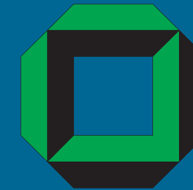
Software aus Komponenten



Prof. Dr. Gerhard Goos
Fakultät für Informatik
Universität Karlsruhe

Karlsruhe, Germany 2004

© Gerhard Goos, Dirk Heuzeroth,
Elke Pulvermüller, Volker Kuttruff
2004

http://www.info.uni-karlsruhe.de/

Organisation

- Vorlesung:
  - 1. Dienstag 14:00 – 15:30, -101, wöchentlich
  - 2. Mittwoch 14:00 – 15:30, -101, vierzehntägig
- Übungen:
  - Mittwoch 14:00 – 15:30, -101, vierzehntägig, erstmals 28.04.2004
- Übungsleiter: Dr. Dirk Heuzeroth, Mamdouh Abu-Sakran, Volker Kuttruff
- Adressen:
  - Allgemeines Verfügungsgebäude, 2. OG
  - ggoos@ipd.info.uni-karlsruhe.de
  - heuzer@ipd.info.uni-karlsruhe.de, msakran@ipd.info.uni-karlsruhe.de, kuttruff@fzi.de
- Sprechstunden:
  - Goos: Dienstags 16:00 – 17:00
  - Heuzeroth, Abu-Sakran, Kuttruff: Mittwochs 16:00 – 17:00
- Vorlesungsseite:

  `http://www.info.uni-karlsruhe.de/lehre/2004SS/swk`

Gliederung

1. Einführung
   - Motivation
   - Definition,
   - Konzepte für Komponenten (klassische, kommerzielle, akademische)
2. Industrielle Komponentensysteme der 1. Generation
   - 2.1 CORBA
   - 2.2 Enterprise JavaBeans
   - 2.3 (D)COM, .Net
   - 2.4 Webdienste, XML
3. Akademische Ansätze
   - Architektursysteme
   - Aspektorientiertes Programmieren
   - Metaprogrammierung und Invasive Komposition

2. Konkrete industrielle Systeme

- Pragmatisch motiviert
  - Wie baue ich große Systeme in einer heterogenen Welt?
- Wie baue ich große Systeme in einer heterogenen Welt?
- Man braucht Mechanismen zur Transparenz von
  - Sprachen (IDL)
  - Plattformen
  - Betriebssystemen
  - Ausführungsort:
    - Global eindeutige Referenz
    - Stellvertreterobjekte (*stub/skeleton*)
    - Wiederverwendbare Dienste zum Ermitteln von Komponenten und deren Eigenschaften (z.B. für dynamische Komposition):
      - Namensdienst
      - Makler
      - Persistenz (Objekte aus Datenbank holen)
      - Reflexion (Schnittstellen erfragen)
    - Objektverwaltung (Lebenszyklus)
- **Jetzt:** Ausprägung in
  - Corba (Mustergültig)
  - Java / JavaBeans / EnterpriseJavaBeans (Sprachgebunden)
  - COM/DCOM/.NET (Plattformgebunden)
  - Webdienste/XML

Java, JavaBeans und EnterpriseJavaBeans

- Java: Plattformunabhängige Programmiersprache
- JavaBeans: Komponentenarchitektur für Java mit dem Ziel visueller Komposition
- EnterpriseJavaBeans: Industrielles Komponentenmodell für Java

Java

- Ursprung von Java: Eingebettete Systeme für Hausgeräte
  - Neue Sprache an C++ angelehnt
  - Reduktion der Komplexität
  - Exakt und sauber spezifiziert
- Verbreitung durch Java-Applets in Netscape (1994)
  - Java benutzt das Web als Zugpferd, so wie C UNIX
  - *Write once, run anywhere* (WORA) durch *Virtual Machine*
  - Frei nutzbar, doch feste Schnittstellen (Lizenzvertrag)
- Klassenbibliotheken:
  - AWT, Swing (GUI)
  - Java Beans (1997)
  - Enterprise Java Beans (1998/99)

Standardisierung

- Java gehört Sun Microsystems
  - Definition von Schnittstellen in einem öffentlichen Prozess
  - Jeder kann Vorschläge einbringen
  - Java Community Process, Java Developer Connection
  - Ziel: Abwenden von Zersplitterung durch Konkurrenten
- Ob Java ein ISO-Standard wird, ist fraglich
  - Standard eingereicht, dann zurückgezogen
- Was passiert, wenn ein proprietäres Java den Markt dominiert?

Java: Spracheigenschaften

- Objektorientiert
- Mehrfachvererbung von Schnittstellen
- Einfachvererbung von Code
- Automatische Speicherbereinigung
  - Speicher ist uniforme Halde
  - Benutzerdefinierte Verwaltung unmöglich
- Keine Zeiger, nur Referenzen
  - Keine Funktionsparameter
  - *Callbacks* durch Hilfsklassen möglich
- Synchronisation mit Monitoren
- Einfaches, hierarchisches Paketkonzept
  - Entspricht Namensräumen
- Keine Generizität (bis Version 1.5)
- Extraktion von Dokumentation (javadoc)
- Annotation weiterer Metainformationen ab Version 1.5 möglich (wie Attribute in C#)

Klassen und Dateien

- Einbettung in Dateisystem
  - Eine öffentliche Klasse pro Datei
  - Pakete entsprechen Verzeichnissen
  - Auffindung: Suchpfad CLASSPATH
    - Definiert Abbildung von Namen auf Klassen
    - Primitive Registrierdatenbank
    - Pendant zu CORBA interface/implementation repositories
    - Eintragen von Klassen durch Kopieren im Dateisystem
- Java Archive Format (JAR)
  - Im Prinzip ZIP
  - Klassendateien
  - Serialisierte Objekte
  - Dokumentation

Sprach- und Plattformtransparenz: Virtuelle Maschine

- Java Virtual Machine
  - Abstraktion von konkreter Maschine und Plattform
  - prinzipiell sprachunabhängig
  - Übersetzer für andere Sprachen, z.B.
    - ADA, COBOL, ECMAscript, Eiffel, Scheme, . . .
- Java Bytecode
  - Stapelorientiert (Interpretation durch Java Virtual Machine)
  - Plattformunabhängig
  - Volle Typinformation
  - Sicherheit durch *bytecode verifier* prüfbar

