Komponentenbasierte Software-Entwicklung (CBSE)

Vorlesungsskript

FIN, IVS, AG Softwaretechnik

Weitere Links zum CBSE:

- eine Foliensammlung zum CBSE aus Australien
- ein sogenannter Komponentenmarktplatz der Uni Chemnitz
- ein spezieller Service der Flashline zur Komponentenbestellung
- die Home Page eines CBSE-Buches von Heinmann und Councill
- eine Community-Seite zum CBD
- eine allgemeine Bibliography zum CBSE
- CBSE-Informationen der Waseda-Uni in Japan
- auch mal eine Konferenzinformation zum CBSE
- schließlich noch eine Link-Liste zum CBSE

Komponentenbasierte Software-Entwicklung (CBSE)

1. Lehrhilfe: Software-Komponentenbegriffe

FIN, IVS, AG Softwaretechnik

1. Komponenten und CBSE

1.1 Ansätze für ein CBSE:

CBSE als ***Component-Based Software Engineering*** beschäftigt sich mit dem Entwicklung, dem Einsatz und der Akquisition von Software-Komponenten zum Zwecke der flexibleren und kostengünstigeren Systementwicklung in allen Bereichen des Software Engineering. Ansätze dafür waren/sind beispielsweise:

- die Anwendung bzw. Einbettung von bereits existierenden Funktions- oder ***Modulbibliotheken*** (beim OOSE als Klassenbibliotheken),
- der Einbau von ***Standard-Software*** in ein Anwendungssystem, wie zum Beispiel als statistische Auswertungskomponente zu einem speziellen Datenaufnahme- und -speicherungssystem,
- die Erweiterung des sogenannten *Programmierens im Großen (programming in the large)* zu einer Art ***Megaprogrammierung***, die auf die Verwendung spezialisierter Komponentenbereiche für die Systemprogrammierung und An-wendungsprogrammierung orientiert.

Das heutige Gebiet des CBSE basiert vor allem auf den Forschungsarbeiten zur ***Software- Systemarchitektur*** mit den dabei notwendigen Standardisierungsaufgaben für eine einheitliche Definition der Komponentenschnittstellen bis hin zu Kriterien für die Bestimmung der Qualität der verwendeten (externen) Komponenten (siehe auch Carnegie Mellon University).

1.2 Der Komponentenbegriff

In der Softwaretechnik-Einführung haben wir bereits den Begriff der Software-Komponente wie folgt definiert:

> *Die Software-Ressourcen für das Software-Produkt sind alle Programm- und Dokumentationsbestandtteile, die bereits vorhanden sind und durch das Beschaffen (Akquirieren), Bereitstellen, Hinzufügen oder Anpassen in das künftige Produkt mit eingehen.*

Wir wollen diese Begriffsdefinition untersetzen und führen zunächst weitere Definitionen aus [Brown 98] an.

- *Komponenten* sind nichttriviale, nahezu unabhängige und leicht ersetzbare Teile eines Systems, die eine eindeutig festgelegte Funktionalität im Rahmen eines Kontextes einer Systemarchitektur besitzen.
- Eine *Laufzeit-Komponente* ist ein dynamisch einbindbares Paket eines oder mehrerer Programme, das als Einheit verwaltet und über eine klar definierte Schnittstelle eine Funktionalität realisiert.
- Eine *Software-Komponente* ist eine Kompositionseinheit mit vereinbartem Interface und expliziten Kontextabhängigkeiten.
- Eine *Business-Komponente* stellt eine Implementation eines autonomen Business-Konzeptes dar und kann dabei verteilt sein.

Nach Orfali ist eine Komponente wie folgt definiert:

> *A component is a piece of software small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability.*

Eine Komponente kann also einerseits verschiedene Ebenen eines Anwendungsbereiches betreffen und zum anderen durch unterschiedliche technologische Sichten geprägt sein. Hinsichtlich der Komponentenarten unterscheidet [Brown 98] die folgenden vier Grundformen:

- ***COTS-Komponenten*** *(commercial off-the-shelf components):* Dazu zählen alle kommerziell entwickelten Systeme und Systemteile, die in die Architektur eingebunden bzw. integriert werden können und ausschließlich über ihre Schnittstellen im Sinne von Anschluss- oder allgemeinen Nutzungsbedingungen definiert sind.

