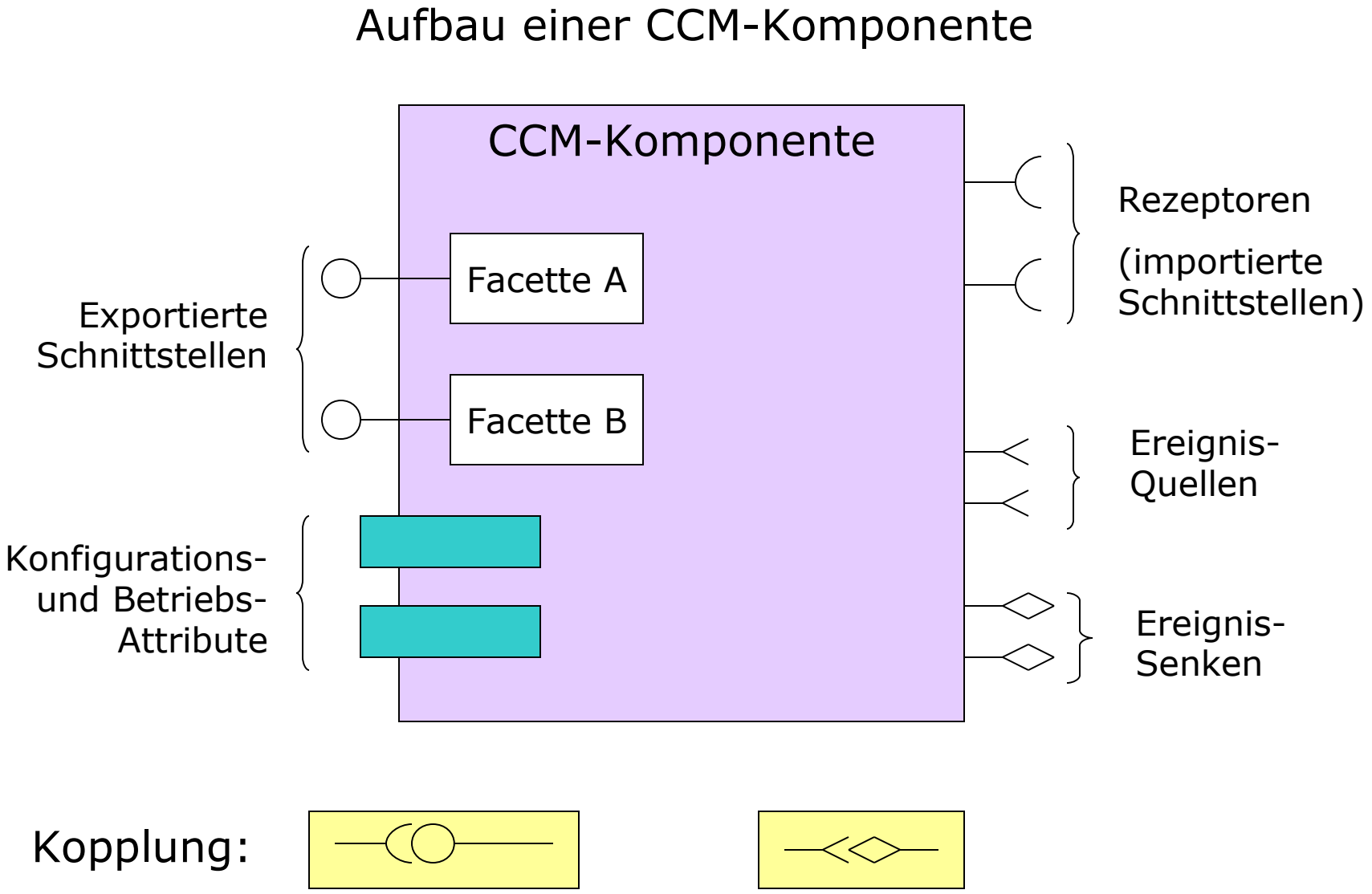
Vorlesung
Software aus Komponenten

4. Moderne Komponentenkonzepte

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2009/10

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ports von CCM-Komponenten

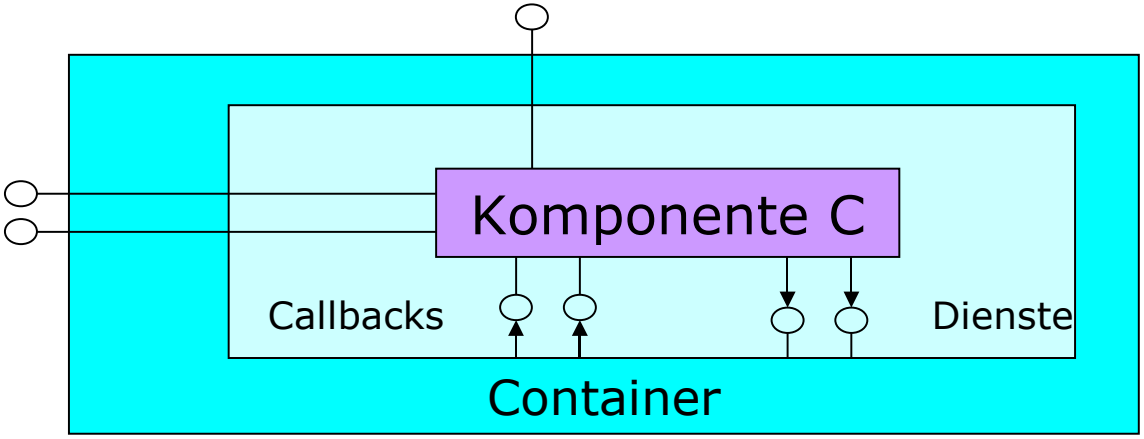
- **Facetten** (facets)
  - exportierte Schnittstelle, gewöhnlich einem Teilobjekt der Komponente zugeordnet
- **Rezeptoren** (receptables)
  - importierte Schnittstellen, intern Referenzen auf externe Objekte, die zum Komponentenbetrieb benötigt werden
  - connect / disconnect Operationen
  - können explizit in der Montage-Beschreibung gefordert oder zur Laufzeit eingebunden werden
- **Ereignisquellen** (event sources) und **Ereignissenken** (event sinks)
  - durch Ereigniskanäle zu verbindende Ports
- **Primärschlüssel** (nur Entity-Komponenten)
- **Konfigurations-** und **Betriebs-Attribute**
  - benannte Werte, die über **Zugriffsfunktionen** (accessor) oder **Modifizierer** (mutator) nach außen sichtbar sind

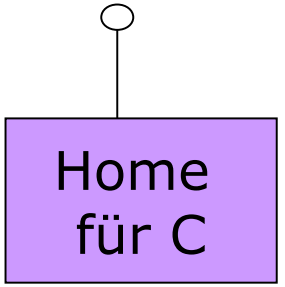
- **Home-Schnittstelle**, über welche die Komponenten-Factory erreicht werden kann
  - in der Komponenten-Klasse implementiert
  - also Komponentenbegriff verschieden von dem in der Vorlesung
  - Management des Lebenszyklus von Komponenten-Instanzen
- Spezielle Facette **E-Schnittstelle** (equivalent interface), über die zwischen den Facetten der Komponente navigiert werden kann
  - Clienten müssen CORBA-3 unterstützen, um diese Navigationsmöglichkeiten auszunutzen
- **Konfigurations-Schnittstelle** (configuration interface)
  - Unterstützung der initialen Konfiguration neuer Komponenten
  - spezielles call-Signal schließt die Konfigurationsphase ab
  - erst danach sind Aufrufe der operationalen Schnittstellen möglich, Aufrufe der Konfigurations-Schnittstelle dagegen untersagt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CCM-Container

- CORBA 3 definiert ein **Komponenten-Implementierungs-Gerüst** (component implementation framework, CIF)
  - Generatoren erzeugen aus Eingaben im **CIDL-Format** (component implementation description language) Code, der den Komponentencode ergänzt
- Jede Komponenten-Instanz ist in einem **CCM-Container** untergebracht, über den die Anbindung der Facetten und Rezeptoren erfolgt. Rezeptoren und Dienste können in einem solchen Container per Callback gebunden sein.





