Komponenten

1. Komponenten

1.1 Was versteht man unter einer Komponente?

Ausgangspunkt:
Dedizierte Aufgabenstellung in einem Unternehmen soll durch Software unterstützt werden

Beispiel:
- **Finanzbuchhaltung**
- **Disagioabgrenzung (Hypothenkenbank)**

Wie kann man diese Aufgabenstellungen softwaretechnisch lösen?

Lösungsansatz 1:

Einsatz von Standardsoftware und anpassen der Parameter

Bsp.: FIBU-System von SAP
Vorteile:
- **Einsatz erprobter Software**
- **Kosten sind präzisierbar**
- **Auf Erfahrungen von anderen Unternehmen kann zurückgegriffen werden**

Nachteile:
- **Software wird auch von anderen Unternehmen eingesetzt, stellt somit keinen Wettbewerbsvorteil dar**

Lösungsansatz 2:

Einsatz von Individualsoftware

Komplette Eigenentwicklung ("from the scratch")

Bsp.: Disagioabgrenzung (Hypothenkenbank)

Vorteile:

- firmeneigenes Know-How kann in Software abgebildet werden
- Option für Wettbewerbsvorteil ist gegeben
- Unterscheidbarkeit zu Wettbewerbern im gleichen Marktumfeld

Nachteil:

- in der Regel entstehen höhere Kosten
- erhöhte Risiken auf funktionaler und finanzieller Seite

Ein Blick in die industrielle Fertigung

Beispiel: Automobilindustrie

- Endprodukt Auto enthält firmeneigenes Know-How
- nicht im Sinne einer Standardentwicklung hergestellt
- keine komplette Eigenentwicklung eines Herstellers
- Vereinfacht kann man sagen:
  Auto wird aus vorgefertigten Komponenten von Zulieferern und selbstentwickelten Komponenten gefertigt
- es handelt sich quasi um eine kombinierte Vorgehensweise
  ein Lösungsweg zwischen
  - Standardentwicklung und
  - reiner Individualentwicklung

  Ziel: schnelle, marktgerechte und kostengünstige Entwicklung

Komponenten

Komponenten in der Technik

- sind vorgefertigte Bauteile mit definierten Schnittstellen, die zu größeren Produkten zusammengesetzt werden

  Beispiele:
  - Maschinenbau
  - Elektrotechnik: Widerstände, IC-Bausteine etc

Unterschiede zur Software

- Technische Komponenten
  - müssen entworfen werden: Plan (Blaupause)
  - müssen produziert werden (physikalische Randbed.)
- Software (Code)
  - Code (Blaupause) ist der einzig zu erstellende Teil
  - Produktion ist einfaches Kopieren und entfällt

Komponenten

Fazit

- **Pläne (Blaupausen) sind mit Risiken behaftet**
- **Vergleich der erfolgreichen Produktion in der Technik und problematischen Entwicklung von Blaupausen in der Informatik (Stichwort: Softwarekrise) ist nicht zulässig**

Individualentwicklung in der Technik (Blaupause und einmalige Fertigung) enthält die gleichen Risiken wie Softwareentwicklung

Komponenten (physikalische zu produzierende Teile) in der Technik und deren Technologien sind nicht mit Komponenten (Blaupausen) in der Informatik vergleichbar.

Komponenten

Beispiel aus der Technik

Ein allgemeiner Hinweis zur Softwarekrise

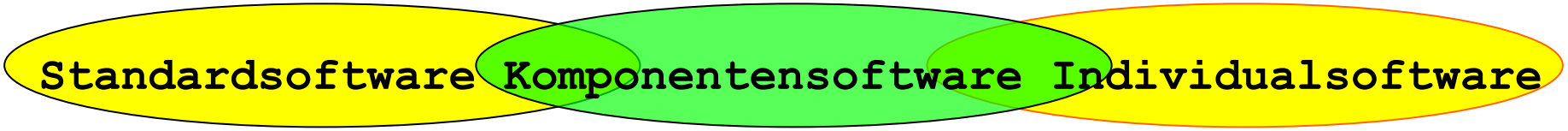
- Begriff Softwarekrise
  - gescheiterte Projekte, ungeeignete Werkzeuge, Explosion der Entwicklungskosten
  - erstmals Mitte der 6o-er erwähnt
- Seitdem gibt es unzählige immer wieder neu vorgetragene Lösungen des Problems
- VORSICHT bei vollmundig angekündigten angeblichen Lösungen

A fool with a tool is still a fool

Komponenten

Ziele des Einsatzes von Komponenten

- Erhöhung der Leistungsfähigkeit durch Verwendung bestehender Teile
  (Hürde bei der Entwicklung von Anwendungen wird gesenkt)
- Erhöhung der Qualität durch Verwendung getesteter Teile
- Senkung der Kosten des Erstellungsprozesses



- sowohl Standardsoftware als auch Individualsoftware können sich in Richtung von Komponenten erweitern

Kernaspekt ist die (Wieder-) Verwendung von vorhandenem Code in einem definierten Umfeld

Komponenten und Software-Architektur

The software architecture of a program or computing system is the structure or the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Len Bass et al.

(siehe: B. Renz Vorlesung Softwarearchitektur)

Komponenten

Elemente der Softwarearchitektur

- Elemente, Komponenten – definieren den Ort von Berechnungen oder auch Speicher, z.B. Filter, Datenbanken, Objekte, Server, Klienten usw.

- Beziehungen, Konnektoren – definieren die Interaktion der Komponenten, z.B. Funktionsaufruf, Pipe, Event usw.

- Eigenschaften – definieren Vorgaben und Beschränkungen für Komponenten, z.B. Schnittstellen, Qualitätsmerkmale usw.