Leistungen und Dienste der Bibliothek

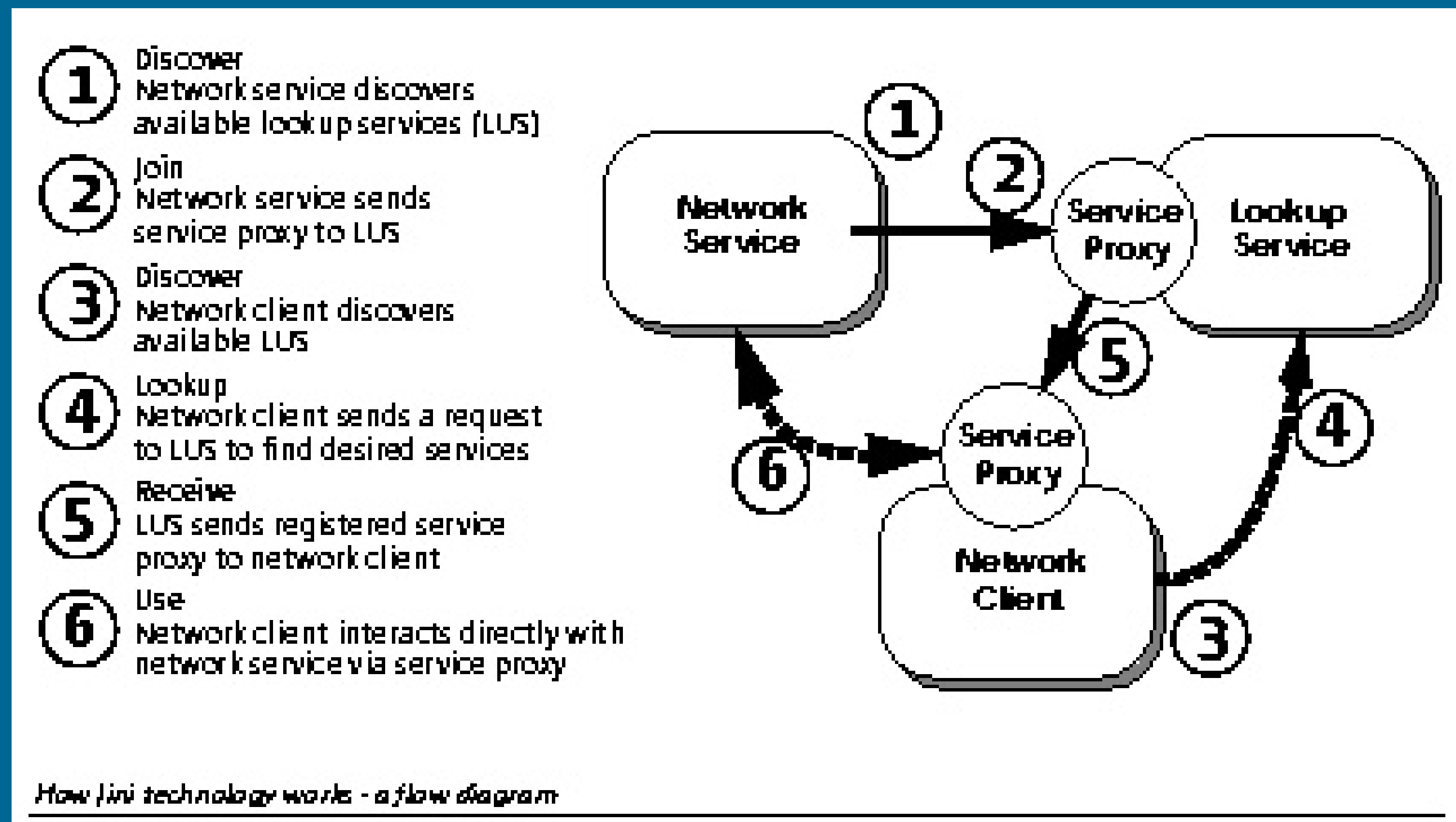
- Ortstransparenz:
  - Verteilung mit Java RMI (Remote Method Invocation)
  - Dienste (normierte Schnittstellen zu (Fremd-)Diensten):
    - Namen über *Java Naming Directory Interface* (JNDI)
    - Makeln mit *Java Intelligent Network Infrastructure* (JINI)
    - Persistenz durch Serialisierung mit `java.io.ObjectOutputStream`
    - Datenbankzugriff über *Java Database Connection* (JDBC)
    - Transaktionen mit *Java Transaction Server* (JTS)
    - Reflexion mit Klassenobjekten `java.lang.Class`
      - Zugriff auf Felder mit `java.lang.Field`
      - Zugriff auf Methoden mit `java.lang.Method`
- „Sprachtransparenz“: Anbindung fremder Sprachen über *Java Native Interface* (JNI)
- Parallelität mit `java.lang.Thread`
- Ereignisse mit `EventListener`
- *Java Spaces* (*Tuple space* à la LINDA, typisierte Whiteboards)

Ortstransparenz — Makler: JINI

- *Java intelligent network interface* (JINI)
  - Dynamischer Aufruf, Dienstvermittlung
  - Dienstbeschreibung: Schnittstellen, Eigenschaften (*properties*)
  - Erzeuge Stellvertreter für Dienst auf Kundenseite
  - Nutzt RMI, JNDI, Migration
  - Verschickt Klassen im Bytecode
- Makler suchen Dienste im Netz (*lookup service LUS*)
  - Verschiedene Suchprotokolle
- *Leases*-Schnittstelle erlaubt Miete von Diensten
- Organisation von Diensten in Gruppen
  - Ein Drucker kann sich der Hardware-, Drucker- oder Raum111-Gruppe zuordnen

JINI - Generelles Schema

- Anbieten
  - Suchen des Maklers (*discovery*)
  - Anbieten beim Makler (*join*)
  - Dienst wird nur zeitbegrenzt angeboten (*leasing*)
- Finden
  - Suche nach Makler (*discovery*)
  - Suche nach Dienst (*lookup*)
  - Benutzung (*use*)
- Also: nichts Neues unter der Sonne



Bewertung Java: Kardinalproblem Leistung

- Sprache teilweise überdefiniert
  - Reihenfolge der Operanden- und Parameterberechnung
  - Reihenfolge von Ausnahmen
- Bytecode ist Kellerarchitektur
  - gut interpretierbar
  - schlecht auf reale Maschinen abzubilden
    - Registerzuteilung
    - Superskalarität
    - Codeauswahl
- Nur primitive Skalare als Wertetypen / primitive Typen nicht als Objekttypen
  - Objektidentität ist Overhead (Objekte min. 32 Byte)
  - Objekterzeugung ist teuerste Operation