- Der CCM-Container ist ein spezieller POA

Vorgefertigte Basisdienste (pre-packaged object services)

- **Transaktionsdienst**: durch Container oder selbst
  - Komponente: Beschreibung enthält die Transaktionsanforderungen (supported, required, required new, not supported)
  - Container: Ausführung der Transaktionen entsprechend der Spezifikation der einzelnen Komponenten
- **Persistenzdienst**: durch Container oder selbst
  - Komponente: Beschreibung der Anforderungen im PSDL-Format (persistent state description language)
- **Sicherheitsdienst**:
  - Zugriffsrechte können im CIDL-Format beschrieben und durch den Container geprüft werden
  - Benachrichtigungsdienst: Aufbau und Verwaltung von Ereigniskanälen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OSGi - Plattform

Hardwareunabhängige dynamische SW-Plattform, auf der Anwendungen und Dienste kombiniert und ausgeführt werden können

- spezifiziert Java-basierte Laufzeitumgebung oberhalb der JVM und deren Basisdiensten
- speicherplatzfreundliche Einbettung verschiedener Dienste in dieselbe JVM

Entstand aus Komponentenansätzen im Bereich eingebetteter Anwendungen

- Ziel: herstellerunabhängige generische SW-Plattform zur Steuerung und Vernetzung von Geräten aller Art
- Anwendungen in den Bereichen Automotive, Handy, Gebäude-Management, Fernsteuerung von Hausgeräten

Grundprinzip des **Gateways**: Plattform wird nicht auf einzelnen Geräten ausgerollt, sondern die Infrastrukturdienste werden im Zusammenspiel der Geräte, als wirklich verteiltes System, erbracht.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Service-Anwendungen als **Bündel**

- können dynamisch zur Laufzeit eingespielt, aktualisiert und auch wieder entfernt werden, ohne die JVM (bzw. das Grundgerät) anzuhalten.
- können über Fernwartung administriert werden
- Abhängigkeiten werden automatisch aufgelöst
- intelligentes Versionsmanagement

Ermöglicht nachträgliche Auslieferung und Installation von Diensten sowie Verteilung von Informationen zur Laufzeit auf verschiedenen Endgeräten

Heute auch als **Applikationscontainer im Enterprise-Bereich** verfügbar

- Eclipse – Equinox-Projekt als prominenteste Entwicklung
  - Seit Eclipse 3 Basis für dessen Plugin-Konzept
- ermöglicht fein granulare Komposition von Anwendungen
- schlank und einfach gegenüber dem etablierten J2EE-Standard

Konzept ermöglicht **Einbeziehung der Clients in Dienstekonzept**

- Fernwartung von Clients
- Rich Client Konzepte

Aufnahme und Standardisierung im Rahmen des Java Community Prozesses als JSR 921 – *Dynamic Component Support for Java SE* – als offizielles dynamisches Komponentenmodell für Java

OSGi Allianz (früher: Open Service Gateway Initiative) – Industriekonsortium zur Koordinierung der Weiterentwicklung dieser Spezifikation

- Derzeit über 100 Firmen aus sehr unterschiedlichen Branchen, als Vollmitglieder u.a. Sun, IBM, Nokia, Motorola, Oracle, NEC, Hitachi, Red Hat, Samsung, Siemens, Telefonica, BEA, Dt. Telekom
- Unterscheidet Vollmitglieder, Anwender und Unterstützer
- Branchenspezifische Belange werden in Expert Groups gebündelt

- Dokumente verfügbar unter der OSGi Specification License
  - enthält eine Nicht-Patent-Klausel
- Community Prozess offen nur für Mitglieder

Leistungsimplementierungen, meist in Teilbereichen mit speziellem Anwendungsfokus, von dritter Seite

- bestehen meist aus dem Framework und einer größeren Anzahl von vorgefertigten Service-Bündeln (Basisdienste), die selbst wieder dynamisch installiert werden können
- kommerzielle Framework-Implementierungen
- Open Source Framework-Implementierungen
  - Equinox (von Eclipse getrieben)
  - Oscar – Apache Felix (Community-Projekt der Apache Foundation)
  - Concierge – für mobile und eingebettete Systeme

Im Enterprise-Bereich besonders im Zusammenspiel mit Spring eingesetzt

Durch Verbreitung von OSGi-basierter Middleware in verschiedenen Anwendungsbereichen entsteht ein Markt für OSGi-Komponenten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Geschichte

- März 1999: Gründung der Allianz
- Mai 2000: R 1 mit Fokus auf Vernetzung von Haushaltsanwendungen
- Oktober 2001: R 2 mit Verbesserungen im Bereich Sicherheit und Fernwartung
- März 2003: Einsatz in den Bereichen Automotion und Unterhaltung
- Juni 2004: R 3 und Einsatz in Eclipse
- Okt. 2005 bis Sept. 2006 : R 4 mit den Teilen Kern-Spezifikation, Kern-Erweiterungen, Mobil-Spezifikation
- Aktuelle Version 4.2 (September 2009)

Die Allianz spezifiziert nur APIs und Testfälle, Referenzimplementierung nur als „proof of concept", nicht für den Produktiveinsatz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Leistungsmerkmale

Standardisiertes nicht-proprietäres SW-Komponenten Framework für Hersteller, Diensteanbieter und Entwickler

- Ausprägung als offener Standard führt zu fairer Balance zwischen den einzelnen Akteursgruppen
- Zwei zueinander orthogonale Modelle: Dienste-Modell (Services) und Komponenten-Modell (Bündel)

SW-Modell für **Koexistenz mehrerer Java-basierter Komponenten in einer einzigen JVM** mit besonderem Augenmerk auf Sicherheitsaspekte

- minimiert die Speicheranforderungen, verbessert die Performanz und die Geschwindigkeit der Kommunikation
- detailliertes Sicherheitskonzept erlaubt Abschirmung der Komponenten gegeneinander und gegen den Kontext (Sandkastenkonzept)
- Adjustierung der Rechte erlaubt aber auch breite Zusammenarbeit verschiedener Komponenten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verwaltungsschnittstelle für Software-Komponenten

- Deployment als Funktion der Komponente selbst
- Standardisiertes Deployment-Format als Bündel
  - Zusammenfassung von Java-Paketen und Ressourcen
- Deployment erfolgt nicht passiv in die Service-Plattform, sondern aktiv durch die Plattform selbst
  - Komponente wird „assimiliert"
- Ermöglicht so „heißes" Laden, Starten, Stoppen, Aktualisieren, Herausnehmen
- Verschiedene Versionen derselben Komponente können in einer VM koexistieren
- Deklaration von Export- und Importrelationen werden auf der Ebene von Java-Paketen über Namensäquivalenz von der Service-Plattform aufgelöst
- Laufzeitkonfiguration wird beim Runter-/Hochfahren persistent gespeichert

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Sichere Ausführungsumgebung

- Mehrebenen-Sicherheitsmodell, das Java-Konzepte mit OSGi-eigenen koppelt
- Java Run Time: Pointer-Sicherheit, Puffer-Überläufe
- Java Sprache: Modifier legen Sichtbarkeiten fest
  - damit können Bündel gegeneinander abgeschirmt werden
  - weiterer Modifier für Zugriff innerhalb eines Bündels
- administrative Rechte auf Bündelebene (minimal bundle content exposure)
  - Zugriff auf Ressourcen kann begrenzt werden
  - Alle Objekte auf dem Aufrufstack müssen über das Recht verfügen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Sichere Ausführungsumgebung (cont.)

- Rechte, einzelne Java-Pakete aus einem Bündel zu importieren oder zu exportieren (managed communication links)
  - Interaktion zwischen Bündeln ist nur bei entsprechenden Rechten möglich
- Paket-Sharing als mögliche Angriffsquelle
  - Möglichkeit der Definition von vertrauenswürdigen Bündeln (package permission)
- Rechte zu Publikation, Suche oder Nutzung dezidierter Dienste (service permission)

- Diese Rechteverwaltung wird vom Plattform-Betreiber zur Verfügung gestellt und kann so an verschiedene Sicherheitsszenarien angepasst werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kooperatives Modell des gegenseitigen Findens und Nutzens der Dienste von Komponenten innerhalb derselben OSGi-Plattform

- Dienst = Java-Objekt, welches ein Bündel einem anderen zur Verfügung stellt
- damit sind sehr schlanke Komponenten möglich
- wichtig besonders im Bereich eingebetteter und mobiler Anwendungen mit seinen Ressourcenbeschränkungen
- Vermeidung von Redundanz auf der Ebene von Basisdiensten, die sonst mehrfach in Komponenten vorhanden sind, wenn sie nicht von der Plattform zur Verfügung gestellt werden (class sharing)
- besonderer Schwerpunkt liegt auf aussagefähigen Testfall-Sammlungen, mit denen die dynamische Zusammenarbeit verschiedener Bündel getestet werden kann.

