Vorlesung
Software aus Komponenten

5. Komponentenkonzepte im Vergleich

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2013/14

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verteilte 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

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

Erster Ansatz: Schnittstellen und deren Spezifikation dürfen nach Veröffentlichung nicht verändert werden (immutable interfaces)

CORBA: Versionsnummern, die zur Objektinitialisierung geprüft werden

- Damit *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: Erstmals Abtrennung der Montage als eigenständiger Schritt und Beschreibung in einem deployment descriptor
- J2EE: erweitert dieses Konzept auf andere Komponentenmodelle
- .Net, EJB 3, Spring: 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

Szyperski: „beyond object oriented programming.“

Grundprinzipien der Objektorientierung:

- Zerlegung des globalen Raums der Programmzustände in lokal abgrenzbare interagierende Einheiten (Objekte)
- Objekte kapseln Zustand und Verhalten
- Objekte sind Instanzen von Klassen und werden damit von einer überschaubaren Zahl prototypischer Generatoren erzeugt
- Diese Generatoren sind als Klassen durch das (traditionelle) Vererbungskonzept verbunden.

Weitere Fragen:

- Ortstransparenz der Objekte in verteilten Umgebungen
- Persistenz von Objektidentitäten

Java

- Alles ist aus Objekten (Ausnahme: ein paar primitive Typen)
- Fokus auf den Klassen und ihren Interaktionen, nicht den Interaktionen der Objekte. Damit steht das Verhalten im Vordergrund. Klassen als Kapselungseinheit, denn nur über diese können Instanzen verwaltet werden.
    - orthogonal dazu das Konzept der Pakete
- Ortstransparenz von Objekten in verteilten Umgebungen über eigenständige Konzepte wie RMI
- Persistenz von Objektidentitäten nur auf der Ebene von Basisdiensten, nicht per se.

CORBA

- Objekt als zentrales Konzept. Damit steht der Zustand im Vordergrund.
- Klasse = Objektimplementierung, hat aber nichts mit Vererbung zu tun. Abtrennung der Funktionalität in die Servanten, die aber über die Objekte aufzurufen 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 den klassischen 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
  - Objekte als Kapselung von Zustand und *potenziellem* Verhalten (Schnittstellendefinition)
  - Davon abgetrennt Servanten als Ort der Realisierung *tatsächlichen* Verhaltens
  - Damit näher an der Trennung Komponente – Objekt dieser Vorlesung
- Moderne Komponentenkonzepte:
  - Identifizierung des Kontexts als eigenständiger Quelle tatsächlichen Verhaltens (cross cutting concerns)
- Java, CLR: Unterscheiden zwischen internen und fernen Objektreferenzen. Letztere können nur über spezielle Infrastrukturdienste (Java RMI, CLR Remote) angesprochen und verwaltet werden

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

- 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, http://research.microsoft.com/en-us/projects/asml/
    - TTCN-3 = Testing and Test Control Notation Language des ETSI (European Telecommunications Standards Institute) Die Hauptaufgabe von TTCN-3 besteht darin, in beliebigen Sprachen implementierte Protokolle und Anwendungen zu testen.

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.
- Entwicklung in Richtung modellgetriebener Ansätze sowie „Software as a Service“.

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
- 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
  - Bekannteste Vertreter der JVM-Sprachen sind Groovy, Scala, der Lisp-Dialekt Clojure sowie die Portierungen anderer Sprachen wie Jython (Python), JRuby (Ruby), Rhino (JavaScript) und Erjang (Erlang).
- 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, http://www.mono-project.com/

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