Einordnung von Java in unser Komponentenmodell

- Sprachtransparenz
  - Jein:
    - Festlegung auf eine Sprache, daher IDL unnötig
    - Einbinden weiterer Sprachen durch Java Native Interface (JNI)
    - Abbilden weiterer Sprachen auf Bytecode
- Plattformtransparenz
  - Ja (durch Java VM)
- Ortstransparenz
  - Teils (Java RMI vorhanden, uniforme Referenz fehlt)
- Reflexion
  - Dynamisch (mit `java.lang.Class`)
- Dienste
  - Namen, Datenbanken, ...
  - typisierte Behälterdatentypen erst ab Version 1.5

- Also: beinahe ein Komponentensystem

JavaBeans

- Erste Komponentenarchitektur für Java (1997)
- Ziel: Visuelle Komposition in Entwicklungsumgebungen (IDE)
  - motiviert durch Borland/Inprise Delphi
- Problem: Schnittstellen benutzerdefinierter Komponenten
  - Java Reflection findet alle öffentlichen Methoden und Felder
  - Viele sind nicht Teil der GUI-Schnittstelle
- Problem: Große Systeme brauchen statische Typprüfung
  - Reflexion $\longrightarrow$ dynamische Typprüfung
- Ansatz:
  - Konventionen für benutzerdefinierte Schnittstellen
  - Metainformationen speichern
  - Codeerzeugung

Bean und BeanInfo

- Bean
  - Enthält Funktionalität
  - Standardisierte Namensgebung
    - set/getXXX für Attribute
    - add/RemoveXXXListener für Ereignisse
  - Unterstützung für Ereignisadapter
- BeanInfo
  - Enthält Metainformation über Attribute, Ereignisse
  - Erlaubt Reflexion über Beans
- Problem: Annotation schwach
  - Nichtfunktionale Eigenschaften nur über Namensgebung
  - Nicht zur Laufzeit erweiterbar

XML Beans

- Lösung: BeanInfo in erweiterbarem Format kodieren
- IBM Bean Markup Language nutzt XML

```
<bean>
  <event> </event>
  <property> </property>
</bean>
```

- Auch Komposition, Referenzen auf Beans
- Serialisierung von Java-Objekten als XML-Daten
- BML Compiler komponiert statisch
- Siehe www.alphaworks.ibm.com

Zusammenfassung: Java Beans

- Sprachtransparenz
  - Nein (daher IDL unnötig)
- Plattformtransparenz
  - Ja (durch Java VM)
- Ortstransparenz
  - Nein (Java RMI nicht genutzt)
- Reflexion
  - dynamisch und statisch
- Dienste
  - Grundlegende GUI-Komponenten
  - Keine Namen, Datenbanken, ...

- Also: eher ein Rückschritt gegenüber Java in unserem Modell
- Heute hauptsächlich als Grundlage von *enterprise beans* und von historischem Interesse

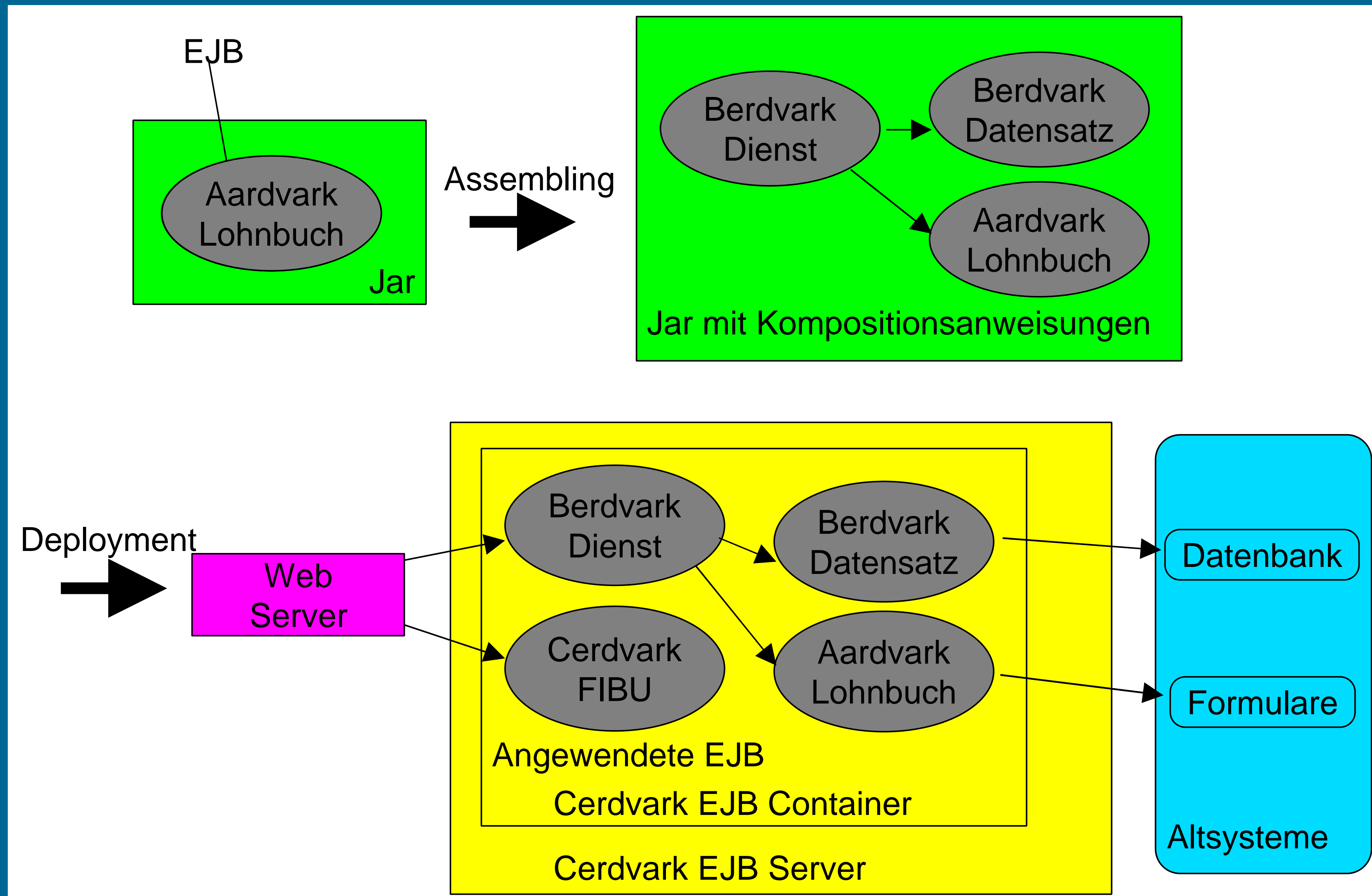
Enterprise JavaBeans (EJBs)