Definition des Begriffs Komponente

Begriff: Komponente

lateinisch componere = zusammensetzen

componendum = „Das Zusammenzusetzende“

Components are for composition

Softwarekomponente

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."
[Szyperski, S.41]

Komponenten

explicit context dependencies only:

- ordnungsgemäße Abarbeitung der Komponente erfordert oft Randbedingungen, die eingehalten werden müssen

  - Bsp.:
    - Übergabeparameter müssen bestimmte Wertebereiche einhalten
    - Datei, die gelesen werden soll muss geöffnet sein
    - Berechnung erfolgt nicht in Echtzeittauglichkeit

- alle Randbedingungen müssen explizit angegeben werden

- durch Überprüfung der Übergabeparameter kann Komponente geeignet auf Verletzung der Randbedingung reagieren und den Benutzer informieren (=> Ausnahmebehandlung)

Komponenten

Was ist für den Einsatz einer Komponente erforderlich?

- **Komponentenmodell**
  Komponente muss einem Komponentenmodell genügen

- **Komponentenplattform**
  Einsatz der Komponente erfordert eine Komponentenplattform, die eine Umgebung zur Verwendung der Komponente darstellt, d. h. die Anforderungen des Komponentenmodells implementiert. Komponenten werden in einem Container verarbeitet.

  Bsp.: Windows-Form ist ein Container für Steuerelemente

Wiederverwendungsformen

- black box
  - Komponente wird als abgeschlossene Einheit verwendet
  - interner Aufbau und Funktionsweise sind unbekannt
  - Komponente kann nicht verändert werden
  - Aufruf erfolgt ausschließlich auf Basis der Schnittstelle und angegebenen Kontextabhängigkeiten

- white box
  - interne Ablauf ist analysierbar
  - Komponente kann verändert werden

- grey box
  - Zwischenstufen zwischen black box und white box

Komponenten

Erfolgreich eingesetzte Komponenten

- prozedurale Bibliotheken
  Beispiel:
  - Numerische Mathematik: Bibliotheken zum Lösen von linearen Gleichungssystemen
- Microsoft's Visual Basic
  Vordefinierte Steuerelemente ermöglichten den schnellen und einfachen Zusammenbau einer ansprechenden und leistungsfähigen Benutzeroberfläche
  >>> Sehr gutes Beispiel wie man sich das Arbeiten mit Komponenten vorstellen kann <<<<

Hinweis: VisualBasic.NET ist heute eine vollwertige objektorientierte Programmiersprache und unterscheidet sich deutlich von früheren Versionen.

Komponenten versus Objekte

Der Begriff Objekt wird in der Informatik extrem vielfältig verwendet

Wir verstehen unter Objekten Instanzen einer Klasse.

Bsp.: class Auto {...}

```
Auto meinAuto = new Auto();
```

Objekte

- sind Instanzen einer Klasse
- haben eine eindeutige Identität
- können beobachtbaren Zustand haben (not stateless)
- kapseln ihren Zustand und ihr Verhalten
- können mit ihrer Umwelt über Interfaces agieren
- sind in der Regel feingranular und haben nur für den Entwickler eine Bedeutung

Komponenten

Komponenten

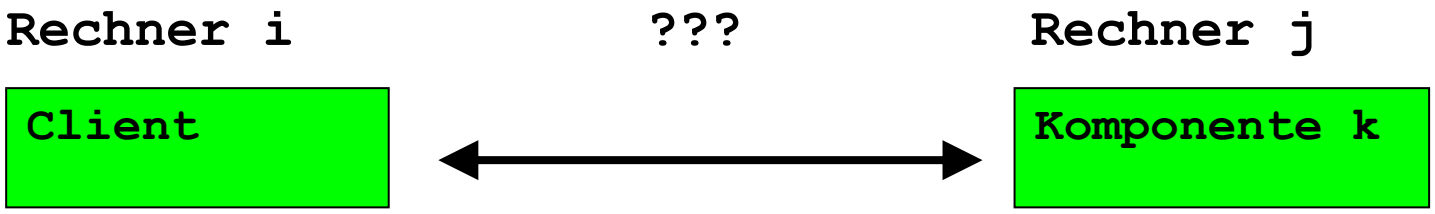
- kapseln ihr Innenleben
- agieren über vertragsmäßige wohldefinierte Interfaces
- können intern objektorientiert implementiert sein, müssen aber nicht
- haben in der Regel für den Fachmann einer Domäne eine Bedeutung
- sind näher an der Anwendungsdomäne als Objekte, die näher am Implementierungsdetail sind
- Entwicklung heute objektorientiert -> Grenzen schwimmend
- sollten keinen Zustand haben (siehe: [Szyperski]) (Aber EJB: Stateful SessionBeans)

Anmerkung: stellt man sich eine Komponente als prozedurale Bibliothek vor, so wird der Unterschied deutlich

1.2 Verwendung von Komponenten
Das Architekturmuster Broker

- Szenario 1: monolithische Anwendung
  - Komponente (Bsp.: Bibliothek mathematischer Funktionen) in Adressraum laden und initialisieren
  - Client kann Komponente direkt verwenden

- Szenario 2: verteilte Anwendung



Komponenten

Offene Fragen

- Wie findet der Client die Komponente?
- Wie wird Interprozess-Kommunikation durchgeführt und wer managed dies?
- Wie programmiert Entwickler auf Clientseite gegen Komponente?
- Wie, wann und von wem wird Komponente auf Serverseite instanziiert?
- Wie werden Komponenten hinzugefügt, entfernt oder ausgewechselt?
- Muss Komponente objektorientiert implementiert sein?
- Können auf Client- und Serverseite unterschiedliche Programmiersprachen verwendet werden?