- ***Qualifizierte Komponenten*** *(qualified components):* Hierbei sind explizite Schnittstellen definiert, die eine entsprechende Varianz für die Anpassung an vorhandene Architekturbedingungen ermöglichen.

- ***Angepaßte Komponenten*** *(adapted components):* Diese Komponenten besitzen bereits Techniken, um ggf. auftretende Schnittstellenkonflikte, beispielsweise im Sinne der sogenannten Wrapper-Technik, zu lösen.

- ***Verbundene Komponenten*** *(assembled components):* Diese Form ist bereits in eine Infrastruktur eingebunden, die selbst die Komposition und Integration von Komponenten unterstützt.

- ***Aktualisierte Komponenten*** *(updated components):* Die Komponenten dieser Art sind bereits erneuert bzw. die sie nutzende Architektur gewährleistet eine problemadäquate Ersetzung derartiger Komponenten.

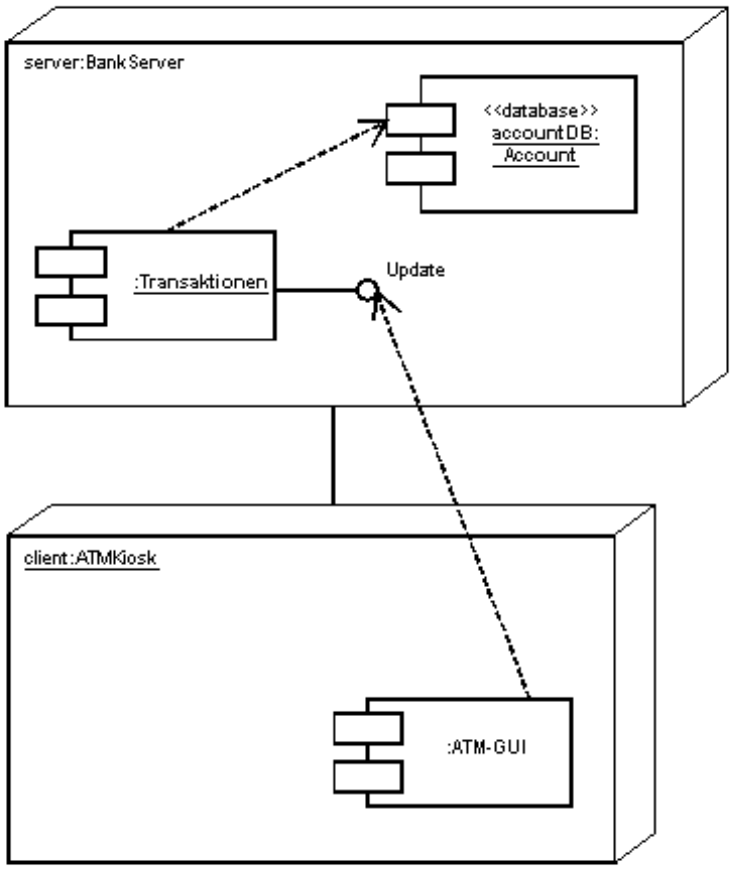
Unter den Bedingungen eines modernen Einsatzes von Komponenten in verteilten Systemen sollten Softwarekomponenten folgenden **Anforderungen** genügen:

- Unabhängigkeit von der Umgebung,
- Ortstransparenz,
- Trennung von Schnittstelle und Implementierung,
- Selbstbeschreibende Schnittstellen,
- Problemlose sofortige Nutzbarkeit (Plug & Play),
- Integrations- und Kompositionsfähigkeit.

1.3 Komponenten vs. Objekte

In der heutigen Welt der Dominanz objektorientierter Programmierung werden ***Objekte*** hergeleitet und im Falle ihrer relativen Gleichartigkeit zu einem ***Klassentyp*** zusammen gefasst. Die Entwicklungsaufgabe besteht also in der "Transformation" einer Problemstellung in ein Objektmodell für dessen Implementation als Softwaresystem.

Beim CBSE geht es bei der Komponentendefinition vornehmlich um einen ***technologischen Aspekt*** einer Systemein- oder -aufteilung. Eine Komponente kann beim OOSE durchaus aus mehreren Objekten bestehen.



Dieses Verteilungsdiagramm zeigt uns gleichzeitig noch eine hardware-mäßige Aufteilung unserer beiden Komponenten an.