Flexible Fernwartungs-Architektur: Erlaubt Verwaltung mehrerer tausend Service Plattformen von einem einzigen Management-Bereich aus

- genaues Lebenszyklusmodell erlaubt sehr feingranulare Verwaltung der Komponenten über enge oder weite Policies
- Definition einer **Fernwartungs-API**: Wartungsaufgaben können auf einer höheren Abstraktionsebene formuliert werden
- Bindung an verschiedene existierende Protokolle
  - erlaubt die Ausführung der Wartungsaufgaben mit einem für die Geräte optimalen Wartungsregime unter Berücksichtigung von Gerätespezifika
  - variable Verfügbarkeit und geringe Bandbreite im Mobilbereich
  - permanente Verfügbarkeit und hohe Bandbreite bei Hausdienstleistungen
- Conditional Permission Admin Service erlaubt zusammen mit der Unterstützung digitaler Signaturen dynamische Wartungsregeln, mit der größere Farmen von Softwareinstallationen einfach gewartet werden können

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

COTS – Kommerzielle Komponenten „aus dem Regal“

- OSGi als Basis für einen sich rasch entwickelnden Komponentenmarkt
  - So gut wie alle allgemein gebräuchlichen Protokolle, die es gibt, liegen heute bereits als OSGi-Bündel vor.
- Fein granulares Konzept erlaubt die Entwicklung von Bündeln mit hoher funktionaler Bindung – ein Bündel, eine Funktion
- Trennung von Spezifikation und Implementierung erlaubt, verschiedene Implementierungen mit verschiedenen Leistungsparametern und für verschiedene Geräteplattformen von verschiedenen Anbietern durch einen Betreiber plattform- und anwendungsspezifisch zusammenzustellen.
  - OSGi Service Plattform – wenn einmal aufgespielt – standardisiert die Verwaltung dieser Komponenten
- Zertifizierung von Plattformen und Bündeln ist (nur) für Mitglieder der Allianz möglich
  - Zusicherung der plattformunabhängigen Lauffähigkeit von Bündeln

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Standardisierte optionale Dienste und Werkzeuge als generische Lösungen durch Verwendung vorgefertigter Komponenten als **Basisdienste**

- Logging, Konfiguration, HTTP, XML, Netzwerk, IO, Ereignisbehandlung und -verwaltung, Geräte-Erkennung und Laden von Treibern (PnP), Nutzer-Authentifizierung und Autorisierung, usw.

Anwendung im Middleware-Bereich

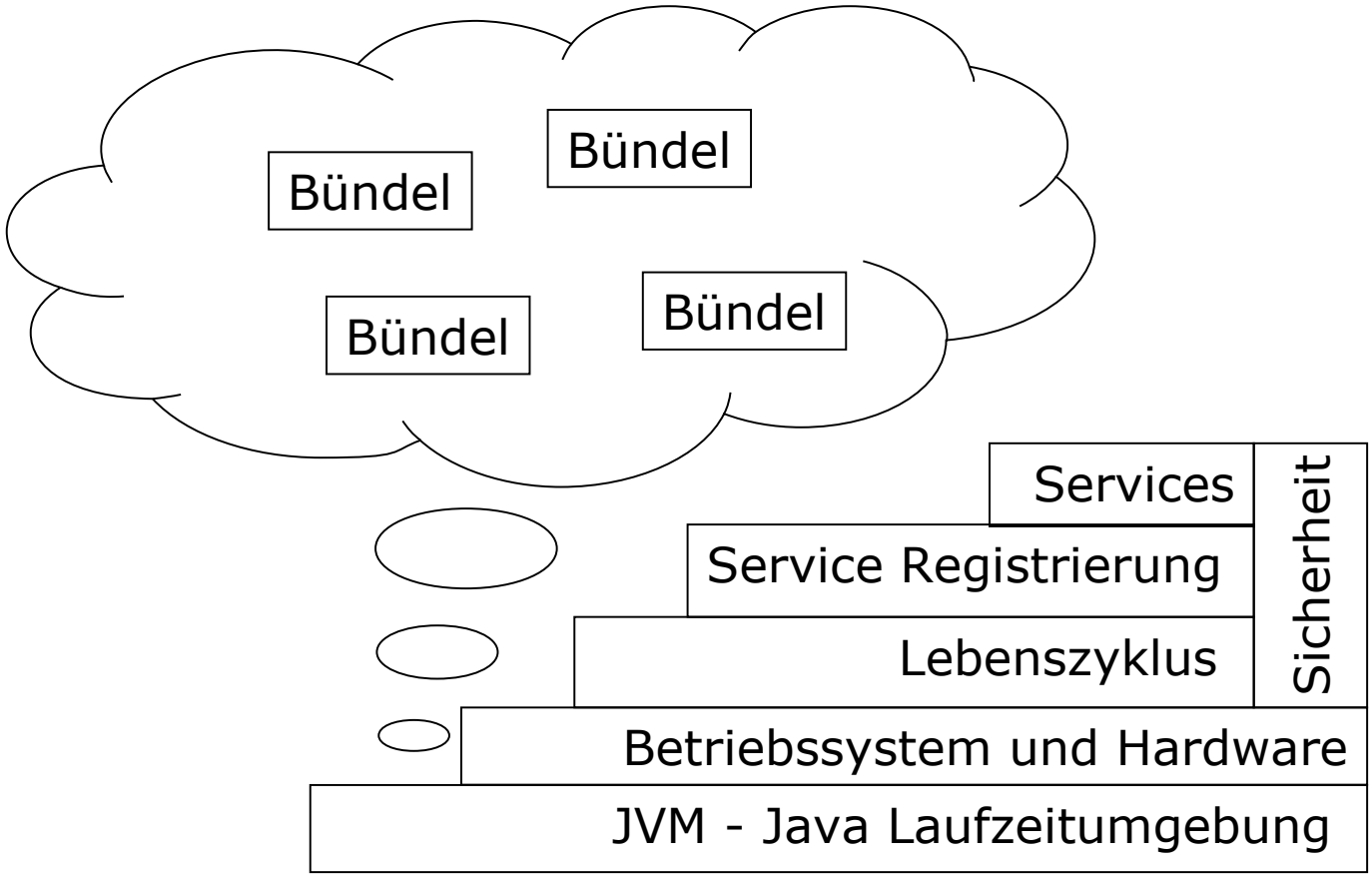
Interkonnektivität innerhalb einer Schicht zusammen mit Anbindung des Kontexts an darunter liegende SW-Schichten bestimmen die Leistungsfähigkeit einer Komponenten-Architektur im Middleware-Bereich

- Besondere Bedeutung des Zusammenspiels von OSGi (Interkonnektivität) und Spring (Kontextanbindung) als zueinander orthogonale Architekturkonzepte

Ende 2006 nahm innerhalb der OSGi-Allianz eine **Enterprise Experts Group** die Arbeit auf

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OSGi Service Plattform Architektur - Überblick



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Aufgaben der OSGi-Plattform

- Installation: **installBundle**-Methode der Plattform
  - Das Bündel wird für den Einsatz in der Umgebung vorbereitet
  - Service Plattform registriert die neuen Import- und Exportrelationen (auf der Ebene von Java-Paketen)
  - Erzeugt ein **Bundle**-Objekt und initialisiert den **BundleContext** innerhalb des Bündels
- Vor dem Start müssen Abhängigkeiten aufgelöst werden
  - Bündel heißt **aufgelöst**, wenn alle Importrelationen durch Exporte anderer aufgelöster Bündel bedient werden
  - Kann Auflösung mehrerer anderer Bündel anstoßen
  - Zirkuläre Abhängigkeiten sind möglich.
- Deinstallation erfolgt virtuell – Pakete des Bündels sind so lange verfügbar, so lange aufgelöste Bündel mit diesen Abhängigkeiten existieren.
  - Neuauflösung von Abhängigkeiten über refresh des Managers sind möglich .

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Start/Stop: Aufgelöste Bündel können gestartet und angehalten werden. Dabei werden die erforderlichen Ressourcen faktisch zugeordnet bzw. entzogen.
  - Dazu erzeugt das Bündel ein **Aktivator-Objekt**, welches den Lebenszyklus über die Methoden **start** und **stop** kontrolliert.
  - Beim Start wird der Kontext an das Bündel gebunden.
  - Aktive Bündel können über den Kontext **Dienste** veröffentlichen, suchen und binden. Dienste sind einfache Java-Objekte.
  - Dienste können über ihre Schnittstelle sowie eine Metadaten-Tabelle **registriert** werden – **Service Registry**
  - Zustand wird persistent gespeichert und bei Stop/Start der Plattform aktive Komponenten automatisch hochgefahren
    - Möglichkeit der Steuerung über **start levels**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Realisierung über den **Management-Agent** – ein normales Bündel mit Administrationsprivilegien
  - Dienst =Verwaltung der Konfiguration der Laufzeitumgebung, in der er selber läuft
  - Komplexität dieses Agenten bestimmt die Leistung der aktuellen Plattform
  - Verantwortlich für die Konsistenz der Laufzeitkonfiguration
- Aufgaben:
  - Verwaltung der Bündelauflösungen
  - persistente Verwaltung der aktiven Konfiguration
  - Verwaltung der Rechte in Bezug auf einzelne Bündel
    - Erfolgt über **Lokalisierungsinformation** (=abstrakte Identität), welche vom Manager für das Bündel bei dessen Installation vergeben wird
  - Sicherung der **Namensäquivalenz**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Namensäquivalenz und Bündelabhängigkeiten

