

Vorlesung Software aus Komponenten

2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2014/15

Schnittstellen und Versionen

- Problem: Schnittstellen können ihr Verhalten zwischen Versionen wechseln
- Management traditionell über Versionsnummern
 - Komponente als unteilbare Einheit => Versionsnummern nur für ganze Komponenten
- Versions-Information in Import- und Exportschnittstellen
 - direkte Schnittstellen: Abfrage zur Bindungszeit, also (nur) vor dem ersten Schnittstellenaufruf
 - indirekte Schnittstellen: Abfrage vor jedem Aufruf erforderlich
 - Alternative: Integration ins Management der Objektidentität
- Problem der Versions-Information, wenn eine Objektreferenz Komponentengrenzen überschreitet
 - Objekt bietet eigene Dienste an
 - etwa Rückgabe in einem bestimmten Format
 - Objekt nutzt Dienste anderer => dynamische Versionskontrolle

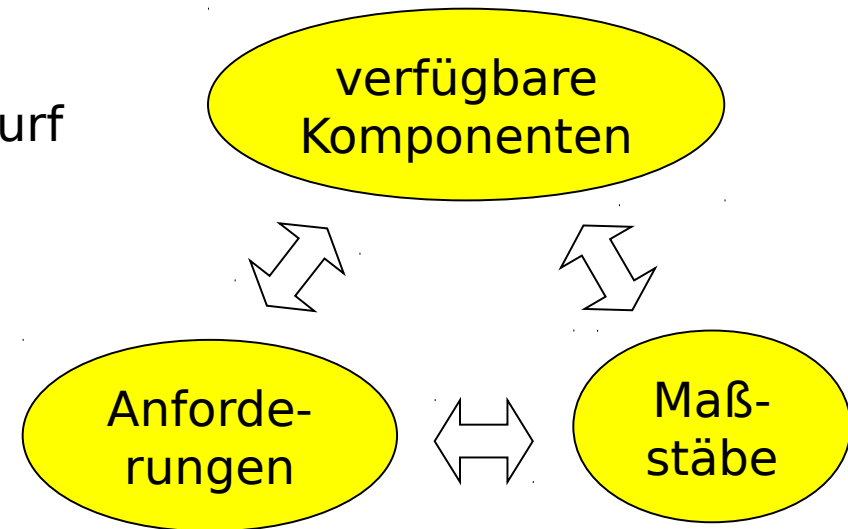
- Schnittstellenversionen müssen klar als kompatibel oder klar als veraltet (deprecated) deklariert werden können
- Ansatz der unveränderlichen Schnittstellen-Spezifikation (immutable interfaces)
 - Neue Versionen nur als neue Schnittstellen
 - Veraltete Schnittstellen werden einfach nicht mehr unterstützt
 - Beispiel: COM = Component Object Model von MicroSoft
- Veränderbare Schnittstellen-Spezifikationen:
 - klares Kompatibilitäts-Konzept erforderlich
 - Installation verschiedener Komponentenversionen in derselben Umgebung kann erforderlich sein.
 - Unterscheidung zwischen Komponenten, die immer in der aktuellsten Version verwendet werden können und Komponenten, die nur in der ursprünglich installierten Version verwendet werden können
 - wird im Rahmen der CLR = Common Language Runtime verfolgt

Komponenten abgrenzen

Problem: Wie ist ein kompletter Entwurf in Komponenten zu partitionieren?

