Vorlesung
Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2004/05

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponentenkonzepte und Anforderungen im Vergleich
Eine Zusammenfassung

- Komponententechnologie und Softwaretechnik (war schon)
- 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

Zwei grundlegende Herangehensweisen

- eng gekoppelte Architektur (J2EE, CORBA, OLE und CLR)
  - Zusammenschalten der Rechner zu einem „verteilten Betriebssystem“ 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

Die Komponentenmodelle, die in der VL vorgestellt wurden

- Microsoft-Entwicklungen
  - COM, COM+ und DCOM sowie OLE
  - .NET, CLR, C#
- Standards der OMG
  - CORBA und das ORB-Modell
  - OMA und die Klassifizierung von Basisdiensten
  - CCM als Komponentenmodell für Applikationsserver
- Java-basierte Konzepte
  - J2SE als plattformunabhängiges Entwicklungskonzept und Framework für Basiskomponenten und Desktop-Anwendungen
  - J2EE als generelles Framework für eng gekoppelte verteilte Anwendungen, EJB, Servlets, Server Pages

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Gemeinsamkeiten der eng koppelnden Konzepte

Alle Zugänge unterstützen

- spätes Binden, Kapselung, dynamische Polymorphie, Vererbung auf Schnittstellenebene

Konvergenz auf der Ebene der Konzepte

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Konvergenz (2)

- Einsatz von Konfigurationsinformationen
  - attributbasiertes Programmieren (CLI) – custom attributes
  - Montage-Beschreibungen
- spezielle Komponentenkonzepte für Applikationsserver
  - EJB, COM+, CCM
- spezielle Komponentenkonzepte für Webserver
  - JSP/Servlets, ASP.NET
- dynamische Konfiguration
  - COM: QueryInterface, CORBA: EquivalenceInterface
- Idee: Komponentenobjekt kann mit mehreren Servantenobjekten verbunden sein (multibodied instances)
  - JavaBeans: Reimplementierung der Methode **instanceOf** in **java.beans.Beans**, um die Funktionalität mehrerer Objekte über eine Beans-Schnittstelle verfügbar zu machen
  - geht nur im Kontext von Schnittstellenvererbung (interface inheritance), nicht für Implementierungsvererbung (implementation inheritance)

Bestehende Differenzen können oft durch Brückenlösungen überwunden werden, die Drittanbieter zur Verfügung stellen.

> Our motto is incompatibility is business – it‘s a huge opportunity for us.
>
> (A. O‘Toole, CTO von IONA)

Binärstandards

- COM: grundlegendes Ziel, wenn auch weitgehend ohne Bedeutung außerhalb der Windows-Welt
- Java: Binärstandard an JVM über JNI gebunden
- CORBA: Nur im Rahmen von CORBA-to-* Compilern. Nicht plattformübergreifend standardisiert
- CLR: Ähnlich Java eine Ebene über Binärstandards angesiedelt, aber direkte und JIT-Compilation vorgesehen
  - CLR übersetzt entweder zur Installations- oder zur Ladezeit und führt immer nativen Code aus.

Quellcodestandards für Kompatibilität und Portabilität

- CORBA: spezielle Sprachbindungen IDL-to-*
  - Problem: Verwendung ORB-spezifischer Funktionen auf der Serverseite ist weit verbreitet
- Java: So lange alles in Java geschrieben ist – kein Problem
  - direkte Übersetzung aus anderen Sprachen in Java Bytecode möglich
  - Beispiel: J2EE-Implementierungen
- 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

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

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

- Konzept der attributgesteuerten Programmierung
  - EJB: Attribute werden in der Montage-Beschreibung verwaltet
    - Faktorisierung des Montage-Schritts
  - J2EE: erweitert dieses Konzept auf andere Komponentenmodelle
  - CLR: kennt sowohl XML-basierte Konfiguration als auch CLI-basierte custom attributes
    - damit werden die Rollen des Komponentenentwicklers und des Komponentenmonteurs klarer unterschieden

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, COM, CORBA, CLR sprachneutral

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

COM

- Objekte nur über Schnittstellenmengen zugänglich
- keine zugänglichen Objektreferenzen, nur Schnittstellenreferenzen
- Objekte als Klasseninstanzen, aber ohne Vererbung
- Polymorphie wird durch n:m-Beziehung zwischen Schnittstellen und Klassen erreicht.
- Persistenz von Objektidentitäten auf der Ebene von Diensten

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
- 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CLR (Fortsetzung)

- Objekte als Klasseninstanzen
- einfache Implementations- und mehrfache Schnittstellenvererbung
- Persistenz von Objektidentitäten auf der Ebene von Diensten

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, COM, CLR: Unterscheiden zwischen internen und fernen Objektreferenzen. Letztere können nur über spezielle Infrastruktur (Java RMI, DCOM, CLR Remote) angesprochen werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Frage: Ist das objekt-orientiert?

praktisch nicht relevant, sondern nur

Was kann man damit anfangen?

Antwort:

Grob-granular partitionierte verteilte Anwendungen entwerfen, in welchen die über das Framework zugänglichen Objekte relativ groß sind.

Java und CLR sind auch für den Entwurf von Anwendungen mit kleineren Objekten auf Programmiersprachen-Ebene geeignet.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Objektmobilität und mobile Agenten

- spezieller Anwendungsaspekt: Object-Roaming in mobilen Netzen
- wird von keinem der Konzepte berücksichtigt, da Objekte generell an (wohl lokalisierte) Serverkontexte gebunden sind.
- Mobilität von Objekten ist auch als Transfermedium interessant:
    - Objekte als Datenkapsel mit Zugriffsschnittstelle
- benötigt ein Konzept der „mobilen Objekte“ und Übertragung aus einem (lokalen) Kontext in einen anderen
    - „Beamen“ von Objekten ist mehr als Marshalling

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

Kontraktspezifikation für Komponenten

- Problem der Behinderung der Entstehung einer Komponentenwelt durch Unterspezifikation
  - Bsp. CORBA: vom BOA zum POA
  - Bsp. J2EE: von EJB zu EJB 2.0
  - 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)
  - COM-Konzept der unveränderlichen Schnittstelle ist Reflex dieser Tatsache
  - COM-UID = „Link“ zu einer solchen Spezifikation

Kontraktspezifikation für Komponenten

- 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 Komponenteneinsatz nur aus strategischen Überlegungen heraus zu rechtfertigen.

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

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

Der deutliche Sieger in diesem Rennen ist Java und CLI ist der Versuch, diese Erfahrungen mit denen der COM-Welt zu vereinigen

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