Bündelabhängigkeiten auf der Ebene von Java-Paketen

- unterscheide Spezifikations- und Implementationspakete
- **Spezifikationspaket**: Name und Version definieren die Schnittstelle, die für alle Provider verbindlich ist.
  - Provider müssen die voll Schnittstelle implementieren
  - spätere Versionen dürfen frühere nur erweitern
    - Problem der semantischen Konsistenz bleibt trotzdem!
- **Implementationspaket**: Nichts davon gilt.
  - Inhalts- und Versionsnummern sind inkompatibel zwischen verschiedenen Providern.
- Beispiel: Paket importiert eine Spezifikationsschnittstelle und exportiert eine Implementierung dieser Schnittstelle.
- Es ist Aufgabe des Management-Bündels, diese Referenzen korrekt aufzulösen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Service-Registrierung

- Einheitliche Schnittstelle, um Änderungen der Konfiguration an alle registrierten Service-Objekte zu propagieren
  - diese müssen insbesondere Event-Kanäle zu der geänderten Komponente aktualisieren.
  - in-memory-Register, um Zugriffszeiten zu minimieren.
  - „Kleber, der die Plattform zusammenhält"
- Registrierte Objekte heißen **Dienste**. Zugeordnet ist immer ein Schnittstellenname und eine Liste von Eigenschaften.
  - Deklarative Dienste – Bündelklassen werden erst geladen und Objekt erzeugt, wenn der Dienst in Anspruch genommen wird.
- Funktionalität:
  - Objekte einzelner Bündel registrieren
  - Service-Register nach passenden Objekten durchsuchen
  - Nachrichten über Registrierung und Deregistrierung von Objekten verschicken

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Laufzeitumgebung

- Laufzeitumgebung basiert auf Einzel-JVM-Umgebung, wobei sogar J2ME Profile zum Einsatz kommen können
- Plattform verwaltet physische Bündelinhalte
- Auffinden über die Lokalisierungsinformation des Bündels, welche das Management-Bündel plattformspezifisch vergeben hat (Namens-Dienst)
- Bündelinhalt wird von dieser Quelle bezogen
  - URL: Bündelinhalt wird über einen Inputstream bezogen
  - lokale Klasse: Bündelinhalt wird über Classloader bezogen.
  - Plattform nutzt für jedes Bündel einen eigenen Classloader
- Strenge Trennung zwischen Schnittstelle und Implementierung, um Abhängigkeiten sauber auf Spezifikationsebene verwalten zu können.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Basisdienste

- Basisdienste wurden im Rahmen von OSGi 4 (Core Specification and Service Compendium) standardisiert
  - eine Spezifikation – mehrere Implementierungen
- Verfügbarkeit optional, abhängig vom Anwendungsgebiet
- Framework Services – Dienste zur Verwaltung der Plattform selbst
  - Permission Admin, Conditional Permission Admin, Package Admin, Start Level, URL Handler
- System Services – horizontale Dienste, die von vielen Bündels benötigt werden
  - Log Service, Configuration Admin, Event Admin, Device Access, User Admin, Deployment Admin, Monitoring usw.
- Protocol Services – Abbildung eines externen Protokolls auf einen Service
  - Http Service: dynamischer Servlet-Starter
  - UPnP Service: neuer Standard in der Unterhaltungselektronik, wird auf die Service-Registrierung abgebildet

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Miscellaneous Services
  - Wire Admin: Zusammenbinden dezidierter Dienste entsprechend einem vorgegebenen Abo-Schema statt allgemeiner Kompatibilität
  - XML Parser
- Programming Services – Unterstützung zur Behandlung der Komplexität des Programmiermodells
  - Service Tracker – verfolgt Änderungen an den Diensten
  - Declarative Services – Definition einer Service Component Runtime, die Servicedefinitionen aus Bündeln liest und diese an der Stelle des Bündels selbst registriert.
- Weitere Basisdienste in Entwicklung
  - Remote Management – Abstraktionsschicht oberhalb der verschiedenen Fernwartungsprotokolle
  - Web Services – Implementierung der Basisfunktionalitäten
  - Und weitere

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile der Plattform

- Deutliche Senkung der Entwicklungskosten und Einarbeitungszeiten möglich.
- Flexiblere Möglichkeit, um Geräte für verschiedene Märkte anzupassen
- Deutliche Verringerung der Probleme im Deployment Prozess
- Fernwartungsmöglichkeiten
- Einheitliche Behandlung eingebetteter Geräte und deren Einbettung in die Umgebung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenkonzepte und Anforderungen im Vergleich

Eine Zusammenfassung

- Komponententechnologie und Softwaretechnik
- Komponentenkonzepte im Vergleich
- Konvergenz der Konzepte
- Differenzen der Konzepte
- Komponenten und Objekte
- Kontraktspezifikationen für Komponenten
- Komponenten und Softwaretechnik
- Komponenten-Montage
- Komponenten und Berufsprofile

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Softwaretechnik als Ingenieurtechnik

Ingenieurtechnik

- Standards, Vorgehensweisen und Zusammenhänge, die beim Bearbeiten einer Aufgabenstellung aus dem jeweiligen Gebiet von einer qualifizierten Fachkraft zu berücksichtigen sind.
- technologische Einbettung der für das jeweilige Gebiet verfügbaren Technik

Softwaretechnik ist eine ingenieurtechnische Disziplin

- Lehre von Planung, Erstellung, Einsatz, Wartung und Weiterentwicklung von komplexen Software-Systemen in einem arbeitsteiligen Prozess
- und den dabei zweckmäßig zum Einsatz kommenden Prinzipien, Methoden und Werkzeugen. [Balzert]

- Im Zentrum steht dabei die **Beherrschung der Komplexität** der Anforderungen aus dem Lebenszyklus von Software-Systemen.
- Als typische Arbeitsschritte haben sich bewährt
  - Anforderungsanalyse, Entwurf, Modellierung, Realisierung, Montage, Einsatz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponententechnologie aus ingenieurtechnischer Sicht

- Die **Nutzung von Komponenten** ist ein Charakteristikum jeder entwickelten Ingenieurtechnik
  - Neue Produkte werden aus vorgefertigten, standardisierten, dem Stand der Technik entsprechenden Bestandteilen nach allgemein anerkannten Standards und eigener Kreativität zusammengebaut.
  - Form der Komplexitätsreduktion
- **Komponententechnologie** hat zum Gegenstand das Zusammenspiel von Komponentenentwicklung und Komponenteneinsatz
  - Rolle: **Komponentenentwickler**, Perspektive: Zulieferer-Sicht
    - Komponenten für möglichst breites Einsatzfeld entwickeln
  - Rolle: **Komponentenmonteur**, Perspektive: Dienstleister-Sicht
    - Komposition von Anwendersystem aus geeigneten Komponenten
- Ansatz findet über mehrere hierarchische Ebenen der Komposition statt
  - Treiber – Betriebssysteme
  - Laufzeitbibliotheken – Hochsprachen-Programme
  - der in dieser VL besprochene Komponentenbegriff

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponententechnologie und Softwaretechnik

- **Ziel**: Montage eines IT-Systems, das als **verteilte Anwendung** auf einem System von mehreren miteinander verbundenen Rechnern aus **Komponenten unterschiedlicher Hersteller** konzipiert ist.

- ***Anforderungen:***
  - formal fundiertes **Komponentenkonzept** als Basis
  - **Beschreibungstechniken** für derartige Komponenten
  - Entwicklung eines **Prozessmodells** zur Entwicklung, Verwaltung und Zusammensetzung von Komponenten
    - Unterstützung der Zuweisung verschiedener Rollen
  - **Werkzeuge**, welche die Beschreibung und das Prozessmodell unterstützen
    - zur Systemgenerierung selbst
    - zur Dokumentation
    - zur Verifikation und Sicherung wichtiger und kritischer Systemeigenschaften

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zwei grundlegende Herangehensweisen

- eng gekoppelte Architektur (J2EE, CORBA, CLR, OSGi)
  - Laufzeitsystem als Infrastruktur, in der Objektinstanzen ausgetauscht werden, in denen Zustand und Funktionalität des Gesamtsystems lokal gespeichert sind.
  - fein granulares Konzept, Technik der Interaktion steht im Fokus
  - Erweiterung objektorientierter Ansätze von einer Einzelplatzanwendung auf eine verteilte Umgebung
  - grundlegendes Konzept: RPC und dessen Verallgemeinerungen
- lose gekoppelte Architektur (Webservices)
  - hohe Autonomie der Rechner, die nachrichtengesteuert gegenseitige „Dienste“ erbringen
  - grobgranulares Konzept auf höherer Abstraktionsstufe
  - näher am Geschäftsprozess-Modell
  - in dieser Vorlesung nicht besprochen