Ziel: Entwickler auf Clientseite sollte Komponente „wie" lokale Komponente ansprechen können.

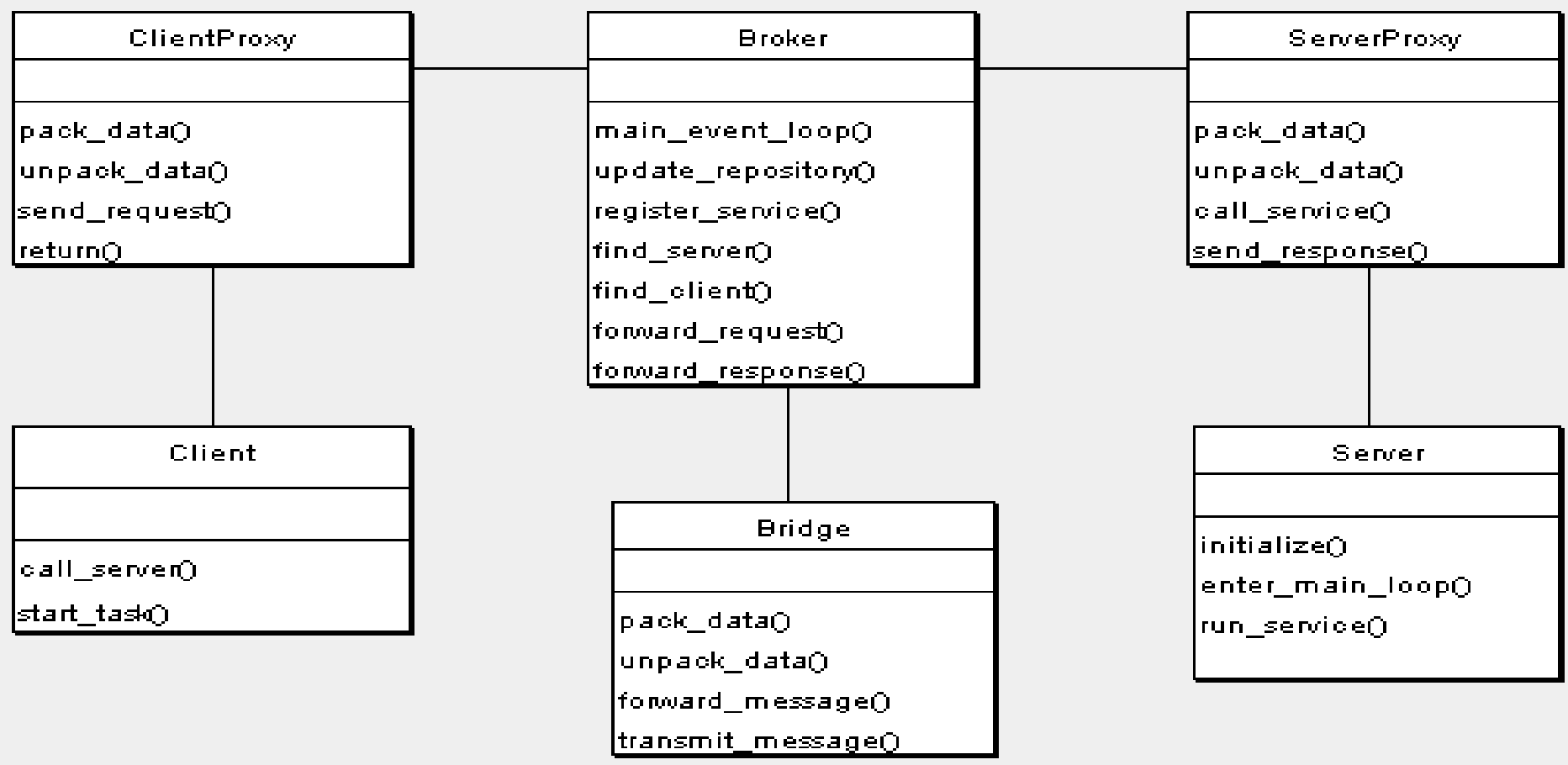
Architekturmuster Broker

Kontext: verteiltes eventuell heterogenes System mit voneinander unahängig miteinander arbeitenden Komponenten