- Ziel: Industrielles Komponentenmodell für Java
  - Sprachspezifisch
  - Plattformtransparent
  - Ortstransparent
  - Dienste: Namen, Persistenz, Transaktionen, Ressourcen, ...
- Weitere Vorgaben
  - Dynamische Dienstanbindung: Komposition ohne Übersetzen
  - Offene Anbindung fremder Komponenten und Werkzeuge
    - Normierte Schnittstellen zu Fremdlösungen (z.B. JNDI, JDBC)
    - Anbindung an CORBA
- Gemeinsamkeiten mit einfachen JavaBeans
  - Name
  - Namensgebung der Schnittstellen

EJBs, Behälter und Deskriptoren

- EJBs sind Komponenten, die ein EJB Interface implementieren. Sie bestehen aus bestimmten Klassen, für
  - Lebenszyklus (Objektdienste)
  - Geschäftsfunktionalität
  - entfernte (RMI) oder lokale Zugriffe (Ortstransparenz)
  - Beschreibung durch *XML Deployment Descriptors*
- Verpackt als EJB JAR
  - Klassenbytecode
  - *Deployment Descriptor*
- Behälter (*Bean Container*) enthalten Komponenten
  - Auffindung (Ortstransparenz)
  - Stellt Ausführungskontext bereit, z.B. *thread*
  - Kontrolle aller Außenkontakte
  - Verwaltet Lebenszyklus, Persistenz von Zuständen (Objektdienste)
  - Sicherheits- und Transaktionsverwaltung
- Server (*Bean Server*) enthalten Container
  - Verwalten und koordinieren Ressourcenzuteilung
  - Zugriff auf Systemdienste

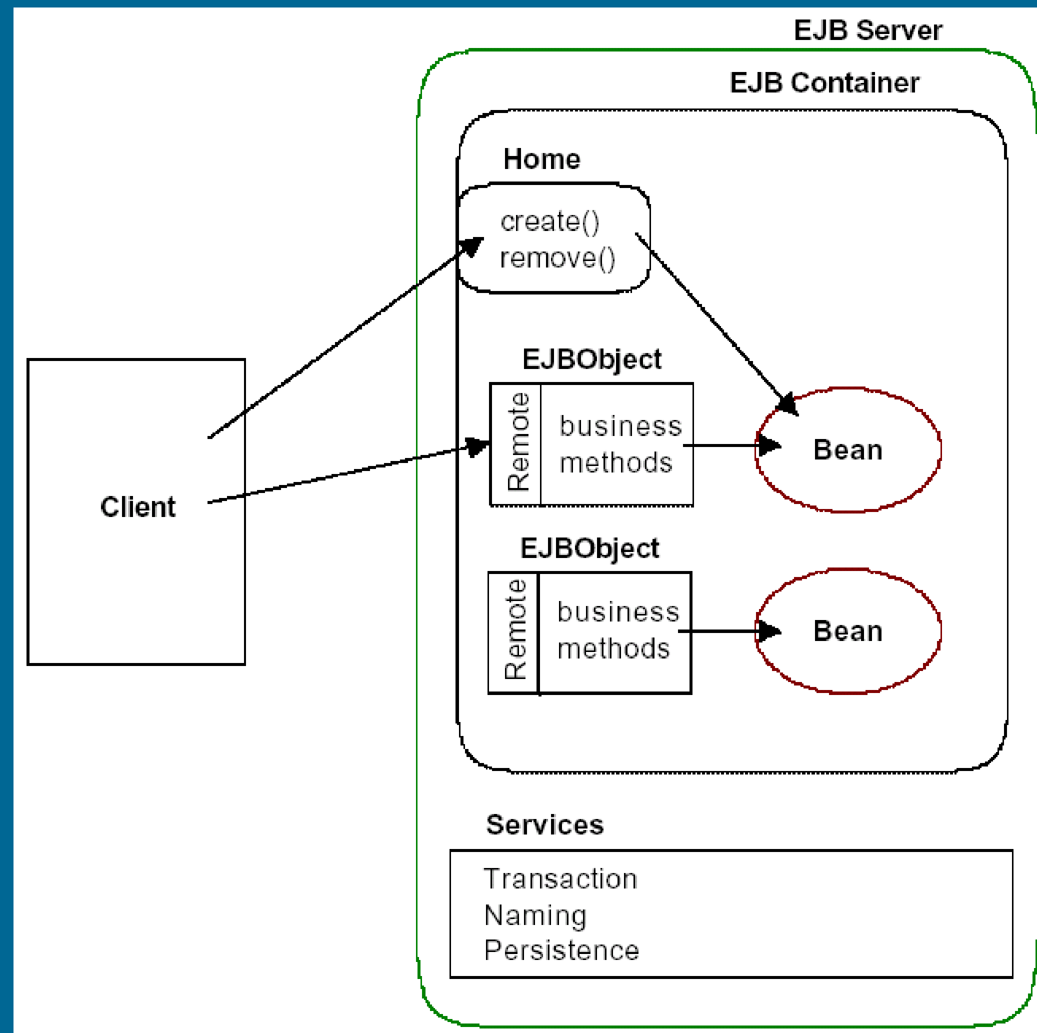
Szenario



Drei Sorten EJBs

- Zustandslose *Session Beans*
  - Träger von Geschäftslogik
  - gebunden an eine Benutzersitzung
  - Klasse, kein Objekt, daher ununterscheidbar
- Zustandsbehaftete *Session Beans*
  - wie oben, nur mit Zustand und damit unterscheidbar
- *Entity Beans*
  - Träger von Geschäftsdaten
  - persistent
  - entsprechen meist einem Datenbankeintrag
  - Persistenz selbst oder durch *Container* verwaltet

EnterpriseBeans aus Kundensicht



Die Kundenschnittstelle (client view)

