Vorlesung
Component Ware und Web-Services

- Komponentenkonzepte -

15. Komponentenansätze im Vergleich

Prof. Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

Inhalt

- Komponententechnologie und Softwaretechnik
- Komponentenkonzepte im Vergleich
- Konvergenz der Konzepte
- Differenzen der Konzepte
- Konsequenzen für eine Komponenten-Infrastruktur

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

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

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

Komponentenkonzepte im Vergleich

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.
  - feingranulares 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

Komponentenkonzepte im Vergleich

Die Komponentenmodell, die in der VL vorgestellt wurden

- Microsoft-Entwicklungen
  - COM, COM+ und DCOM sowie OLE
  - MTS Microsoft Transaction Server
  - .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
  - J2EE als generelles Framework für eng gekoppelte verteilte Anwendungen, EJB
- Webservices
  - Die Rolle von XML und seinen Unterstandards als Markup für standardisierten Datenaustausch
  - Aufbau spezieller Protokolle SOAP, WSDL, UDDI

Gemeinsamkeiten

Gemeinsamkeiten der eng koppelnden Konzepte

Alle Zugänge unterstützen

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

Konvergenz auf der Ebene der Konzepte

- Verwendung von Komponenten-Transfer-Format
  - 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

Gemeinsamkeiten

Konvergenz (2)

- Einsatz von Konfigurationsinformationen
  - attributbasiertes Programmieren (CLI) – custom attributes
  - Deployment-Deskriptoren
- 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 des ‚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)

Differenzen

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

> Out 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.

Differenzen

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: Interoperabilitätskonzept durch Sprachbindungsstandards

Gewachsene versus gesetzte Standards

- je neuer, desto kürzere „Wachstumszeiten", bis zu Standardisierungsversuchen
- unterschiedlich starke Märkte
  - ActiveX: einige 1000 Komponenten,
  - Beans: einige Dutzend mit stark steigender Tendenz,
  - CLR und C#: für Prognosen noch zu früh

Differenzen

Speicherverwaltung und Garbage Collection

- Komplizierte Aufgabe in Systemen mit verteilten Objekten
- CORBA: keine Konzepte
- COM/DCOM: Referenzzähler-Konzept
  - verlangt Kooperation aller Komponenten
  - skaliert schlecht in offenen verteilten Umgebungen
- Java: JavaRMI mit sehr gutem Modell für GC in verteilter Umgebung.
  - Idee: Object leasing = Objektreferenzen haben nur beschränkte Lebensdauer
- 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

Differenzen

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)
  - Möglichkeit der dynamischen Erweiterung
- CORBA: Versionsnummern, die aber nur zur Objektinitialisierung geprüft werden
  - 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

Differenzen

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
  - Marker-Interface
- CLI: Unterstützung über Nutzerattribute

Deployment / Konfigurierung

- Konzept der attributgesteuerten Programmierung
  - EJB: Attribute werden in speziellem Deployment-Deskriptor verwaltet
    - Faktorisierung des Deployment-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 Deployers klarer unterschieden

Differenzen

Unterstützung von Webservices

- ältere Ansätze (COM, CORBA) haben keine speziellen Komponentenmodelle für Webservices
- J2EE: JSP und Servlets
- .NET: ASP.NET
- Entwicklung geht hin zu stärkerer Trennung zwischen Darstellung (rendering, HTML) und Funktionalität
- Einige neuere Entwicklungen (CLR) unterstützen auch XML-Formate zum Datenaustausch auf Applikationsebene sowie spezielle Webservice-Protokolle wie SOAP

Konsequenzen für Komponenten-Infrastruktur

Infrastruktur-Ansätze in den verschiedenen Komponentenmodellen

- COM: Ausgerichtet auf binäre Aufrufstandards und –Schnittstellen
- Java: Schnittstellenformat durch Java-Standard vorgegeben
  - Load-File-Format mit allen Metainformationen
  - Standard für Serialisierung von Objekten (interface Serializable)
- CLI: Schnittstellenstandard
  - klassenbasiertes Framework in sprach-unabhängigem Format
  - Load-File- und Metadaten-Format (Assemblies)
  - mehrere Serialisierungsprotokolle
- OMA: kein Infrastrukturstandard vorgeschrieben
- COM: benötigt COM-Infrastruktur, die es im Wesentlichen nur für Windows gibt

Konsequenzen für Komponenten-Infrastruktur

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

Konsequenzen für Komponenten-Infrastruktur

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.