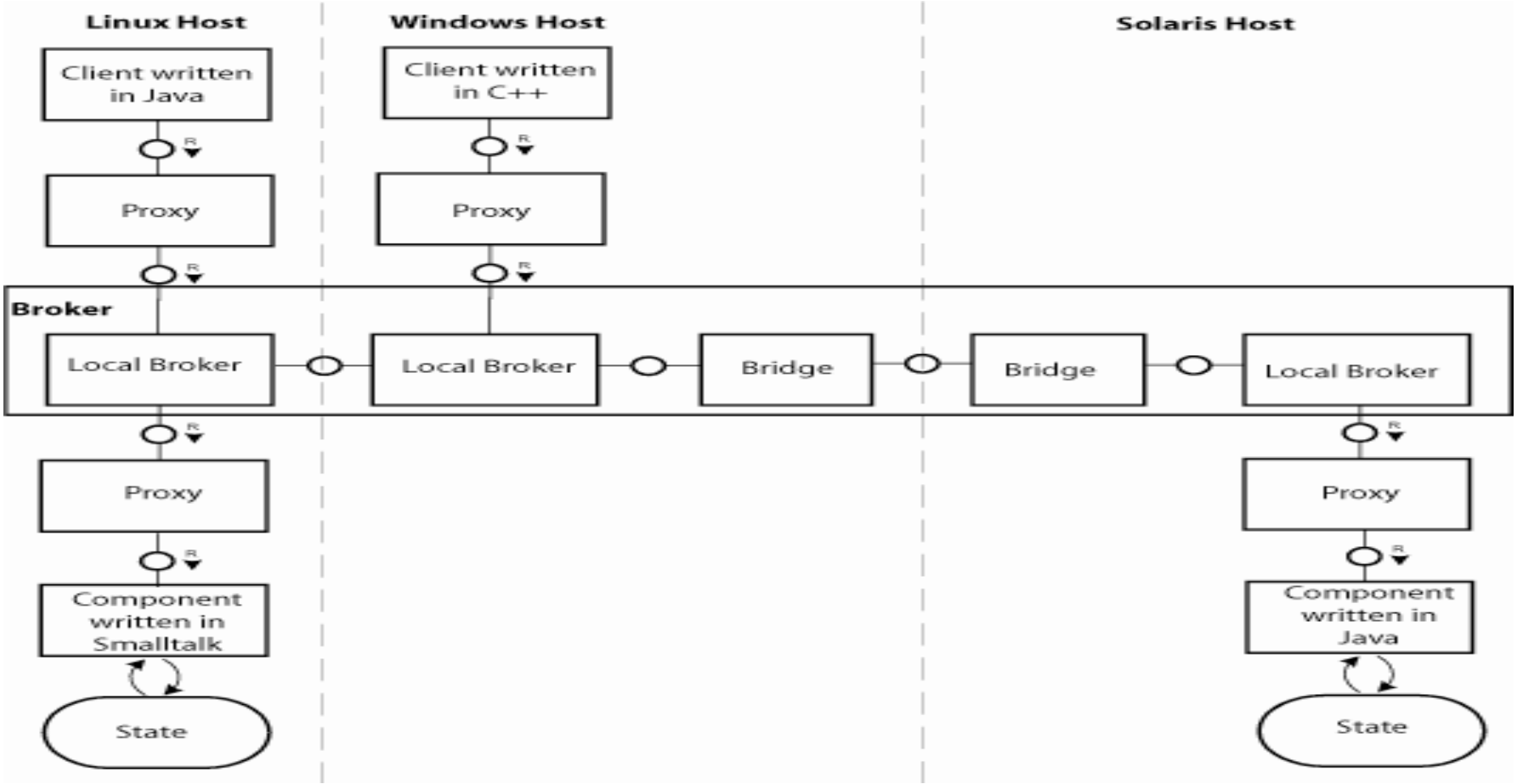
- Vermittlungsinstanz in verteilten Softwaresystemen
  - Server registrieren Dienste beim Vermittler
  - Clients stellen Dienstanforderungen an Vermittler
  - Vermittler leitet Dienstanfoderungen an Server weiter und gibt Ergebnis bzw. Fehlermeldung an Client zurück (indirekte Kommunikation, direkt: Client<->Server)

- Vorteile:
  - Ortstransparenz
  - Sprach-, Plattformunabhängigkeit
  - Entkopplung der verteilten Komponenten

