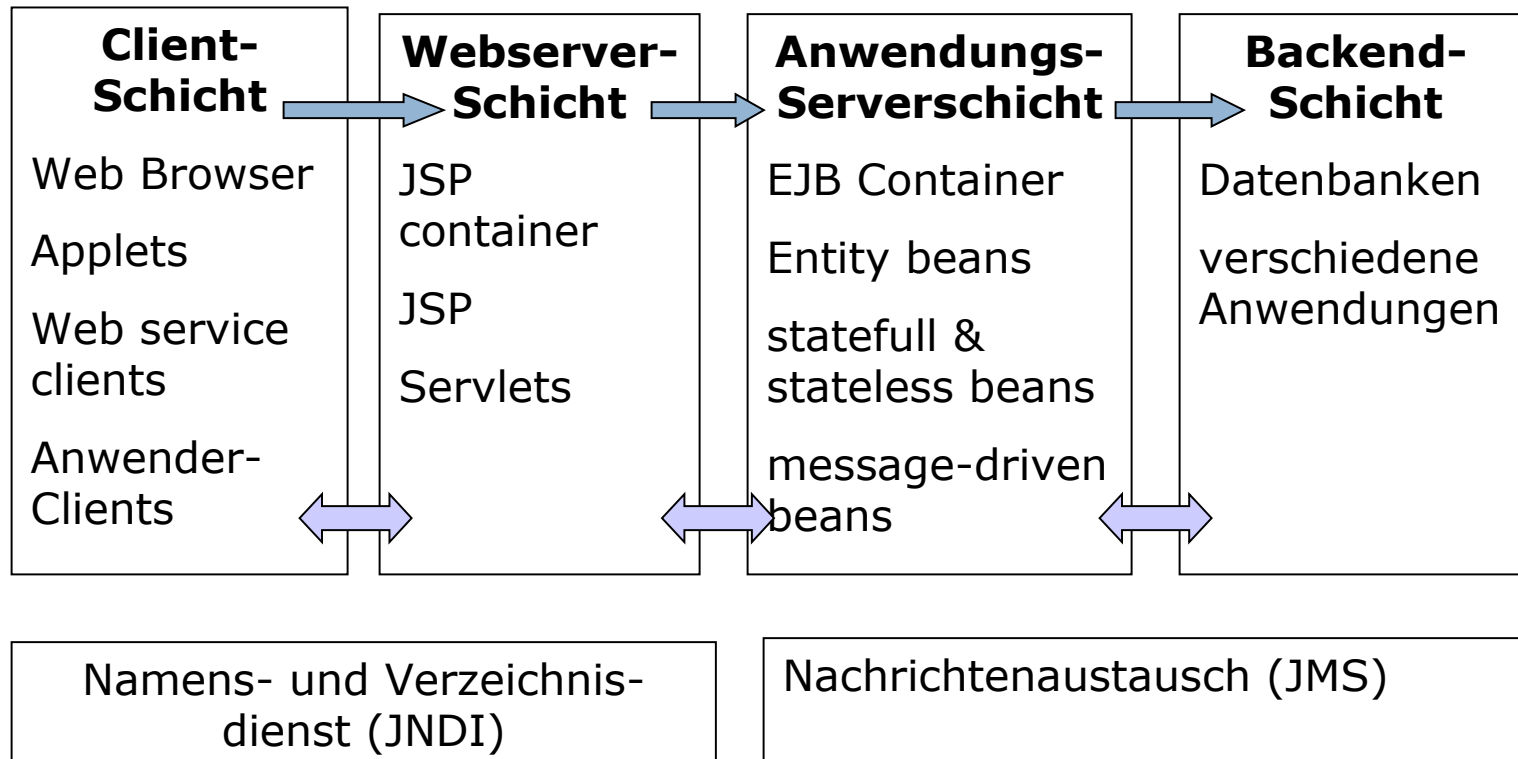
- **Objektserialisierung:** Objekte, welche die Schnittstelle *Serializable* implementieren, können in einen Datenstrom geschrieben oder aus einem solchen aufgebaut werden (z.B. Speichern in eine Datei)
- **Ereignisse (events):** werden einige Folien später genauer besprochen