1.4 Komponentenbezogene Technologien

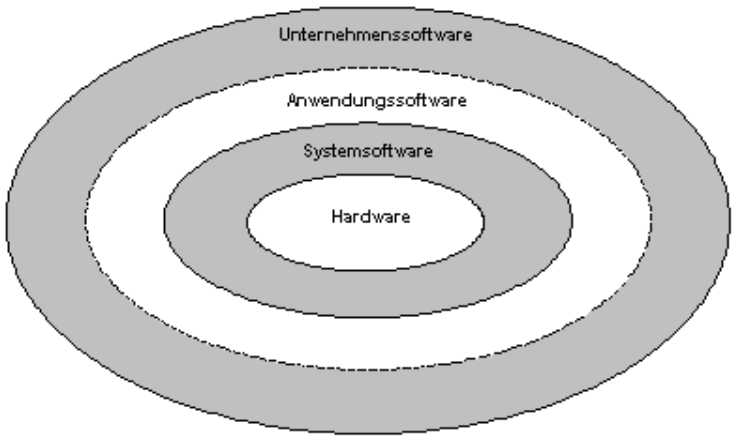
Im folgenden wollen wir einige Technologien auflisten, die sich mit der Entwicklung von komponentenbasierten Software-Systemen bzw. mit der zweckmäßigen Konstruktion beschäftigen. Im Rahmen dieser Lehrveranstaltung werden wir auf diese Technologien mehr oder weniger umfassend eingehen.

- die Konzipierung und Entwicklung von Software-Komponenten für deren späteren Einsatz im Rahmen des CBSE als sogenannte ***Komponenten-Frameworks***,
- die ***Wiederverwendung (Reuse)*** von Software-Komponenten,
- die ***Komponentenauswahl*** auf der Grundlage von Eignungs- bzw. Qualitätskriterien,
- ***Komponentenbasierter Systembildung aus COTS*** unter Berücksichtigung der Plattform- und Funktionsbedingungen,
- die Portierung alter Softwarekomponenten mittels der sogenannten ***Wrapper-Technik***,
- Behandlung einer ***Beispieltechnologie***,
- komplexe Systemintegration als ***Enterprise Application Integration (EAI)***.

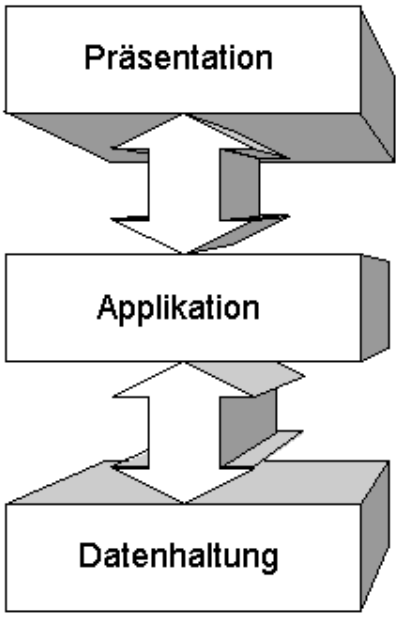
Als Beispieltechnologie wählen wir die Enterprise JavaBeans aus, um eine relativ weit verbreitete Form komplexer verteilte Systemrealisierung und -implementation kennen zu lernen.

1.5 Komponentenarchitekturen

Komponentenarchitekturen können nach verschiedenen Aspekten dargestellt werden, bei denen die Symbolik für die Komponenten und die Verbindungen zum einen eine ***funktionelle Unterteilung*** und zum anderen eine ***technologische Abgrenzung*** impliziert. Eine derartig "reine" technologische Form zeigt uns die folgende Abbildung.