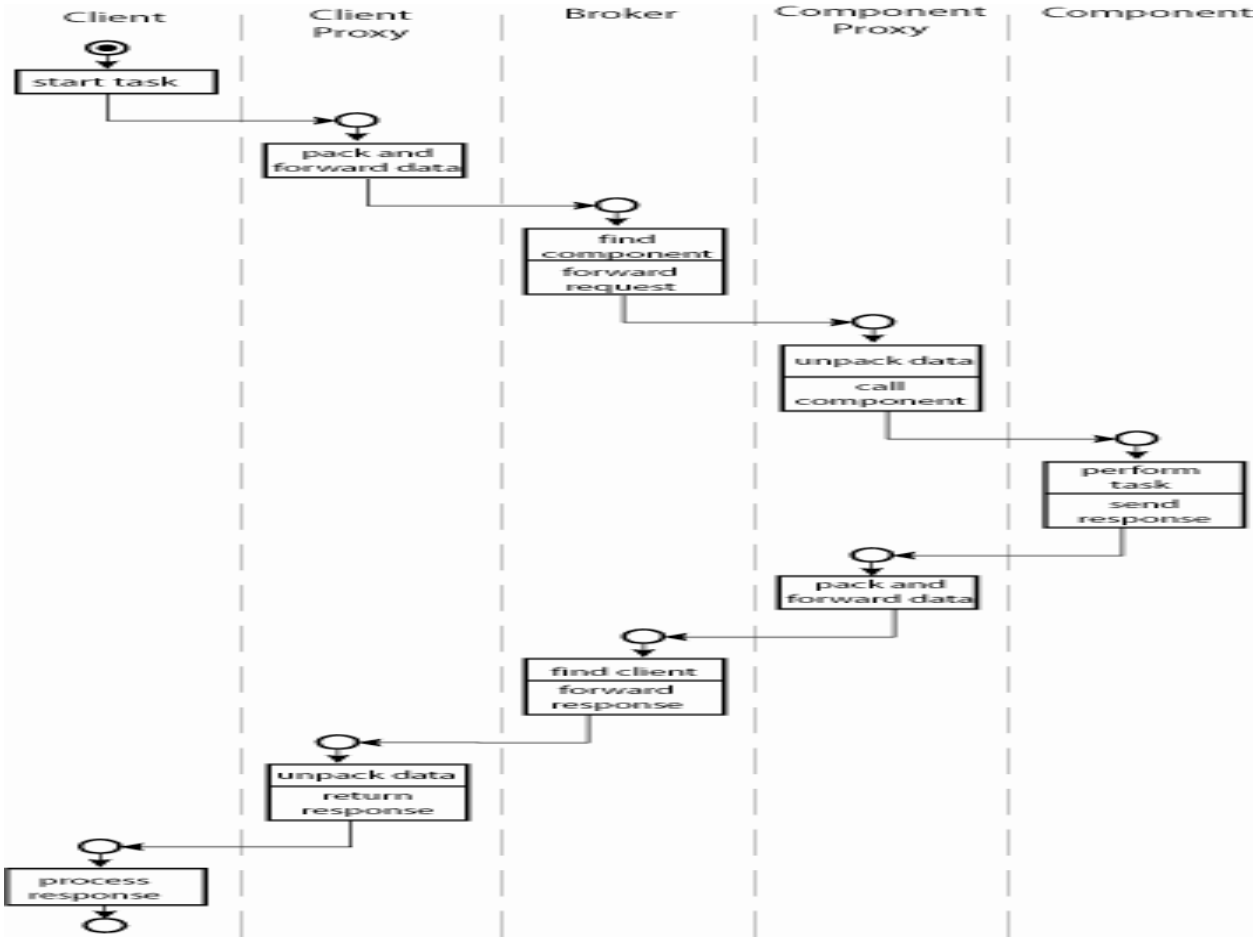
Übersicht: Brokersystem



Broker mit Proxy und Bridge



Dienstanforderung



Komponenten

Schnittstellen

- OMG hat Schnittstellensprache IDL spezifiziert

- Verwendung Interface Definition Language (IDL)
  - Schnittstelle wird mit IDL definiert
  - IDL-Übersetzer generiert Stellvertreter (Clientproxy und Serverproxy) in der jeweils vorliegenden Programmiersprache
  - CORBA verwendet dieses Konzept

Brücke (Entwurfsmuster)

Verdeckt Implementierungsdetails der Kommunikation zwischen 2 Brokern

1.3 Komponententechnologien

- CORBA Komponenten Technologie
  (Common Object Request Broker Architecture)
- Microsoft COM/DCOM
  (Distribiuted Component Object Model)
- Java:
  - Javabeans (im gleichen Prozess)
  - Enterprise Java Beans
- .NET CLR
  Common Language Runtime
  Assembly

Aktuell sind derzeit Java (CORBA integriert, siehe NetBeans), COM und .NET Komponententechnologien

CORBA Komponenten Technologie

- Common Object Request Broker Architecture
- Standardspezifikation der OMG (Object Management Group, 1989 gegründet, www.omg.org)
- Ziel der CORBA Spezifikation: Kommunimation verteilter Objekte über Betriebssystem-, Netzwerk- und Programmiersprachengrenzen hinweg
- wird auch als Middleware bezeichnet
- Mittelpunkt der Architektur: Entwurfsmuster Broker
- existieren viele verschiedene Referenzimplementierungen zur CORBA-Spezifikation
- CORBA definiert ab Version 3.0 Komponententechnologie

Komponenten

CORBA Architektur (OMG)