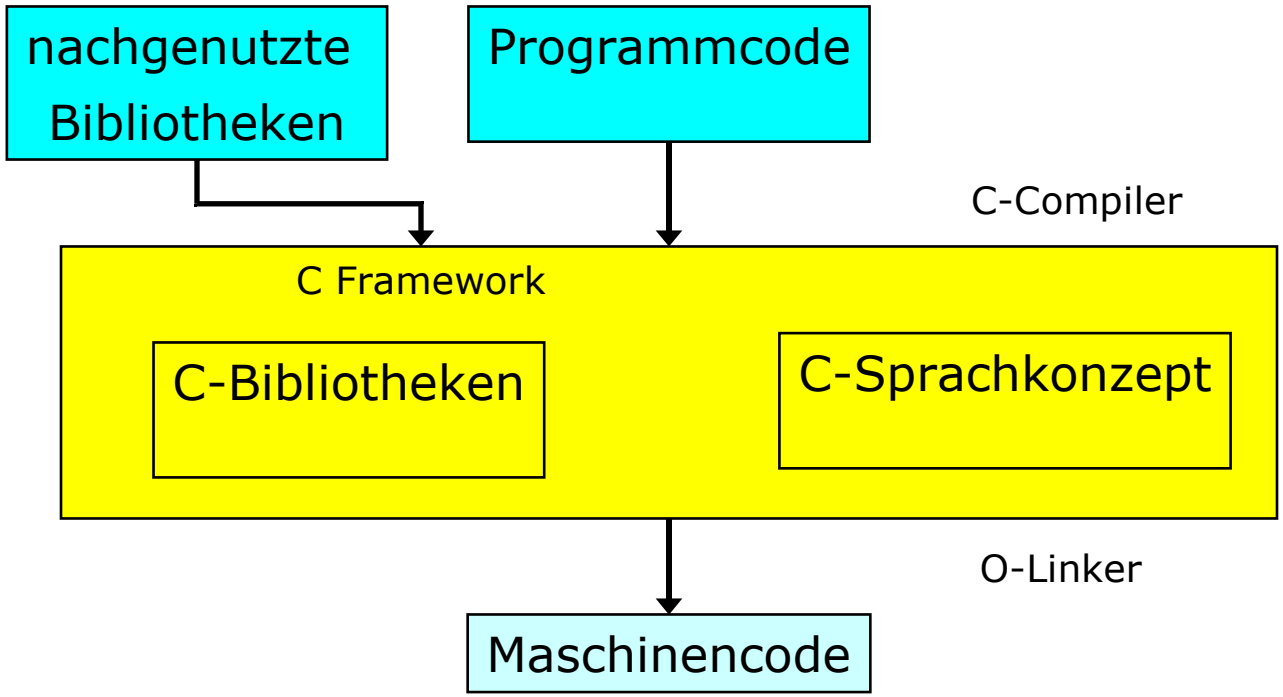
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenmodelle auf Quellcode-Ebene:
Aufbau von Anwendungen aus Software-Bausteinen

Ziel: Sicherung plattform- und sprachübergreifender Kompilierungskompatibilität
Anwendungsbereich: Desktop, Basiskomponenten
Grundlage: Gemeinsame Designprinzipien
Beispiele: C, Java, .NET

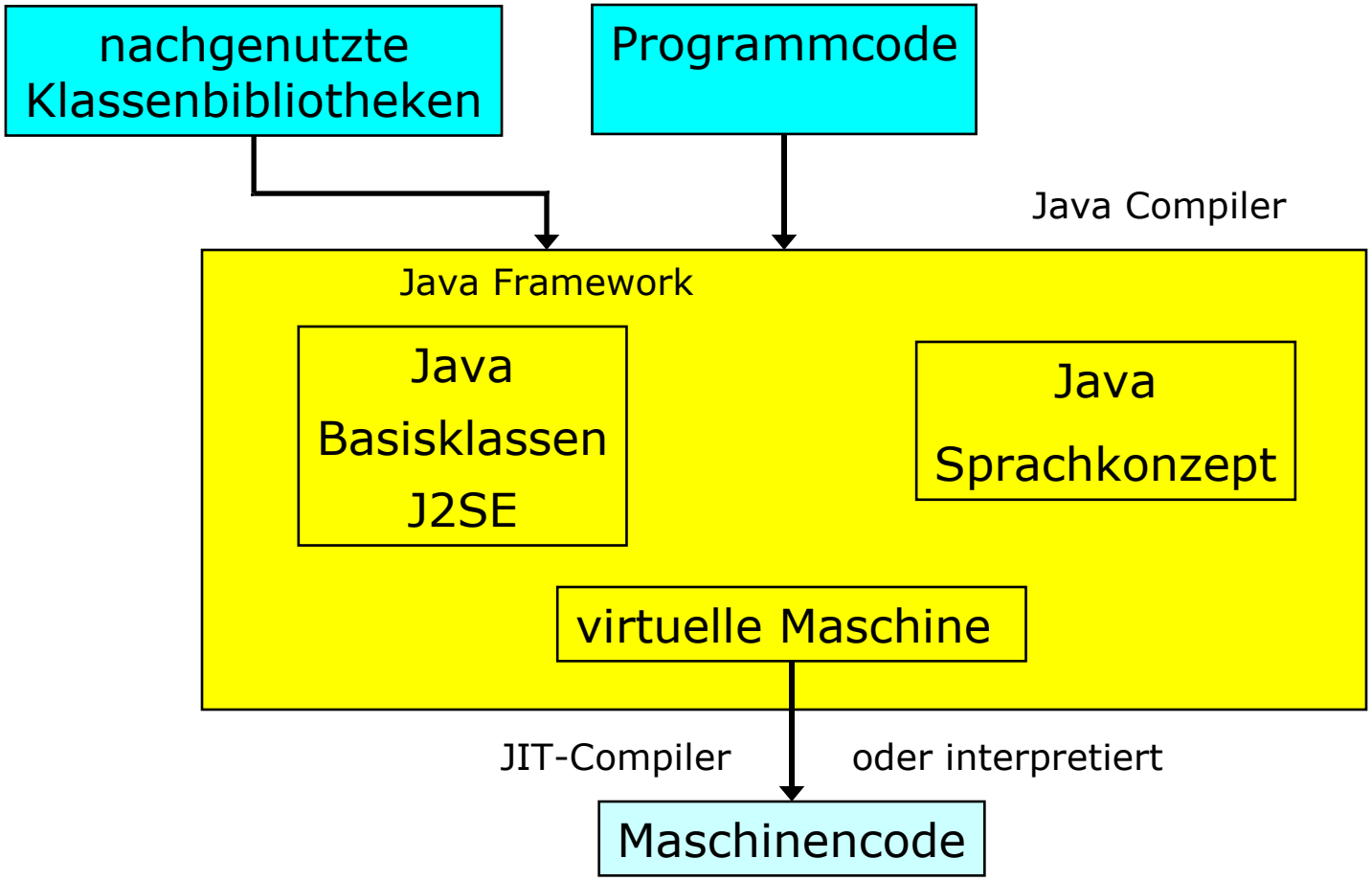
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, eine Plattform: C