- Invariant bezüglich Server und Container
- Heimatschnittstelle (erbt von `javax.ejb.EJBHome`)
  - Standardisiert
  - Auffinden von Beans mit JNDI
  - Erzeugen, Löschen
  - evtl. globale Identifikatoren mit `getPrimaryKey()`
- Entfernte Schnittstelle (erbt von `javax.ejb.EJBObject`)
  - Definiert durch Anbieter der EJB
  - Bietet Geschäftsfunktionalität
- Ab EJB 2.0:
  - Lokale Heimatschnittstelle: Finden und Erzeugen von Beans im selben Container.
  - Lokale Version der entfernten Schnittstelle: Funktionsaufrufe statt RMI
  - Behälter (Container) unterstützen auch behälterverwaltete Assoziationen
- Metadaten (erbt von `javax.ejb.Metadata`)
  - zur Reflexion über die Bean

Die Container-Komponenten-Schnittstelle

- Entspricht CORBA BOA
- Schnittstellen für Container und Komponente
  - SessionBean-Schnittstellen `javax.ejb.SessionBean`
  - EntityBean-Schnittstelle `javax.ejb.EntityBean`
  - Container-verwaltete Persistenz
  - Session-Kontext-Schnittstelle
  - JNDI-Namenskontext
  - Transaktionsverwaltung

Deployment descriptor

- EJB Eigenschaften
  - Namen: Name, Klasse, Name der Heimatschnittstelle, Name der Remote-Schnittstelle, Klasse des primären Schlüssels
  - Typ (*session*, *entity*)
  - Zustand
  - Persistenzverwaltung
  - Referenzen auf andere EJB
- Zusatzinformation für Anwendungsersteller
  - Name, Umgebungswerte, Beschreibungsfelder
  - Einbettung in Transaktionen
- Deskriptor ist XML-Dokument zu spezieller DTD