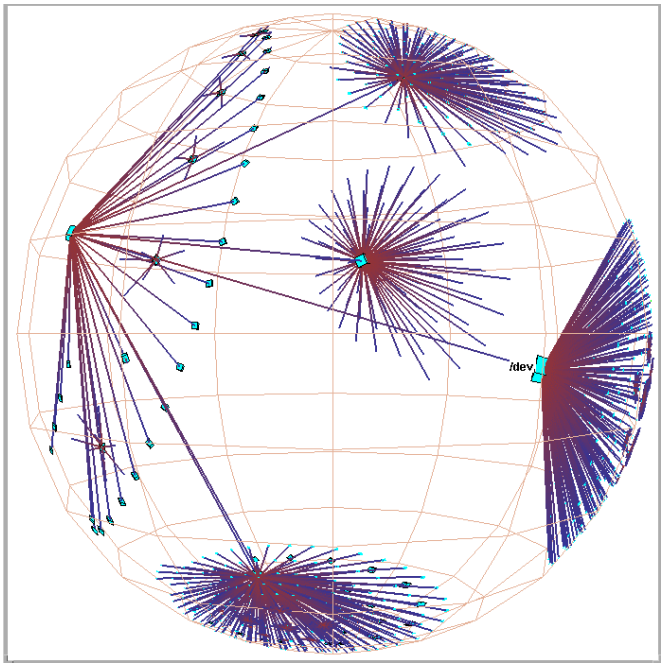


Neben dieser Schalendarstellung können die Schichten (z.B. als sogenannte ***Tier***) auch untereinander angegeben werden, wie in der folgenden *3-tier-Architektur*.



Komponenten können dabei Teil der jeweiligen Schicht sein, die Schicht selbst darstellen oder auch die Architekturform insgesamt als Merkmal besitzen.

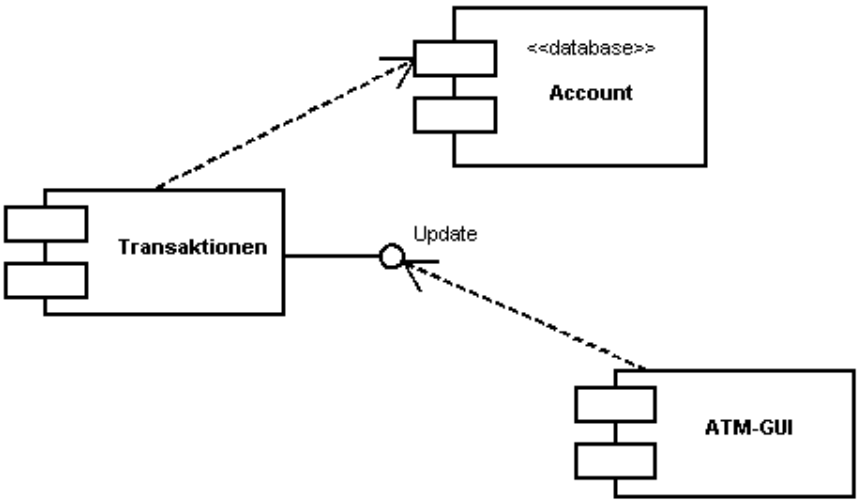
Schließlich können auch spezielle Modelle, wie zum Beispiel 3-D-Formen zur Visualisierung (verteilter) Architekturen verwendet werden. Das folgende Beispiel visualisiert eine (HTML-) Dokumentenstruktur bzw. -architektur bei der die einzelnen Komponenten Dokumente darstellen.



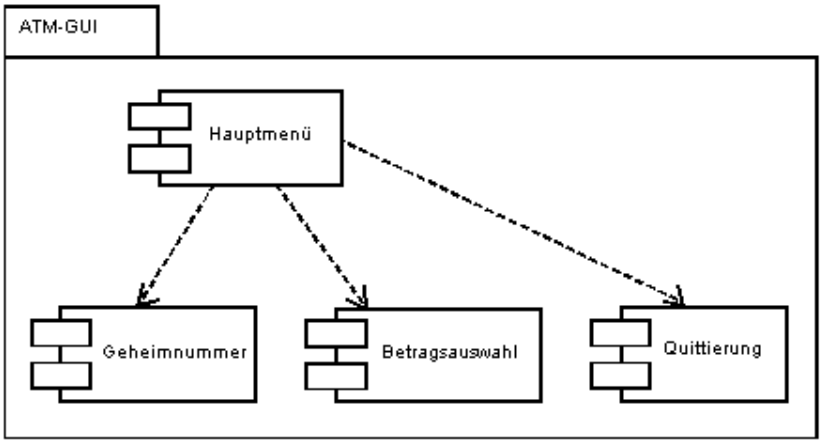
Für die Komponentendarstellung steht uns zum einen das ***Komponentendiagramm*** der UML zur Verfügung. Es hat sich aus dem Booch-Klassensymbol entwickelt, wobei die für die Entwicklung wichtigen Symbolkennzeichnungen, wie

- die einfache Linienzeichnung als Symbolik für eine ***spezifizierte Klasse*** bzw. Komponente,
- die gestrichelte Linienzeichnung als Symbolisierung einer ***generischen Komponente***,
- die (schwarz) ausgefüllte Symboldarstellung als Kennzeichnung einer ***implementierten Klasse*** oder Komponente.