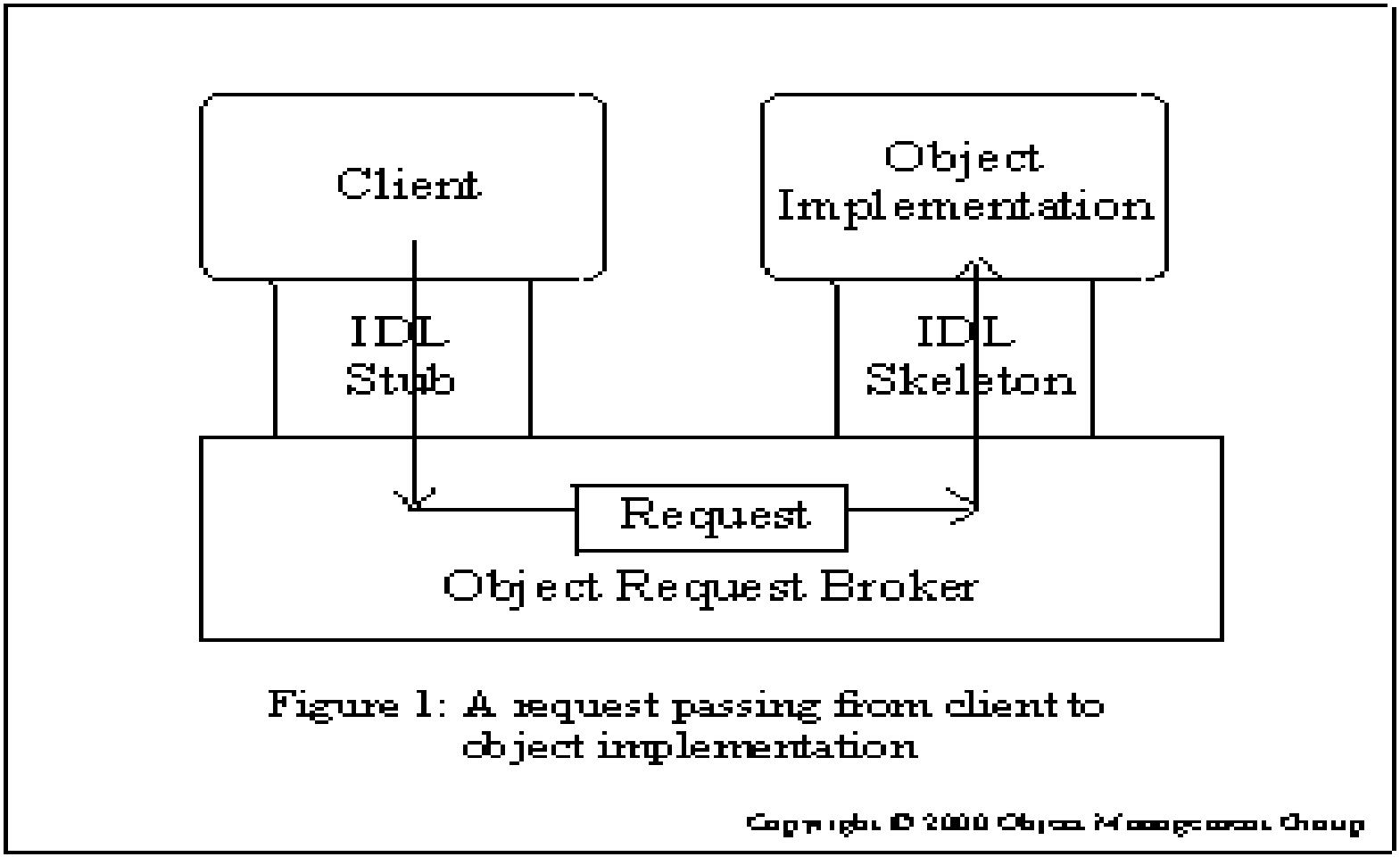


Figure 1: A request passing from client to object implementation

Aufgaben des ORB (Object Request Broker)

- Unterstützung des Klienten beim Aufruf einer Methode durch
  - Suchen des Objektes
  - gegebenenfalls Aktivierung des Objektes
  - Mitteilung der Anforderungen an das Objekt
  - zurückgeben der Objektantwort

- ORB Implementierungen
  Orbix, MICO, IBM, ORBit, VisiBroker, OmniORB, TAO, MiddCor, OpenORB, JacORB

Entwicklung einer CORBA Applikation

1. Definition des Interfaces mit der Corba Interface Definition Language (IDL)
2. IDL-Schnittstellen dem Broker bekanntgeben
3. Automatisches Generieren von Stub und Skeleton mit dem IDL-Compiler in der jeweiligen Programmiersprache
4. Erstellen des Server-Programms in einer beliebigen Programmiersprache, die an das generierte Corba-IDl-Skeleton angeknüpft werden kann
5. Erstellen eines Klientenprogramms in einer beliebigen Programmiersprache, die an das generierte Corba-IDl-Stub angeknüpft werden kann
6. Zugriff vom Klienten über den Broker auf den Server

Komponenten

CORBA IDL

- dient der Definition der Schnittstellen entfernter "Objekte"
- ist an C++ angelehnt

```
interface Artikel{void setPreis(double price);};
```

- IDL-Compiler generiert auf Basis des Interfaces Proxies auf Klienten- und Serverseite (Stub, Skeleton) in der jeweiligen Programmiersprache
- Die OMG hat die Abbildungen von der IDL auf folgende Sprachen standardisiert:
  C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, und IDLscript.
- der Aufrufer auf der Klientenseite und der Aufgerufene auf der Serverseite müssen kein Objekt sein

CORBA Dienste

- **Namensdienst**
  Erlaubt das Finden von Corba-Objekten dem Namen nach

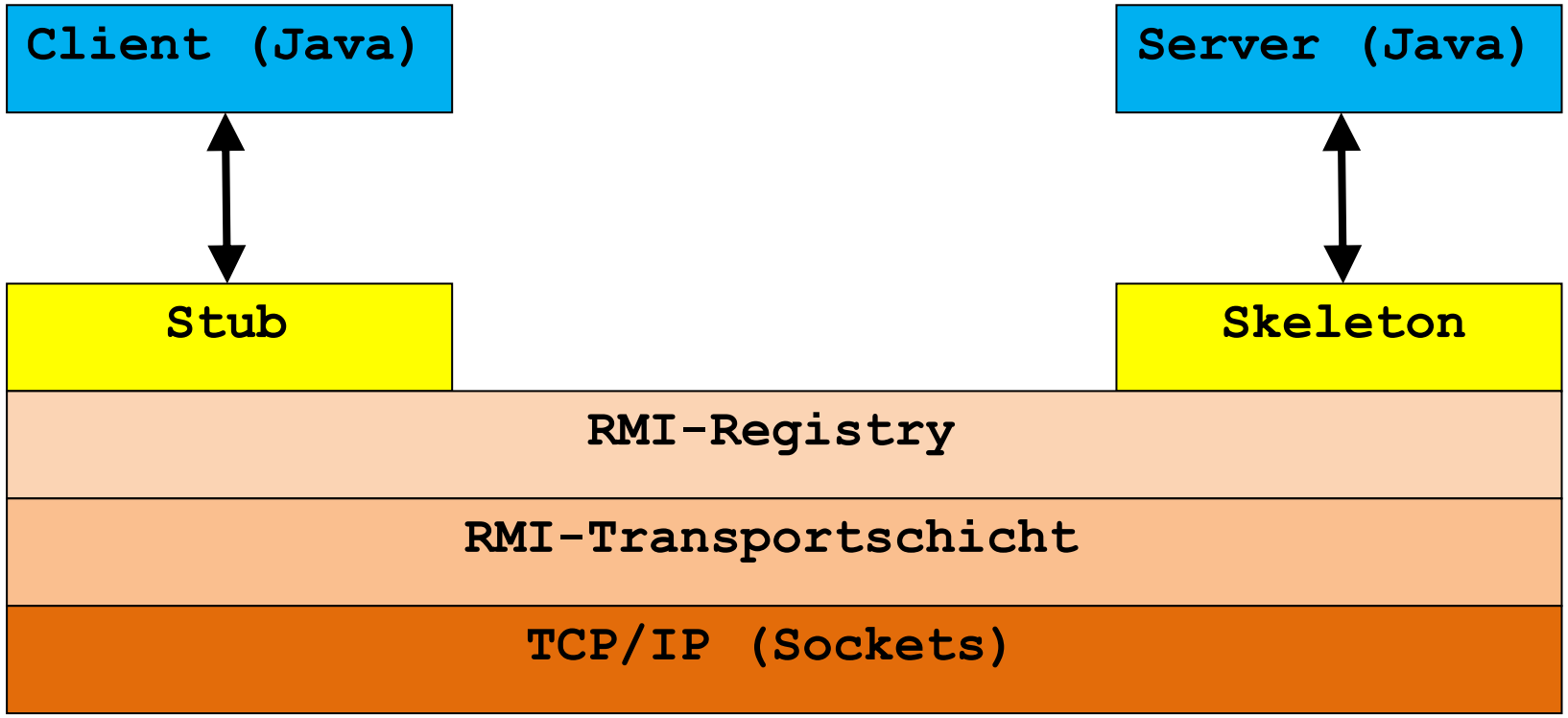
- **Ereignis- und Benachrichtigungsdienst**
  Erlaubt das Senden von Benachrichtigungen an Abonennten