Beispiel: deployment descriptor

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Micro..//EN">
<ejb-jar>
  <description> This is an EJB</description>
  <enterprise-beans>
    <session> <description> Session bean 1 here </description>
      <ejb-name>EmployeeService</ejb-name>
      <home>com-wombat.empl.EmployeeServiceHome</home>
      <remote>com.wombat.empl.EmployeeService</remote>
      <ejb-class>com.wombat.empl.EmployeeServiceBean</ejb-class>
      <ejb-ref><ejb-ref-name>ejb-EmplRecords</ejb-ref-name> ...
      </ejb-ref>
    </session>  ...
    <entity>... </entity> ...
  </enterprise-beans>
  <assembly-descriptor>
    <security-role> ....
    <method-permission> ...
    <container-transaction> ...
  </assembly-descriptor>
<ejb-jar>
```

Anpassungen in der Deployment Phase

- Erzeugung von Klassen
  - Implementierung des *remote interface* (Stub)
  - Lokale Handles und Klebecode (Skeleton)
  - Metadaten
- Generierte Anpassungen
  - Anpassung Bean / Container
    - analog zu CORBA BOA
  - Einweben von Aspekten aus dem *deployment descriptor*
    - Persistenz
    - Transaktionsverwaltung
    - Komposition

Fehlende Anpassungen

- Daten, Funktionen, Kommunikation werden explizit und extern angepaßt
- Synchronisation und Protokollanpassungen entsprechen Anpassungen von Transaktionen und Sessions (siehe vorige Folien)
- Keine Konzepte zur globalen Lebendigkeit
  - Synchronisationskonzepte unter Komposition
  - Lebendigkeitstests

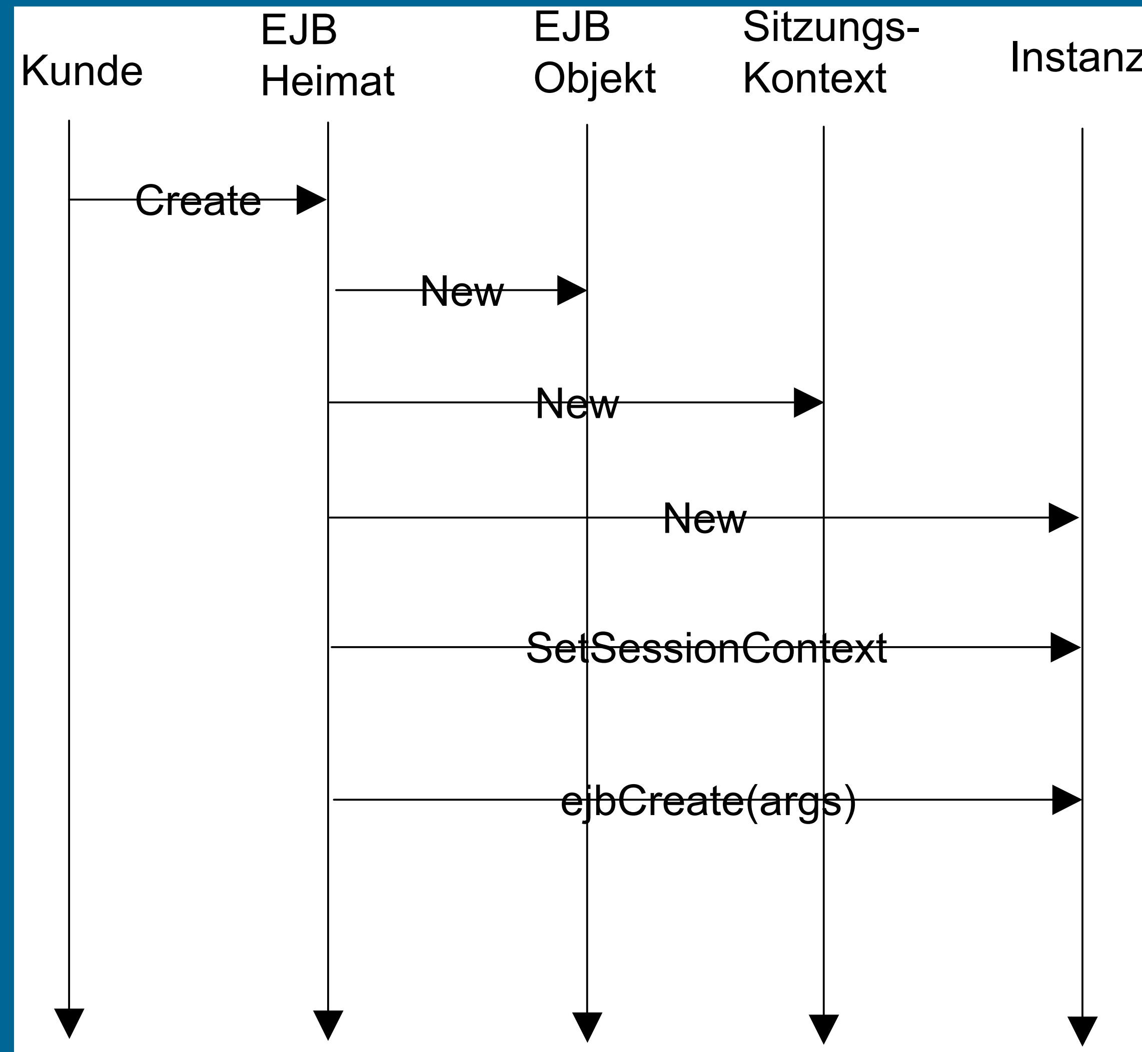
Entwicklerrollen bei EJB

- Komponentenersteller (*bean provider*)
  - Anwendungsexperte
  - Baut EJBs mit fachspezifischen Methoden
- Anwendungsersteller (*application assembler*)
  - Anwendungsexperte
  - Komponiert EJBs zu grösseren EJBs (Anwendungseinheiten)
  - Erweitert die Deployment-Deskriptoren
- Einsetzer (*deployer*)
  - Setzt die EJB in eine Umgebung ein
  - Umgebung = EJB Server und Container
- EJB-Container-Hersteller (*container provider*)
  - Spezialist für Konfiguration, Klebecode, Persistenz, ...
- EJB-Server-Hersteller (*server provider*)
  - Spezialist für Systemdienste
    (oft identisch mit *container provider*)

Sitzungsbohnen (session beans)

- Transient oder halb-transient
- Ausgeführt auf Anfrage eines einzelnen Kunden (möglicherweise in Transaktion)
- Stirbt mit dem Bohnen-Container
- Zustandsbehaftet oder auch nicht
- Passivierung bedeutet
  - Serialisierung aller Objekte, die transitiv von nicht-transienten Attributen aus erreichbar sind
- Aktivierung bedeutet Deserialisierung.
- Session-Kontext hält den jeweiligen Zustand fest

Erzeugen von Sitzungsbohnen



Dauerbohnen (entity beans)

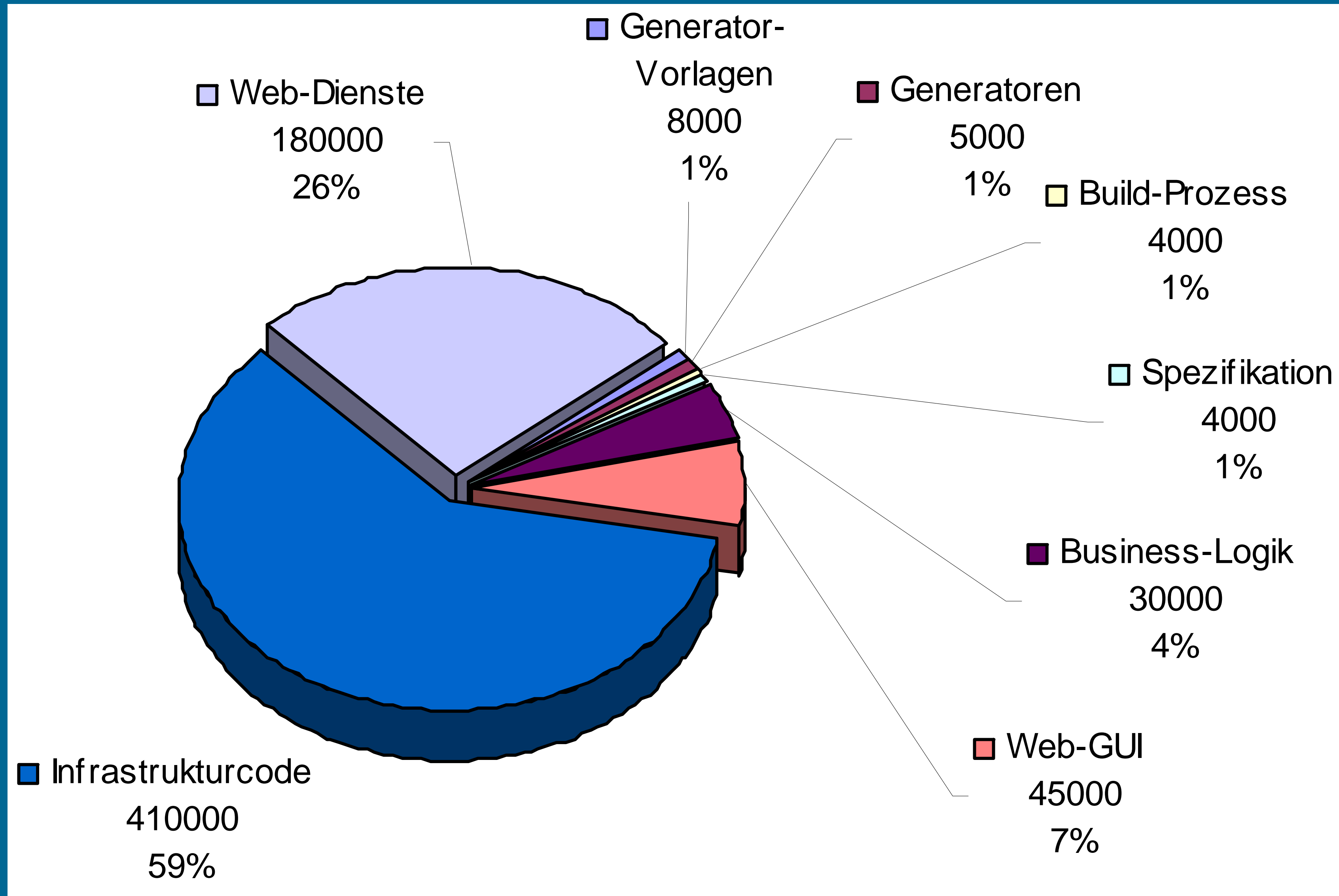
- persistent
- mehrbenutzerfähig, transaktionsorientiert
- globaler Identifikator (analog GUID, IOR)
- ähnlich zu Corba-Objekten (und Monikers unter DCOM)

Szenario