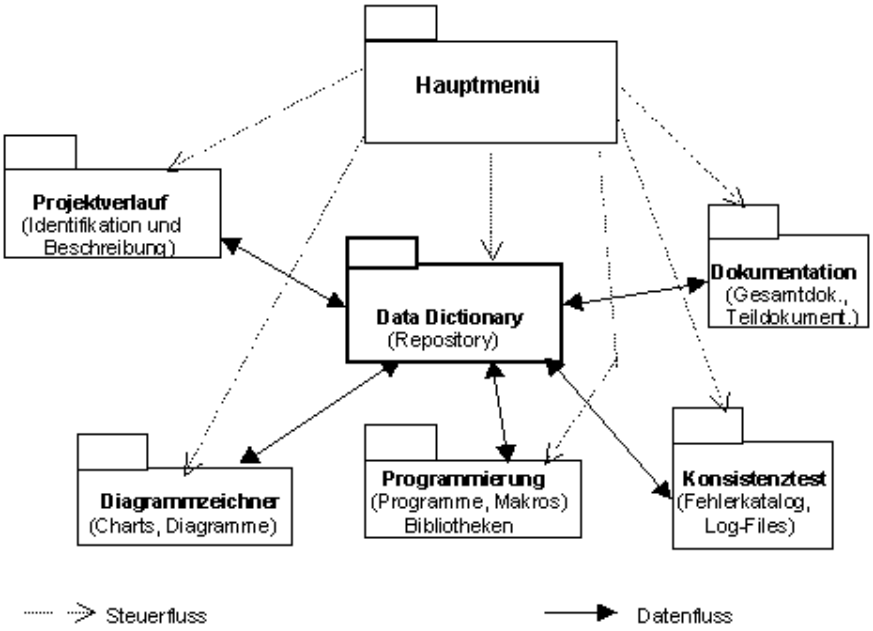
verloren gegangen sind. Ein Beispiel eines Komponentendiagramms ist in der folgenden Abbildung anegeben. Man beachte, dass die Komponenten hierbei auch aus mehreren Klassen bestehen kann.



Eine weitere *Kapselung* von Komponenten haben wir in UML mit dem sogenannten ***Package-Symbol*** kennen gelernt. Die folgende Abbildung zeigt eine derartige Darstellung.