- **Sicherheitsdienst**
  Ermöglicht Zugriffskontrolle und Authentifizierung

- **Handelsdienst**
  Erlaubt das Finden von Corba-Objekten über Attribute

- **Transaktionsdienst**
  Erlaubt Corba-Objekten an Transaktionen teilzunehmen

Corba vs. Java Remote Method Invocation



- RMI ist reines Java
- RMI/IIOP ermöglicht Kommunikation mit anderen Sprachen

Komponenten

COM/DCOM

- COM Component Object Model (lokale Variante)
- DCOM Distributed Component Object Model (verteilte Variante)
- DCOM basiert auf der Standard-Spezifikation des Remote Procedure Calls(RPC) der Open Software Foundation (OSF)
- kein spezifizierter Standard sondern konkrete Implementierung von Microsoft (unterschiedliche Vorgehensweise zur OMG)
- binärer Standard (nicht sprachspezifisch) zur Integration von Komponenten d.h. Programmiersprache an Standard anbinden
- Kommunikation erfolgt über einen als Schnittstelle bezeichneten Mechanismus (Mittelpunkt von COM)
- Komponenten bestehen aus ausführbarem Code

Komponenten

Interface Definition Language , GUID

- Schnittstellen müssen mit IDL deklariert werden
- IDL-Compiler generiert Proxies (Stub, Skeleton)
- COM-Komponente (exe oder dll) enthält COM-Klassen
- COM-Komponenten müssen in Registry registriert werden
- COM-Klasse ist benannte Implementierung einer (mehrerer) Schnittstellen
- Schnittstelle hat IID (Interface ID)(GUID global uniqueID, weltweit eindeutig)
- COM-Klasse erhält eindeutige GUID (CLSID = Class Id)
- COM-Objekt ist Instanz einer COM-Klasse
- COM-Objekt ist transient (explizite Referenzzähler-Verwaltung)
- ein Klient fordert eine Referenz auf eine Schnittstelle mit IID und CLSID an

Schnittstellen

- Schnittstellen sind die fundamentale Einheit in COM/DCOM
- Schnittstelle ist im wesentlichen Tabelle mit Zeigern auf Implementierung der Methoden
- zentrale Schnittstelle: IUnknown, muss von jedem COM-Objekt implementiert werden
- IUnknown enthält die Methoden:
  - QueryInterface: ermöglicht die Anforderungen einer Referenz auf ein Interface über IID
  - AddRef und Release: dienen der Verwaltung von Referenzen (Speicherverwaltung durch Referenzzähler)
- in COM/DCOM-Konzept dürfen Schnittstellen nach ihrer Veröffentlichung niemals geändert werden
- Interfaces basieren auf Extension-Interface-Pattern