```
Context initialContext = new InitialContext();
CartHome cartHome = (CartHome)javax.rmi.PortableRemoteObject.narrow(
        initialContext.lookup("applications/mail/freds-carts"),
        CartHome.class);
Cart cart = cartHome.create(...);   // EJB spezifisch, vom Bohnenbauer spezifiziert
cart.addItem(66);
cart.addItem(44);

// Speichere die Sitzungsbohne
Handle cartHandle = cart.getHandle();

// ... serialisiere in Datei

Handle cartHandle = .. deserialisiere von Datei
Cart cart = (Cart)javax.rmi.PortableRemoteObject.narrow(
        cartHandle.getEJBObject(), Cart.class);

cart.purchase();
cart.remove();
```

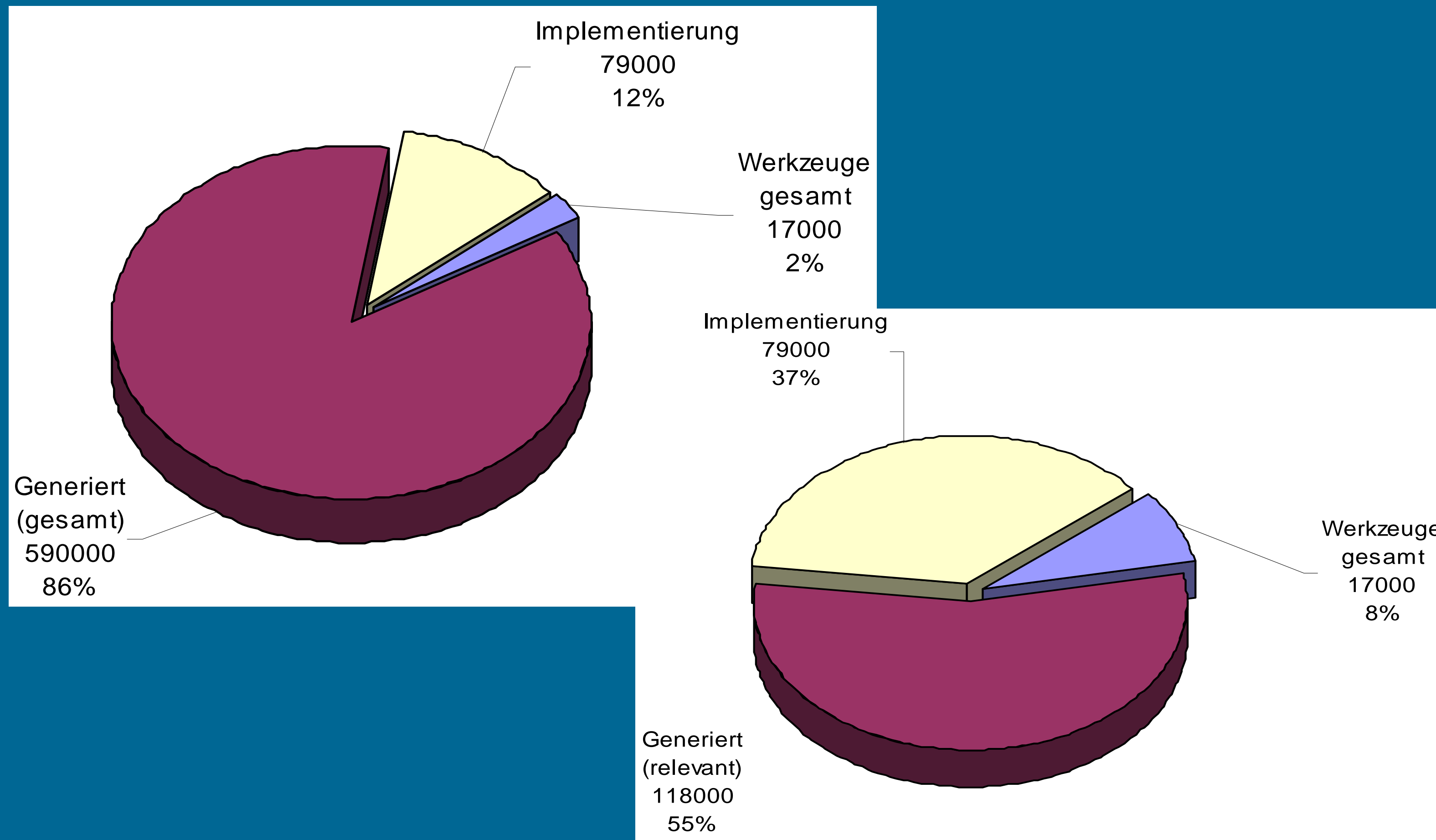
EJB-Fallstudie: Codegröße (1)

- Das Schaubild visualisiert den gesamten Umfang des Fallstudiencodes gemessen in LOC.
- Automatisch erzeugt wurden die beiden größten Blöcke Web-Dienste und Infrastrukturcode (J2EE-Infrastruktur).



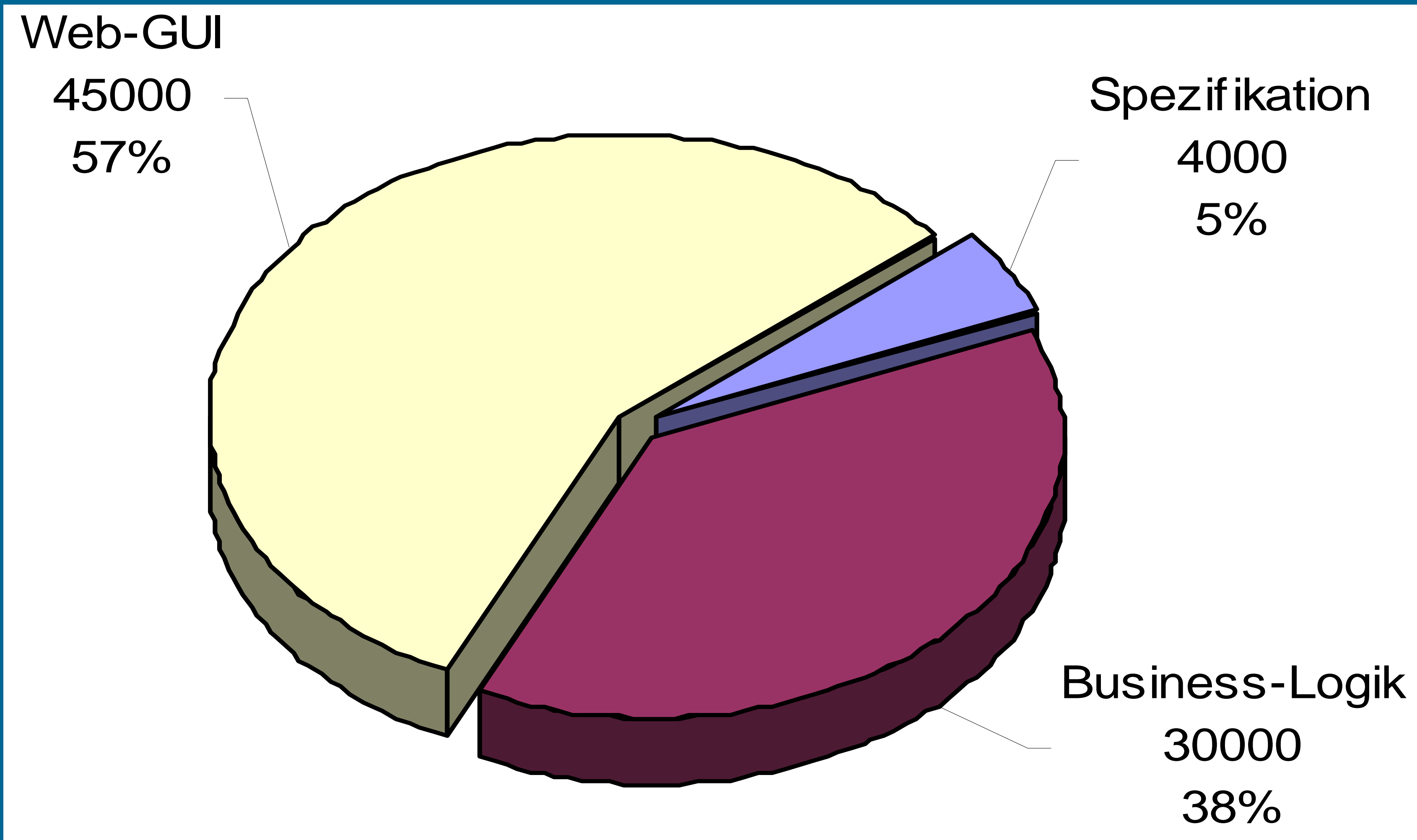
EJB-Fallstudie: Codegröße (2)

- Diese beiden Diagramme fassen die Zahlen zu drei Gruppen zusammen:
  - Werkzeuge
    - Generatoren
    - Generatorvorlagen
    - Build-Prozess
  - Implementierung
    - Spezifikation
    - Buisness-Logik
    - Web-GUI
  - Generierter Code
    - J2EE-Infrastruktur
    - Web-Dienste



- links: entält generierten Code, der nicht benötigt wird
- rechts: bereinigt, nur 20% des generierten Codes wird als relevanter Code angesehen

EJB-Fallstudie: Codegröße (3)



- Aufteilung für die Implementierung:
  - Kernspezifikation (Datenmodell und Schnittstellen) ist sehr kompakt.
  - GUI: über die Hälfte des Codes
- Erkenntnis: Viel Einsparung möglich durch geeignete Spezifikation und Generierung

EJB Zusammenfassung (1)

- Generische Komponenten:
  - *Deployment* entsprechend *Descriptor*
    - Generierung von Stummeln und Skeletten wie aus IDL,
    - Anpassungen an Behälter,
    - Einweben von Persistenz, Transaktion, ...
- Abstrakte Komponenten und Schnittstellen:
  - Generierung bei *Deployment*,
  - Mehrfachvererbung
- Konkrete Komponenten:
  - Mit Transaktionskonzept / Persistenzkonzept
  - Java / Plattformunabhängig
  - Einfachvererbung
- Zusammengesetzte Komponenten:
  - Aufruf über Behälter
  - Java Sprachkonzepte zur Delegation

EJB Zusammenfassung (2)

- Basiskommunikation:
  - Erzeugung und RMI (*Remote Message Invocation*) über Container
  - Anbindung an Corba
  - Zentrale Vermittlung über Behälter
  - Migration von Javacode
- Wiederverwendung:
  - Keine Standardisierungen von Beans
  - AWT und Swing (JavaBeans) meistverwendete Java Bibliothek
  - Industriestandards: SanFrancisco gescheitert, Fiscus-Projekt?
  - Derzeit wird eher die Architektur wiederverwendet, um firmenintern Komponenten wiederzuverwenden
- Generierung aus Spezifikationen:
  - Klare Trennung von Entwicklung und Einpassung (*deployment*)
  - Unterschiedliche Rollen im Entwurf
  - Ausbaufähiges Konzept zur Entwicklung von Komponentensystemen durch Trennung von
    - Entwurfsentscheidungen aus Komponentensicht (Implementierungsdetails)
    - Entwurfsentscheidungen aus Kontextsicht (Transaktion, Persistenz, aber auch verteilte Protokolle zur Diensterfüllung)
  - Schnittstellen-Stellvertreter Generierung

EJB Bewertung (1)

- Die Java-Schnittstellen sind sehr flexibel, funktionieren und können in der Praxis eingesetzt werden
- Standard in Firmenhand Sun Microsystems - kann die gleichen Probleme bereiten wie bei Microsoft
- Java/Beans/EJB wird die Basis für Geschäftsobjekte
- Die Definition von Beans und EJB als Standardkomponenten erhöht den Grad der Modularisierung und Wiederverwendung, bislang aber noch keine Komponenten von der Stange am Markt
- Die Dokumentation ist gut
- *Deployment* in EJB gehen weiter als andere Konzepte, da Anpassungen generiert werden (das kann Corba/DCOM nur für Verteilung)

EJB Bewertung (2)

- Sprachtransparenz
  - Nein (Java)
- Plattformtransparenz
  - Ja (durch Java VM)
- Ortstransparenz
  - Ja (durch *remote interface*, RMI, globalen Identifikator)
- Reflexion
  - Dynamisch und statisch
- Dienste
  - Namen, Transaktionen, Datenbanken, ...

- Industriell sehr bedeutsames Komponentensystem

- Neu oder anders gegenüber CORBA:
  - Deployment Phase
  - Unterscheidung zwischen Sitzungs- und Dauerbohnen
  - Behälter statt ORB. Menge der Behälter kann man als ORB interpretieren.