Prinzipien:

- Modularität und Kapselung
- Abhängigkeiten zwischen K. sind expliziert
- Mehrere Abstraktionsebenen, hierarchische Strukturierung
- natürliche Zuordnung von Verantwortlichkeiten
- Migrations-Erfordernisse vorab berücksichtigen



- Wie „fett“ soll eine Komponente sein?
 - optimal, aber unreal: „richtige Schnittstellenmenge“ und keine Kontext-Abhängigkeit
 - maximal: „fette“ Komponente, die alle benötigten Dienste mitbringt (=Applikation, grobkörnig)
 - minimal: Auslagern aller bis auf die zentrale Funktionalität (=Klasse, feinkörnig)
 - „Maximizing reuse minimizes use.“
 - Grund: Explodierende Kontext-Abhängigkeit
 - würde nur unter statischen Entwicklungsbedingungen funktionieren
 - Beispiel: Linux-Probleme mit Bibliotheksversionen
 - praktisch ist hier ein je ausgewogenes Mittel zu finden
- Je detaillierter Normierung und Standardisierung, desto schlankere Komponenten sind möglich
 - Standardisierung ist in vertikalen Marktsegmenten (funktional) eher möglich als in horizontalen, aber wegen der geringen Marktgröße schwieriger

Komponenten und Systemdesign

Komponenten und Analyse

- Analyse komplexer Systeme erfolgt durch Zerlegung in handhabbare Einheiten, Tiefenanalyse der Einheiten und Zusammensetzung von Extrakten der Analyseergebnisse zum Gesamtsystem
- Ansatz: white box → black box
- Entwurfsexpertise lässt sich auf diese Weise kapseln und wiederverwenden (design expertise ready for use)
- Komponentenzuschnitt längs dieser Abstraktionsgrenzen erleichtert das Verständnis des Gesamtsystems

Komponenten aus technischer Anbietersicht

Komponenten und Compilierbarkeit

- Compilierbarkeit kann aus Sicht der Auslieferung interessant sein
- Compilierbarkeit und Analyse sind eng verbunden
 - white box: Analyse
 - black box: Compilierbarkeit
- Compilierbarkeit und Optimierung
 - Optimierung ist ein globales Phänomen, deshalb Antwort 1: Einheiten möglichst groß wählen
- Antwort 2: Optimierung *zwischen* Komponenten, vielleicht sogar erst nach der Lokalisierung
 - Verfahren muss allerdings im Komponentenkonzept und in der Komponentenbeschreibung verankert sein

Komponenten als Einheit der Packung

Komponenten und der Auslieferungsprozess

- Entpackung (deployment) = Prozess der Vorbereitung der Komponente auf den Einsatz in einer speziellen Umgebung (Lokalisierung)
 - wurde lange nicht als separater Schritt betrachtet
 - Prozess der Anbindung an eine spezielle Komponentenplattform
- Konfiguration = Einstellung spezieller Eigenschaften der Komponente für den konkreten Einsatz
- Installation = plattformspezifische Aktivität, mit der eine entpackte Komponente für die Nutzung in einer speziellen Hardware-Konfiguration verfügbar gemacht wird, die von der Plattform unterstützt wird.
 - Zeit, in der auch kritische Tests ausgeführt werden, die vor dem eigentlichen Betrieb erfolgen müssen (etwa Integritätstests)
- für alle drei Aktivitäten müssen entsprechende Beschreibungen erstellt werden

Komponenten und Management

K. als Einheit der (Auseinandersetzung um) Fehlersuche

- Problem: Was ist (u.a. wer haftet?), wenn ein aus Komponenten zusammengebautes Produktivsystem fehlerhaft arbeitet?
- Problem der Lokalisierung von Fehlern (und damit Verantwortlichkeiten)
 - besonders schwierig wird es, wenn Objektreferenzen die Komponentengrenzen verlassen
- vitale Regel: Fehler müssen in den verursachenden Komponenten bleiben (bug containment)
 - typische nicht-lokalisierbare Fehler: Speicherzugriffsfehler
- Folge: Ausnahmebehandlungen müssen in der Regel innerhalb einer Komponente bleiben
 - Ausnahmen davon sind im Komponenten-Kontrakt zu fixieren
 - Komponente als Einheit der Fehlerbehandlung