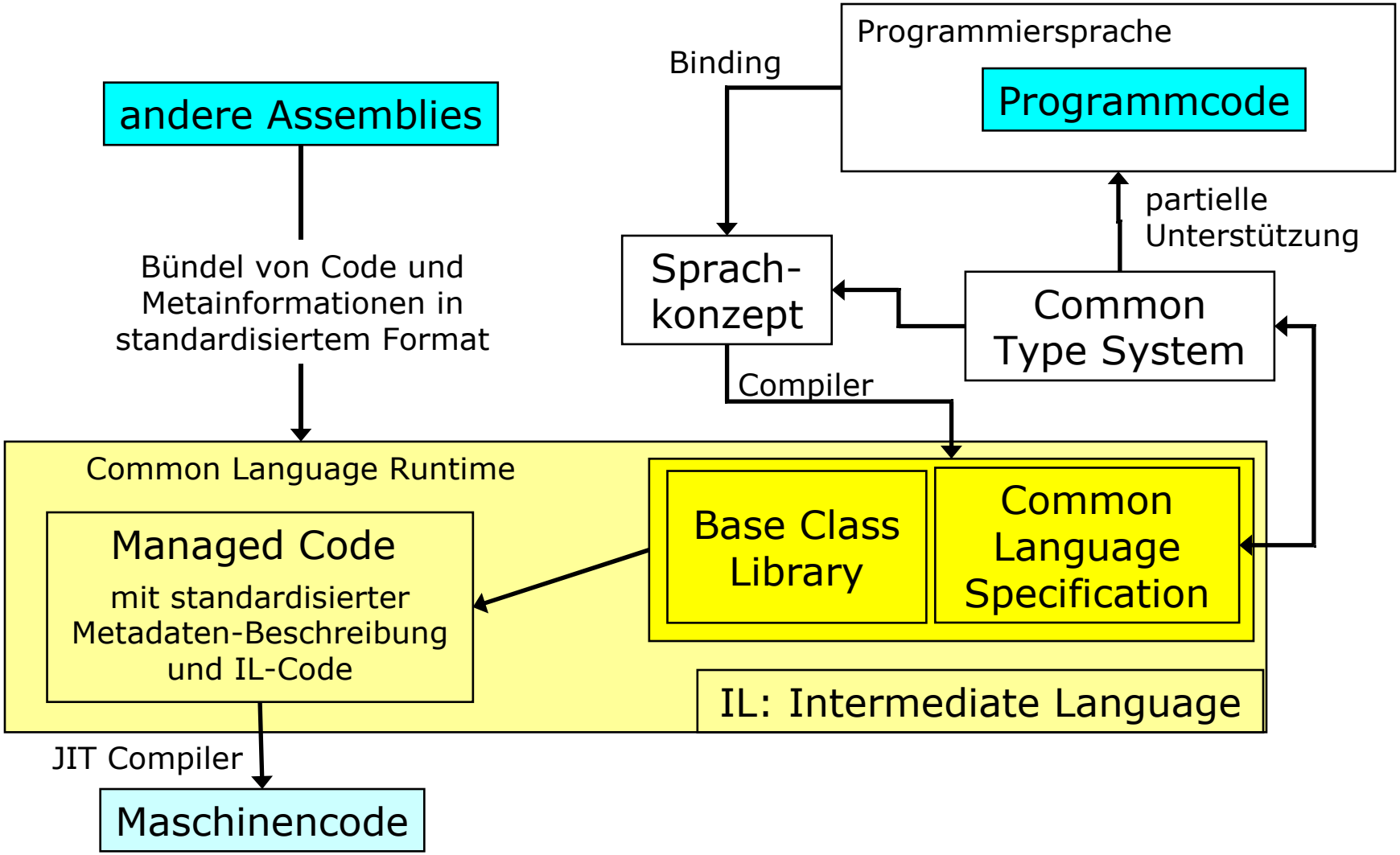
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Eine Sprache, mehrere Plattformen: Java



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Mehrere Sprachen, mehrere Plattformen: .NET



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenmodelle für verteilte Anwendungen:

Aufbau von Anwendungen aus Komponenten

Ziel: Integration von Diensten in eine standardisierte verteilte Infrastruktur

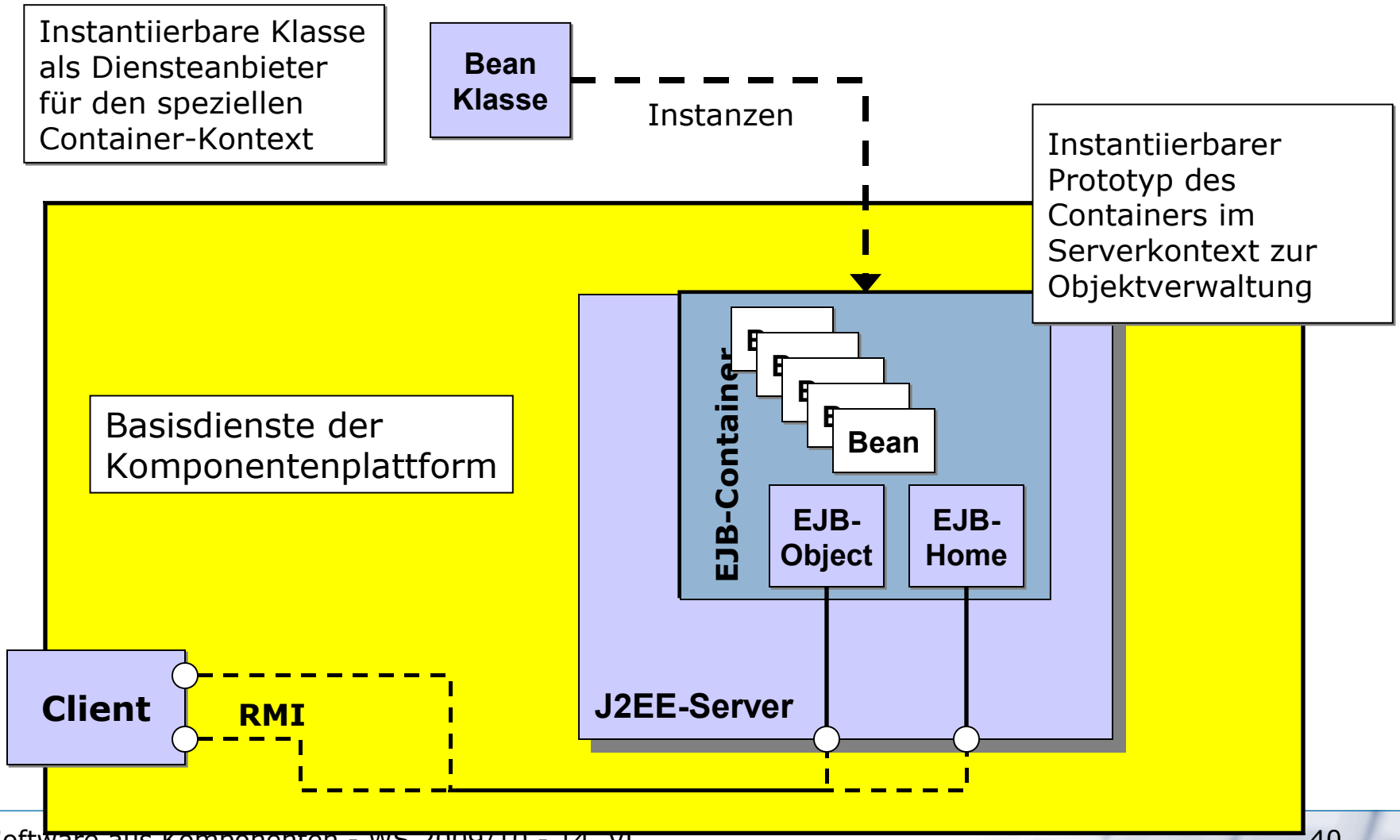
Anwendungsbereich: Middleware und verteilte Systeme

Grundlage: eng gekoppelte Client-Server-Architektur, gemeinsames Framework

Beispiele: EJB, Servlets, CORBA,

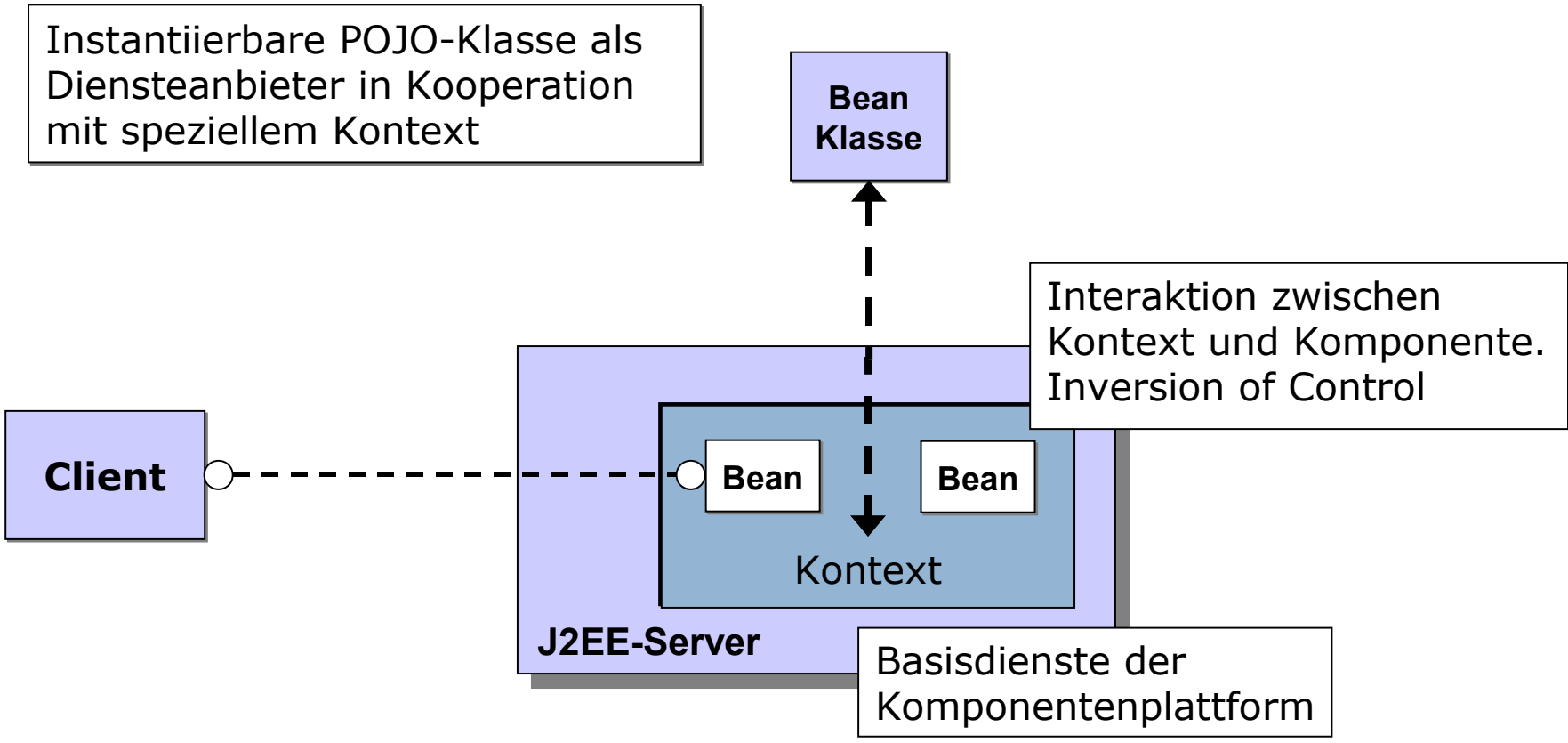
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Prototypischer Aufbau von CORBA und EJB 2