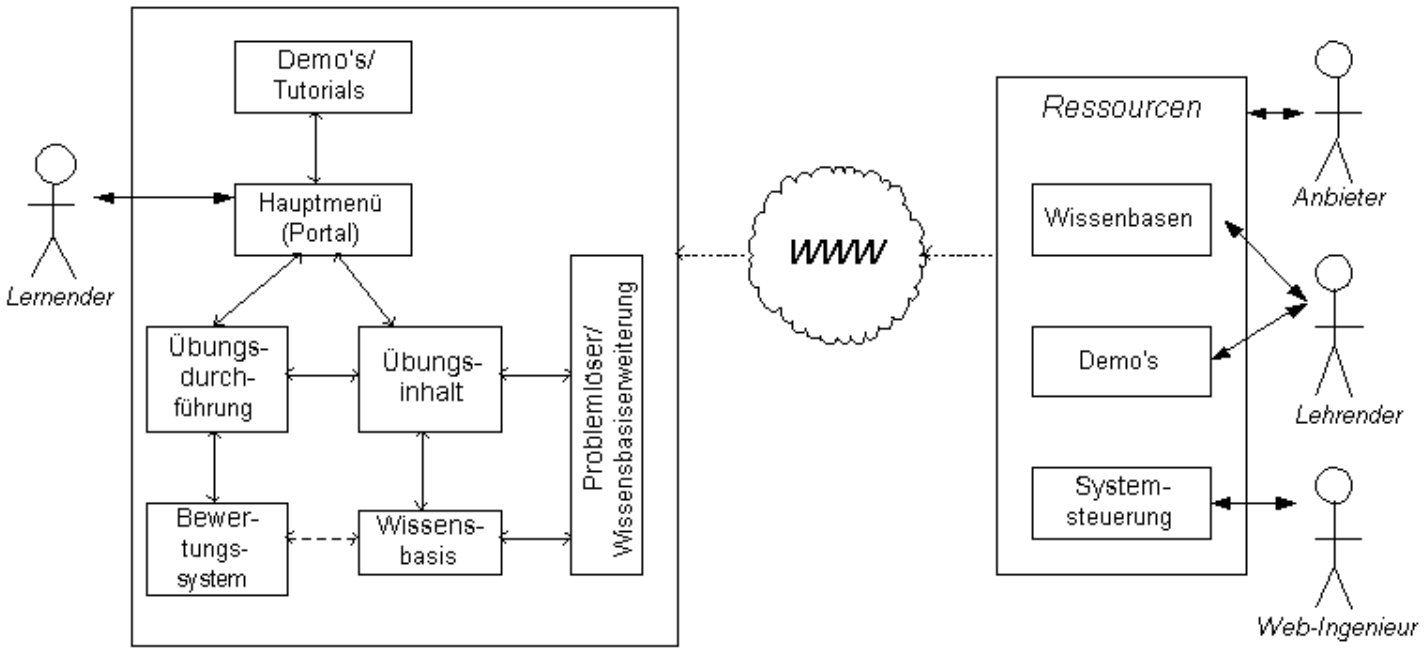


Dabei enthält ein Paket mehrere Komponenten. Eine aus mehrerer Paketen bestehende Komponentendarstellung zeigt uns die folgende Abbildung einer allgemeinen ***Architektur von CASE-Tools***.

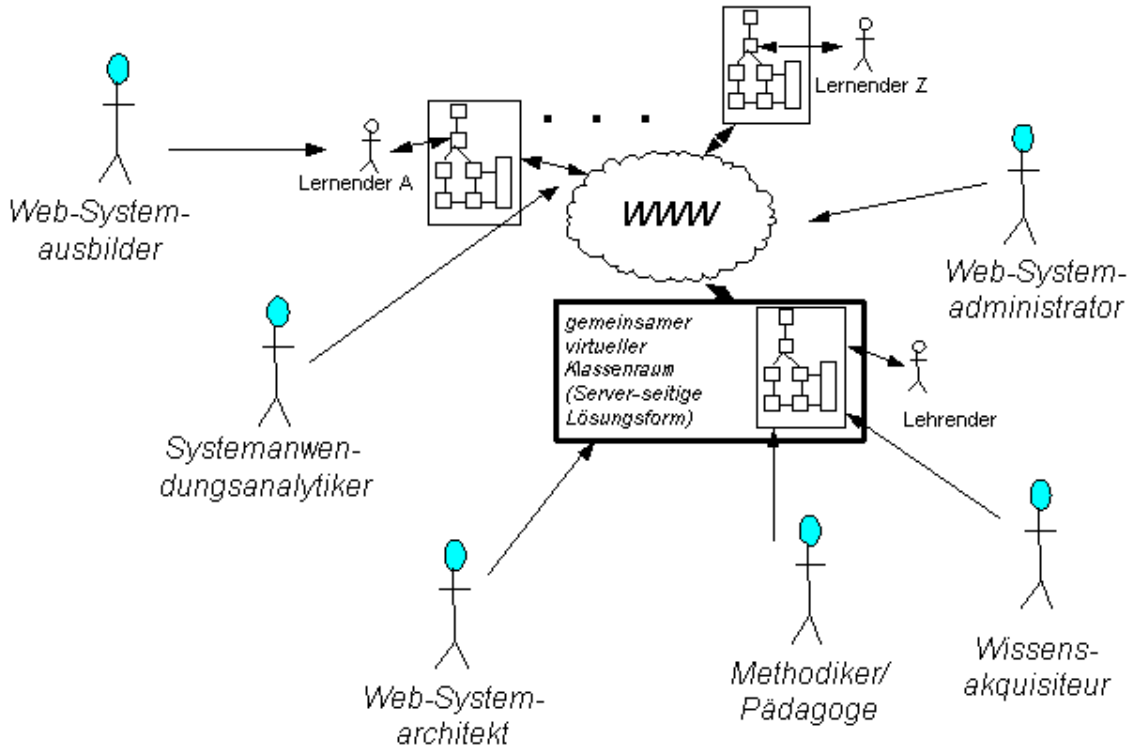


Sie demonstriert insbesondere die nahezu ausschließliche Verwendung des Technologieaspektes für die Systemstrukturierung.