K. als Auslieferungseinheit

- Bündel der technischen und wirtschaftlichen Aspekte
- Management (Service, Wartung, Schulung, Updates, ...) treibt den Preis in die Höhe
- betriebswirtschaftliche Bedeutung jenseits der (geringen) Replikationskosten

K. als Einheit der Kostenrechnung

- wichtig in größeren industriellen Kontexten, um Projektentwicklungskosten verfolgen zu können

K. als Einheit des Managements

- oft zu klein, Management auf der Ebene von Subsystemen, die mehrere Komponenten zusammenfassen
- etwa auf der Ebene der Server

Begriffsbildung: Im Gegensatz zum OO-Ansatz wird die Unterscheidung zwischen Programm und Daten wieder stärker betont

- Verhalten wird von **Komponente** bestimmt, gehört zur *Designzeit* und in die Verantwortung der Komponenten-*Entwickler*
- Attributwerte sind in **Objekten** gekapselt, werden zur *Laufzeit* geändert, Interpretation der Werte gehört zur Welt der Komponenten-*Nutzer*

Praktische Granularitäts-Ebenen von Komponenten:

- Ebene einzelner Desktop-Anwendungen – Intraprozesskommunikation, Zusammenbinden einzelner Funktionalitäten zu einem Gesamtprozess
- Ebene Middleware – Interprozesskommunikation, Zusammenspiel mehrerer Prozesse, auch auf verschiedenen Rechnern, zur Erfüllung einer Gesamtaufgabe / Anwendung (unser weiteres Thema)
- Ebene Geschäftsprozessmodellierung – Zusammenspiel mehrerer Anwendungen innerhalb eines betrieblichen Informationssystems (Thema der VL „Betriebliche Informationssysteme“ usw.)

Vorlesung Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2012/13

Komponenten-Modelle

1. Erste Komponentenansätze
2. RPC als Middleware Kommunikationskonzept
3. OMG und CORBA, Common Object Request Broker Architecture
4. Sun und Java, Servlets und JSP, EJB
5. Microsofts und .NET, Common Language Runtime
6. Neuere Komponenten-Frameworks

Der dokumentenzentrierte Ansatz

Idee: Nutzer wird nicht mit vielen verschiedenen Applikationen konfrontiert, sondern mit Dokumenten, die aus mehreren Teilen bestehen können. Diese Teile können unterschiedliche Applikationen zur Darstellung benötigen, kennen diese aber selbst.

Erste Realisierung unmittelbar auf der Ebene von integrierten Textdokumenten: Hypercard (Apple), Word mit Visual Basic und VBX (Microsoft)

OLE als Weiterentwicklung dieses Ansatzes

Formulare \Rightarrow Container für beliebige Anwendungen

Kontrolleinheit \Rightarrow Dokumentenserver

Container können hierarchisch ineinander geschachtelt werden

Der webzentrierte Ansatz

- Idee: Einbettung von beliebigen Objekten in HTML-Seiten
 - z.B. Java Applets, Form-Bestandteile
- Einheitliche und erweiterbare Darstellung im Browser durch Plugin-Technologie
- Schritt weg vom OLE-Containerkonzept und zurück zum (nicht hierarchischen) Formularansatz von Visual Basic

Aktuelle Entwicklungsrichtungen

- COM und .NET (Microsoft)
- CORBA (Object Management Group)
- Java und Java-Frameworks (EJB, Spring, OSGi)
- Webservices als lose gekoppeltes Konzept

Interprozess-Kommunikation (IPC) auf OS-Ebene

Charakteristika

- kaum plattformübergreifend standardisiert
- nicht Teil des von-Neumann-Modells
- Prozess = virtueller Rechner auf physischem Host