J2EE Architektur und Javas Komponentenmodelle für Middleware-Anwendungen

- Im Zentrum steht **Familie von Komponentenmodellen**
 - Client-Schicht: Anwenderkomponenten, JavaBeans, Applets
 - Webserver-Schicht: Servlets, JSP
 - Anwendungsserver-Schicht: EJB in vier Varianten (stateless session, statefull session, entity, message-driven session)
- Integrations-Ebenen (Basisdienste):
 - Namens- und Verzeichnis-Infrastruktur (naming and directory interface, JNDI) sowie Nachrichten-Infrastruktur (Java messaging service, JMS) bilden die Klammern zwischen den verschiedenen Schichten
 - weitere I.-E.: Transaktionskoordinierung, Sicherheitsdienste

J2EE Architektur

Kontrollfluss: 
Datenfluss: 



Java-Komponentenmodelle

- Komponentenmodelle in der Java 2 Enterprise Edition
 - **Servlets / Java Server Pages**
 - **Enterprise JavaBeans** und
 - **Application Client Components**
- Alle drei Modelle sind serverseitige Komponentenmodelle.
- Wichtige Gemeinsamkeit ist die **Absetzung der Entpackungsphase** (deployment): Gesteuert durch Deployment-Deskriptor im XML-Format
 - **Entpackung** = Prozess der Vorbereitung einer Komponente auf den Einsatz in einer speziellen **Umgebung** (context)
 - Herstellen/Prüfen der (komponentenspezifischen) Voraussetzungen, unter denen die Komponente arbeitsfähig ist.
 - Beispiel: Datenbank-Anbindung, Persistenzfragen
 - wird unterschieden von der **Installation**, die (plattformspezifisch) eine entpackte Komponente in einer speziellen Hardware-Konfiguration verfügbar macht.

3.4. Java

Java-Komponentenmodelle

Java Komponentenmodelle im Überblick

- **Applets**: herunterladbare leichtgewichtige Komponenten für den Einsatz in Webbrowser-Clients
 - Einsatzgebiet heute durch andere Technologien abgedeckt
- **JavaBeans**: unterstützt verbindungsorientiertes Programmieren
 - zentraler Ansatz: Beobachter und Ereignisse
 - wenig Gemeinsamkeit mit EJB (außer dem Namen)
- **Java Servlets** greifen den Gedanken von Applets auf, laufen jedoch auf dem Server ab und sind (meist) leichtgewichtige Komponenten
 - werden durch einen Web Server instanziiert
 - **Java Server Pages (JSP)**: verwandte Technologie, mit der dynamische Webseiten deklarativ definiert werden können
 - Vergleichbar mit den Microsoft Active Server Pages .NET (ASP.NET)
- **Application Client Components**
 - Im Gegensatz zum Prinzip des Webbrowsers konsistentere Verteilung der Funktionalität zwischen Client und Server (Rich Client)

Distribution und Packung von Java-Komponenten

- Geschäftsanwendungen (enterprise applications)
 - ⇒ *.ear Dateien (enterprise archive)
- Servlets und JSP
 - ⇒ *.war Dateien (web archive)
- Applets, JavaBeans, EJB
 - ⇒ *.jar Dateien (Java archive)
- Entpackungs-Beschreibung (Deployment descriptor)
 - Formulierung von Anforderungen der Komponente durch Komponenten-Entwickler
 - Setzen offener Parameter (der „Blanks“) der Komponente durch Komponenten-Assembler
 - hart verdrahtet während der Entpackungsphase
 - Komponenten sind sonst zustandslos
 - Assembler ist eine andere Rolle als Komponentenentwickler, oft sogar in einer anderen Organisation
 - andere Ansätze trennen diese beiden Rollen deutlicher