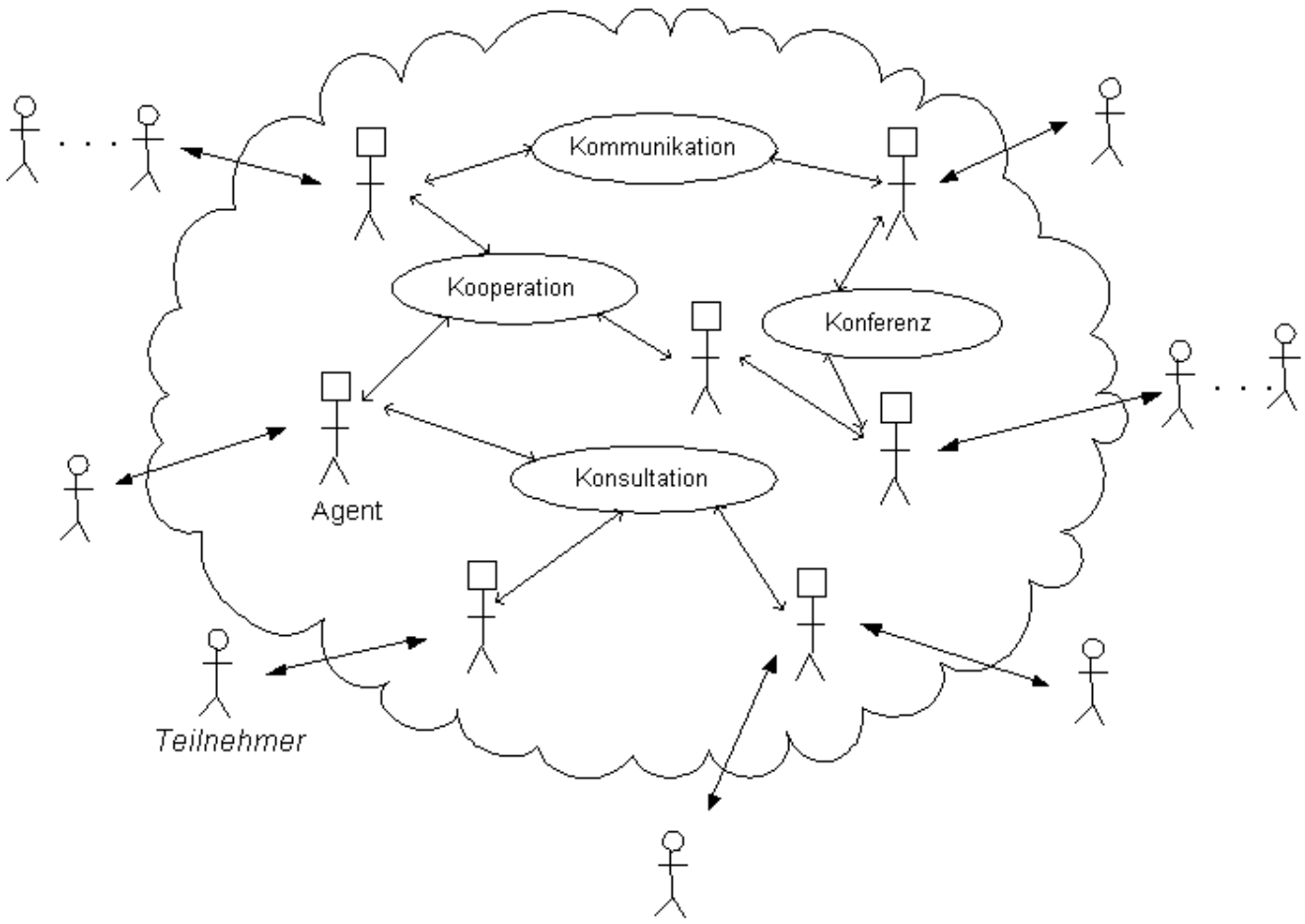
Softwarearchitekturen können auch die Darstellung der jeweilige ***Rollenverteilung*** für deren Nutzung erfordern. Das folgende Beispiel der Darstellung eines einfachen *Telelearnig* impliziert die Rollen des Nutzers sowie die der Ressourcenbereitstellung für derartige Lehr- und Lernsysteme.