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Prototypischer Aufbau leichtgewichtiger Komponenten-Frameworks



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Konvergenz auf der Ebene der Konzepte

- Alle Zugänge unterstützen spätes Binden, Kapselung, dynamische Polymorphie, Vererbung auf Schnittstellenebene
- Standardisierte Komponenten-Transfer-Formats
  - Java: *.jar, COM: *.cab, CLI: Assemblies
- Uniformer Datentransfer
  - einheitliche Konzepte der Serialisierung von Objekten
  - Entwicklung von Persistenzmechanismen auf dieser Basis
- Ereignis- und Ereigniskanal-Konzept
- Metainformationen über Introspektion und Reflektion
  - erlaubt dynamische Erweiterung von Schnittstellen
- Einsatz von Konfigurationsinformationen
  - Montage-Beschreibungen
  - attributbasiertes Programmieren (CLI) – custom attributes

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Quellcodestandards für Kompatibilität und Portabilität

Wie kann Quellcode in verschiedenen Sprachen eingebunden werden?

- CORBA: spezielle Sprachbindungen IDL-to-*
  - Problem: Verwendung ORB-spezifischer Funktionen auf der Serverseite ist weit verbreitet. Damit nicht außerhalb einer schwergewichtigen Plattform lauffähig
- Java: So lange alles in Java geschrieben ist – kein Problem
  - direkte Übersetzung aus anderen Sprachen in Java Bytecode ist möglich (wird etwa bei J2EE-Implementierungen verwendet)
- COM: keine Standards jenseits der Microsoft de-facto Standards
- CLR und .NET: Interoperabilitätskonzept durch Sprachbindungsstandards

Speicherverwaltung und Garbage Collection

- Komplizierte Aufgabe in Systemen mit verteilten Objekten
- explizites Management des Lebenszyklus: CORBA
- Referenzzähler-Konzept: COM/DCOM
  - verlangt Kooperation aller Komponenten
  - skaliert schlecht in offenen verteilten Umgebungen
- Object leasing = Objektreferenzen haben nur beschränkte Lebensdauer
  - Java: GC von Java-RMI mit sehr guter Performance in verteilter Umgebung.
  - CLR: verwendet ähnlichen Ansatz

Containerverwaltetes Persistenz-Management

- Mit EJB eingeführt und mit EJB 2.0 auch auf Relationen ausgeweitet
  - sehr datenbankspezifisch, außerhalb dieses Einsatzgebiets nicht sehr performant
- OLE-Datenbanken: Konzept der Persistenz-Abbildung (pluggable persistence mapping) erlaubt Abbildung auf verschiedene externe Speichermedien

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Evolution und Versionsmanagement

- Sehr wichtig, wenn man Software-Entwicklung als Prozess verstehen will. Wird aber bisher kaum unterstützt
- COM: Schnittstellen und deren Spezifikation dürfen nach Veröffentlichung nicht verändert werden (immutable)
  - Aber: Möglichkeit der dynamischen Erweiterung
- CORBA: Versionsnummern, die zur Objektinitialisierung geprüft werden
  - Aber: dynamische Versionsprüfung nicht möglich
- Java: einige Regeln, aber inkonsistent
  - Problem der vorübersetzten Konstanten bei Versionswechsel
- CLI: Adressiert das Problem erstmals in voller Komplexität
  - Jede Assembly trägt Versionsinformationen von sich und allen Import-Komponenten. Es kann festgelegt werden, welche Toleranzen der Versionen erlaubt sind.
  - In einer Komponente können mehrere Versionen koexistieren
  - Standard wird weder von .NET noch von der CLR voll unterstützt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kategorien

- erstmals von COM zur Klassifizierung von Software eingeführt
- Kategorie = Schnittstellenkontrakt auf Komponentenebene
  - Konkrete Komponente kann zu mehreren Kategorien gehören
  - Kategorie = abstrakte Zusicherung (high level assertion)
- Java, CORBA: kennen dieses Konzept nicht (aber: Marker-Interface)
- CLI: Unterstützung über Nutzerattribute

Montage / Konfigurierung

- EJB 2: Attribute werden in der Montage-Beschreibung verwaltet
  - Erstmals Faktorisierung des Montage-Schritts
- J2EE: erweitert dieses Konzept auf andere Komponentenmodelle
- .Net und EJB 3: Inversion of Control und aspektorientierte Programmierung
  - Trennung von Installations-Konfiguration und zur Laufzeit erforderlicher Kontextinformation
  - damit werden die Rollen des Komponentenentwicklers und des Komponentenmonteurs klarer unterschieden
- CLR: kennt sowohl XML-basierte Konfiguration als auch CLI-basierte custom attributes

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Objekte

Objektorientierung: Ist es das Evangelium und Synonym für Qualität schlechthin oder nur ein Weg, um Qualität zu erreichen?

Prinzipien:

- Alles wird in Objekte zerlegt, die Zustand und Verhalten kapseln.
- Objekte sind Instanzen von Klassen; letztere durch das (traditionelle) Vererbungskonzept verbunden.
- Objekte in polymorphen Kontexten

Vorbemerkung: Java = OO + Sprache, CORBA, CLR sprachneutral

Java

- Alles ist aus Objekten (Ausnahme: ein paar primitive Typen)
- Vererbung auf Schnittstellen- und Implementierungsebene
- Polymorphie durch Subklassen und Subschnittstellen
- Klassen (nicht Objekte!) als Einheit der Kapselung
    - orthogonal dazu das Konzept der Pakete
- RMI: Ortstransparenz von Objekten in verteilten Umgebungen
- Persistenz von Objektidentitäten auf der Ebene von Basisdiensten, nicht per se.

CORBA

- Objekt als zentrales Konzept
- Klasse = Objektimplementierung, hat aber nichts mit Vererbung zu tun
- Mehrfachvererbung auf Schnittstellenebene, was ebenfalls die Basis für Polymorphie ist
- Kapselung durch Restriktion aller Interaktion auf Objektschnittstellen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CORBA (Fortsetzung)

- CORBA-Objekte sind recht gewichtig
  - Unterschied zwischen lokalen Objektreferenzen (kennt nur POA) und CORBA-Objektreferenzen
  - keine Unterstützung „kleiner" oder „serverloser" Objekte
  - zu teuer für jegliche Kommunikation innerhalb einer Komponente
  - OMG IDL kennt nur CORBA-Referenzen

CLR

- Einheitliches Typsystem mit **Object** als Wurzel, das Wert- und Referenztypen vereint
  - Instanzen der Basistypen sind keine Objekte, können aber wie solche behandelt werden.
- Objekte als Klasseninstanzen
- einfache Implementations- und mehrfache Schnittstellenvererbung
- Persistenz von Objektidentitäten auf der Ebene von Diensten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zusammenfassung:

- Java und CLR kommen OO-Prinzipien am nächsten
  - Ausnahme: Klassen, nicht Objekte als Kapselungseinheit
- COM und CORBA: Kapselungseinheit Objektserver, aber keine Konzepte der Interaktion von Objekten im selben Server
- Java, CLR: Unterscheiden zwischen internen und fernen Objektreferenzen. Letztere können nur über spezielle Infrastruktur (Java RMI, CLR Remote) angesprochen werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kontraktspezifikation für Komponenten