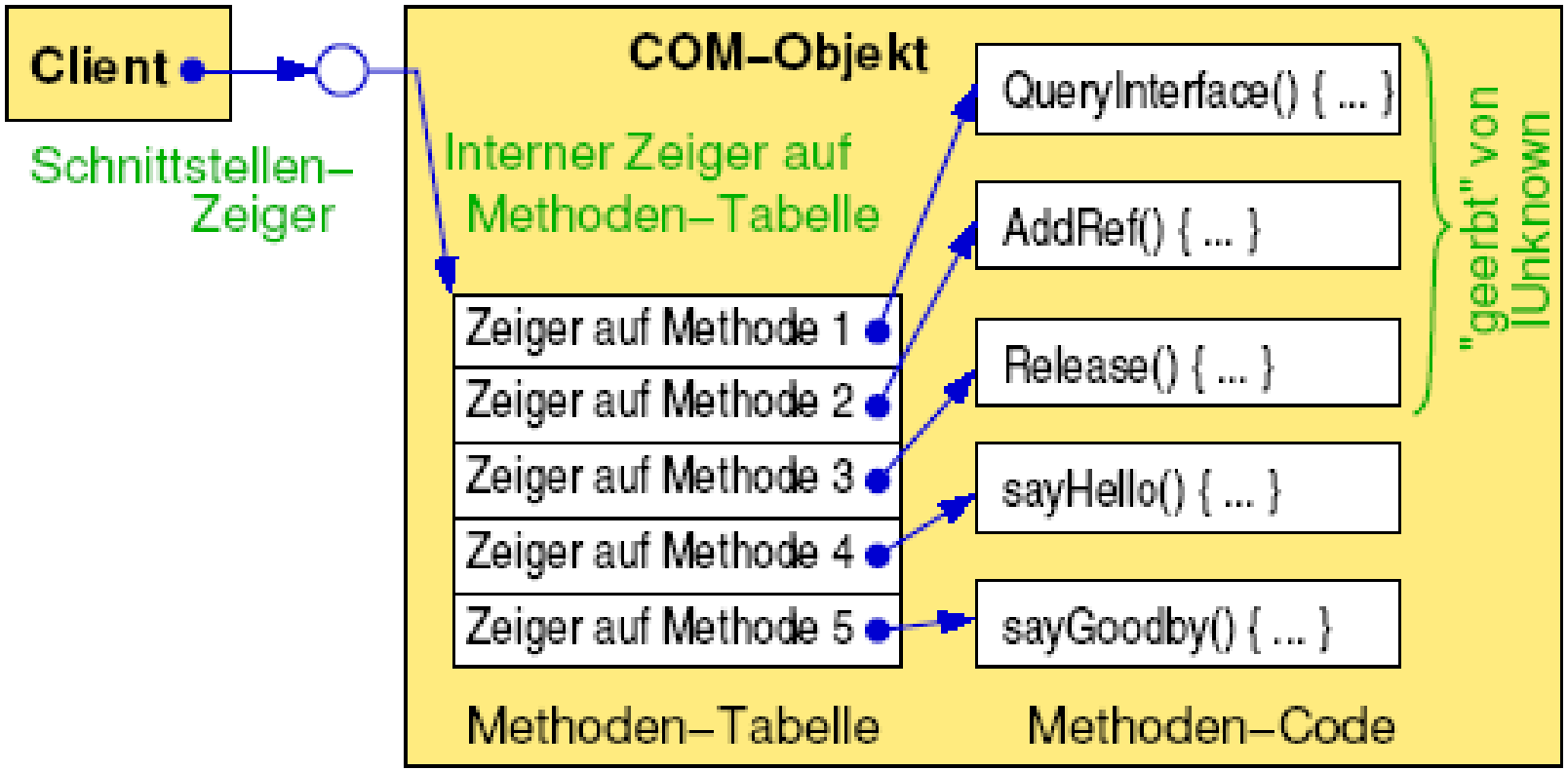
Serverarten

1. **In-Process-Server:**
   Objekte liegen auf demselben Rechner und im selben Adressraum wie der Klient (dll)

2. **Local-Server:**
   Objekte liegen auf demselben Rechner aber in unterschiedlichen Adressräumen (exe)

3. **Remote-Server:**
   Objekte liegen auf unterschiedlichen Rechnern

Schematischer Aufbau einer COM-Schnittstelle



Auszug aus einem COM-Klienten

```
CLSID clsid;
LPCLASSFACTORY pClf;
LPUNKNOWN pUnk;
HRESULT hr;

IRechner* pRech;
if ((hr = ::CLSIDFromProgID(L"ComTest4.Rechner", &clsid)) !=
NOERROR) {return;}
  if ((hr = ::CoGetClassObject(clsid, CLSCTX_INPROC_SERVER,
  NULL, IID_IClassFactory, (void **) &pClf)) != NOERROR)
{return;}
pClf->CreateInstance(NULL, IID_IUnknown, (void**) &pUnk);
pUnk->QueryInterface(IID_IRechner, (void**) &pRech);
pRech->rechne();
pRech->Release();   // Freigabe
```

Verbreitungsgrad und Aktualität von COM

- sind im Microsoft Umfeld weit verbreitet
- Implementierungen liegen sehr oft in C++ vor
  Auch Visual Basic ist weit verbreitet
- Kommunikation zwischen verschiedenen Programmiersprachen ist problematisch

- COM ist ein proprietärer Standard (herstellerspezifisch)

- COM unterstützt keine Versionsverwaltung
  Schlagwort: Dll-Hölle

- .NET bietet neue Komponententechnologie an
  ermöglicht aber die Integration von bestehenden COM-Komponenten über Wrapper

Standards (Informatik) vs. Definition (Mathematik)

- Definition (Mathematik)
  - ist allgemeingültig für jedermann, zeitlich unbegrenzt und nicht änderbar
  - mathematische Sätze basieren auf Definitionen (vorher festlegen)

- Standards (Informatik)
  - sind ein Versuch gemeinsamer Übereinkünfte als Basis einer Kooperation
  - können, müssen aber nicht akzeptiert werden und sind zeitlich änderbar

Vorgehensvarianten zur Festlegung von Standards

1. Zuerst Standard spezifizieren und anschliessend Implementierungen erstellen (Bsp.: OMG)
   Problem:
   ungenügende oder unvollständige Spezifikation, Hardwarerandbedingungen

2. Problemstellung durch konkrete Implementierung lösen und anschliessend standardisieren (Bsp.: Microsoft)
   Problem:
   Standard wird von anderen Firmen nicht aktzeptiert

Komponenten in Java

- Javabeans
  - im selben Prozess
- EJB: Enterprise Java Beans
  - Prozess- und Rechner-übergreifend
    - sind einfache Java Klassen (erst ab EJB 3.0!)
      (POJO: Plain Old Java Objects)
    Anm.: Javabeans und EJB sind grundverschieden!

- Erstellung einer EJB Applikation
  1. Installation einer JEE Plattform
     (z. B.: Applikationsserver JBOSS verwenden)
  2. Implementierung einer EJB
  3. Installation der EJB auf JEE-Plattform (Deployment)
  4. Implementierung des Clients

Hello World EJB Beispiel

```
// Implementierung einer HelloWorld EJB

@javax.ejb.Remote  // Annotation == .NET Attribut
public interface HelloWorld {public String hello(String
who);}

@javax.ejb.Stateless
public class HelloWorldImpl implements HelloWorld {
    public String hello(String who) {
      System.out.println("Say hello to " + who);
        return "Hello " + who;}
}
```

Komponenten

```
// Implementierung des Client
public class HelloWorldClient {
   public static void main(String[] args) throws
       Exception {
    InitialContext context =new InitialContext();
    HelloWorld hello =
    (HelloWorld)context.lookup("HelloWorldImpl/remote");
    System.out.println(hello.hello("World"));   }
}


// siehe z. B.: http://docs.jboss.org/ejb3/app-
server/tutorial/stateless/stateless.html
```