IPC-Modelle: Dateien, Pipes, Sockets, Semaphore, shared memory

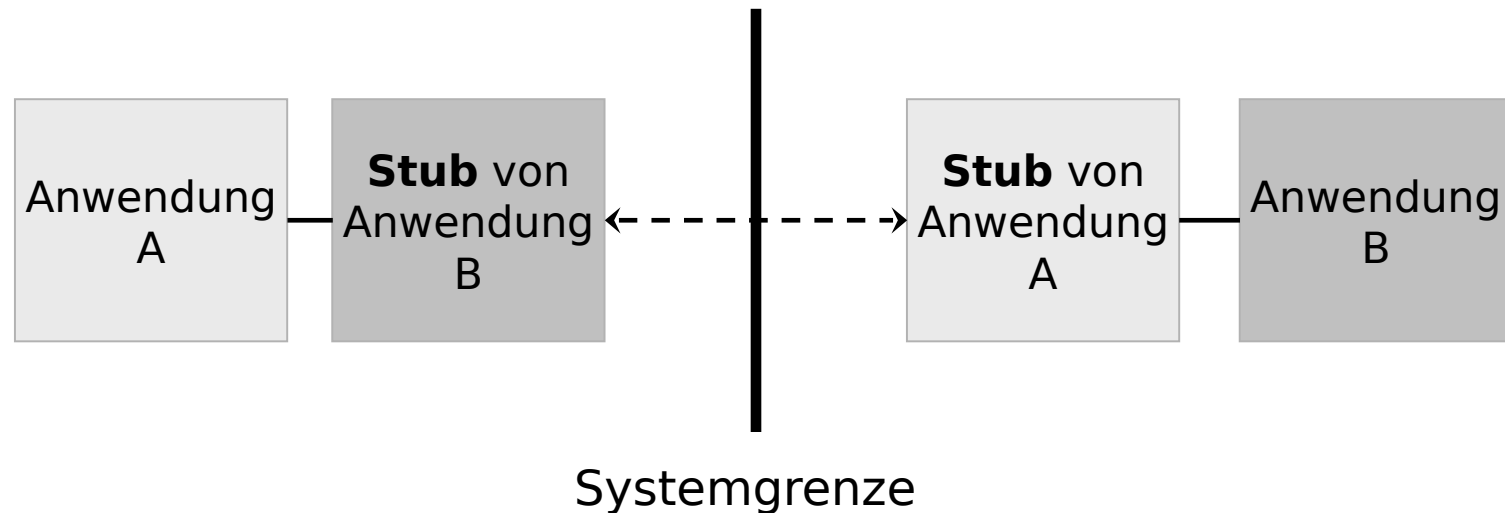
- außer Sockets keins so weit standardisiert, dass es plattformübergreifend eingesetzt werden könnte.
- außer shared memory skalierbar und internetfähig
- operiert auf Bitebene. Das ist zu kompliziert für komplexe Anwendungen

IPC operiert auf Bitebene und ist deutlich zu kompliziert für komplexe Anwendungen.

Remote Procedure Calls (RPC, 1984)

Ansatz: Stubs, die auch entfernte Prozeduraufrufe lokal aussehen lassen

- Aufgabe des Stub: Serialisierung bzw. Deserialisierung des Prozeduraufrufs und der Aufrufparameter unter Beachtung von plattformabhängiger Byte-Kodierung, Zahldarstellung, ...



3.2 Kommunikationskonzepte

Remote Procedure Calls

- Vorteil: einheitliches Abstraktionsniveau für alle Kommunikationserfordernisse (innerhalb eines Prozesses, zwischen Prozessen, zwischen Computern)
- Nachteile:
 - Versteckte Kommunikationskosten (Unterschied um Faktor $10 \dots 10^4$), Client kann nicht unterscheiden, ob lokaler oder entfernter Aufruf
 - blockierendes Konzept
 - Umgang mit Versionierung und Evolution von Komponenten vollkommen unklar

Das RPC-Konzept bildet zusammen mit dynamisch linkbaren Bibliotheken (DLL) die Basis für das einfachste Komponenten-Framework (und ist das heute am weitesten verbreitete).