- „Bessere Kontrakte für bessere Komponenten“ [Szyperski, 19.5]
- Anforderungen an die Kontraktspezifikation sind höher als bei klassischer Software aus folgenden Gründen:
  - technologische Aspekte sind komplexer
  - Qualitäts-, Haftungs- und Sicherheitsfragen bei der Nutzung von Komponenten „Dritter“
- Wird in heutigen Komponentenkonzepten so gut wie nicht angesprochen
- QS ist nur bei klarer Spezifikation überhaupt kommunizierbar
  - Schnittstellen-Listing mit informeller Beschreibung (etwa auf der Ebene von **javadoc**) von Funktionalität reicht dafür nicht aus.
  - Qualität wird heute meist de facto durch starke Anbieter gesichert; am besten auch gleich im Kontext von Anwendungen dieser Anbieter
    - Beispiel: OLE und Word, Excel, Internet Explorer

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Problem der Behinderung der Entstehung einer Komponentenwelt durch Unterspezifikation
  - Bsp. CORBA: vom BOA zum POA
  - Erfahrung kommt erst im praktischen Einsatz konkurrierender Implementierungen desselben Standards
  - Es geht um Konvergenz der Interpretation des Standards
    - ausgewogene Balance von Enge und Freiheit
- Schnittstellenkontrakt von Komponenten ist immer mehr als die Spezifikation des „Zusammenschaltens“
  - wird immer informelle Elemente enthalten, da es (auch) ein sozialer Kontrakt zwischen Entwicklern und Nutzern von Komponenten ist
  - klarer Link zwischen Schnittstelle (als „Kontrakt-Instanz“) und Kontraktspezifikation erforderlich
- Zu jeder Schnittstelle gehört eine solche Spezifikation
  - Verbindung von Schnittstellen-Syntax und Semantik (Bedeutung)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Konzept der Kategorien: Spezifikation nach dem Baukastenprinzip
  - Java: Marker-Schnittstellen, COM: Kategorien
  - CORBA: Repository ID‘s verbinden eindeutige ID mit OMG IDL Typen
  - CLR: Konzept der Assembly sowie Konfigurationsattribute (custom attributes) zur Fixierung von Metadaten-Informationen
- Das sind bisher aber alles rein deklarative Methoden
  - Muss weiter formalisiert und in den Komponenten-Lebenszyklus (Entwicklung, Test, Zertifizierung, Laufzeit-Monitoring, ...) integriert werden
  - Entwicklungsrichtungen:
    - ASML = Abstract State Machine Language
      u.a. Werkzeuge zur automatischen Generierung von Testorakeln und -fällen
    - TTCN = Test and Test Control Notation Language
      des ETSI (European Telecommunications Standards Institute)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentensoftware und die Grundlagen der Softwaretechnik

- Komponentenansatz enthält eine Reihe neuer Herausforderungen für einen modularen Ansatz auch in der Software-Technik
  - Schlüsselproblem: Ansatz der unabhängigen Erweiterbarkeit
  - späte Integration von Komponenten unabhängiger Hersteller
    - Konflikte mit Integrationstestkonzepten der klassischen SWT
  - Erweiterbarkeit muss selbst „designed" werden, sonst passt nichts
  - Problem der verschiedenen methodischen Ansätze im SWE für interagierende Komponenten
  - Top-down-Design (ausgehend von der Anforderungsanalyse) trifft mit ziemlicher Sicherheit nicht die verfügbaren Komponenten
  - Bottom-up-Design ausgehend von Basisfunktionalitäten der verfügbaren Komponenten trifft mit ziemlicher Sicherheit nicht die Anforderungen
- Hier ist noch vieles unausgereift und ein umfassender Komponenteneinsatz nur aus strategischen Überlegungen heraus zu rechtfertigen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenorientiertes Programmieren als Methodologie

- Wie OOP die Methodologie des Programmierens objektorientierter Lösungen ist, so ist COP die Methodologie des Programmierens von Komponenten

- Definition (Szyperski):

  Komponenten-orientiertes Programmieren bedeutet Unterstützung von
  - Polymorphie (Substituierbarkeit)
  - modulares Kapseln (Verstecken von Information)
  - spätes Binden und Laden (unabhängige Auslieferbarkeit)
  - Sicherheit (Typ- und Modulsicherheit)

- Bisherige Methodologien erstrecken sich nur auf die Entwicklung einzelner Komponenten
- Neuere Entwicklungen: Die Catalysis-Methode
  - http://www.catalysis.org

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Montage

- Komponenten als Einheiten der Auslieferung durch Dritte und als Einheiten der Komposition
  - Ein Weg zur Komposition ist traditionelle Programmierung
  - Attraktivität von Komponenten nimmt zu, wenn einfachere Kompositions-Prinzipien verfügbar sind
    - visuelle Komposition in Grafik-Werkzeugen
    - zusammengesetzte Dokumente
    - Zusammenbinden durch Skripting
    - Zusammenbinden als Web Services
  - besonders attraktiv, wenn der Endnutzer diese Montage selbst vornehmen kann (IKEA-Prinzip)
- Alle diese vereinfachten Montage-Prinzipien setzen auf inhaltlicher Seite kontextuelle Kapselung und Komposition der Komponenten voraus

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Infrastruktur-Aufwand für Komponentenanbieter

- OMA: Jeder ORB-Anbieter muss seine Sprachanbindung zu allen unterstützten Sprachen herstellen
- COM: benötigt COM-Infrastruktur, die es im Wesentlichen nur für Windows gibt
- Java: Überall lauffähig, wo eine JVM läuft
  - ein Classfile-Compiler pro unterstützter Sprache ist erforderlich
  - es gibt solche Compiler für viele gebräuchliche Sprachen
  - JVM-Standard ist allerdings für die Verwendung mit Java optimiert
- CLI: verfolgt ähnliche Strategie wie Java, zielt aber auf eine breitere Unterstützung von anderen Sprachen
  - braucht so was wie die JVM auf allen unterstützten Plattformen
    - CLR als Implementierung auf .NET (Windows, Microsoft)
    - Open-Source-Projekte Mono und Open CLI Library Project
    - FreeBSD-Version von Corel und Microsoft

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Folgerung: Komponentenkonzepte müssen in eine (technische) Infrastruktur eingebettet sein.

Eine Lehre aus CORBA:

Wenn zu viele Dimensionen von Freiheit gekoppelt werden, um eine möglichst große Variation von Lösungen zu ermöglichen, dann werden die meisten praktischen Lösungen nur für Marktnischen relevant sein.

CORBA versagt bei einem seiner zentralen Versprechen: eine breite Varietät nicht nur von möglichen, sondern von realen Lösungen zu unterstützen. Es fehlen dafür strenge low-level Integrationsstandards.

Die Maximierung der Zahl der kombinatorisch möglichen Variationen minimiert die Zahl der real verfügbaren Varianten.

Für ein Komponentenmarkt ist die Freiheit der Inhalte ebenso entscheidend wie die Beschränktheit der Design-Konzepte.

Diese Standardisierungsbemühungen stehen noch ganz am Anfang.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Berufsprofile für Informatiker

Komponenten-Systemarchitekt

- Komponenten funktionieren nur innerhalb eines Frameworks (konkrete Implementierung eines Architektur-Konzepts)
- Konsistentes Architekturkonzept deckt mehrere Frameworks und deren Interoperabilität ab
- Entwickelt die Architektur für die Architekten - der einzelnen Frameworks

Komponenten-Frameworkarchitekt

- Entwicklung von Konzepten und Werkzeugen, um konkrete Komponenten in eine Infrastruktur „einzustöpseln“
- Muss sich im gesamten Anwendungsbereich des Frameworks gut auskennen
- Implementierung des Frameworks ist die Basis für eine funktionierende Komponentenwelt
- Muss Anforderungen an die Komponenten-Entwickler spezifizieren

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Entwickler

- Komponenten-Entwickler erstellen die „Blätter" für das Komponenten-Framework
- Die funktionalen Spezialisten in dieser arbeitsteiligen Struktur mit Spezialkenntnissen aus den konkreten Anwendungsbereichen, die von den zu entwickelnden Komponenten abgedeckt werden

Komponenten-Monteur

- Aufgabe: Anpassen, „Zuschneiden" und Integrieren von Komponenten für den konkreten Gebrauch in speziellen Anwendersystemen
- Auflösung des klassischen Begriffs der „Anwendung" als monolithisches und statisches System zugunsten des Konzepts einer organischen (und organisch wachsenden) IT-Infrastruktur
- End-Nutzer übernehmen in einem solchen Konzept zunehmend eine eigenständige Rolle, die vom Komponenten-Monteur abzugrenzen ist
- Aspekte der Nutzerschulung treten dann ergänzend hinzu
- Feedback zu Komponenten-Entwicklern und Framework-Architekten