Eine komplexere Darstellung für ein ***Teleteaching*** ist in der folgende Abbildung angegeben. Dabei symbolisieren die Rollenangaben den prinzipiellen Einfluss der jeweiligen Personengruppe, der durchaus zu völlig *verschiedenen Zeitpunkten* erfolgen (kann).



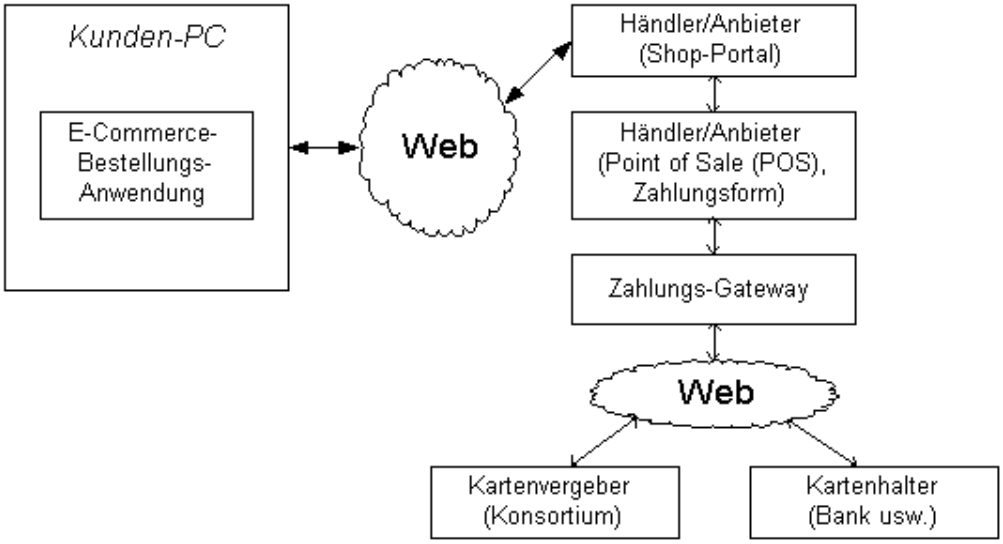
Eine besonders intensive Rollenverteilung bzw. Rollendarstellung haben wir bei agentenbasierten Systemen. Die folgende Abbildung zeigt uns derartige Strukturformen für unterschiedliche kommunikationsbasierte Systemarten.



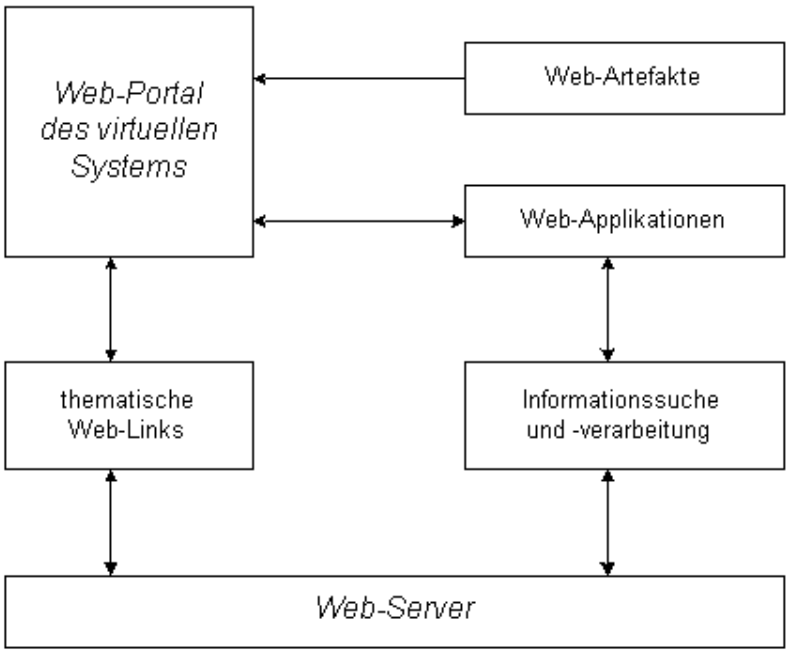
1.6 Integrationsformen von Software-Komponenten

Nachdem die Komponenten derartige Eigenschaften haben, können ***Techniken zur Integration*** *(component integration techniques)* eingesetzt werden. Beispiele dafür sind (siehe [Griffel 98], [Pfleeger 98], [Sametinger 97]):

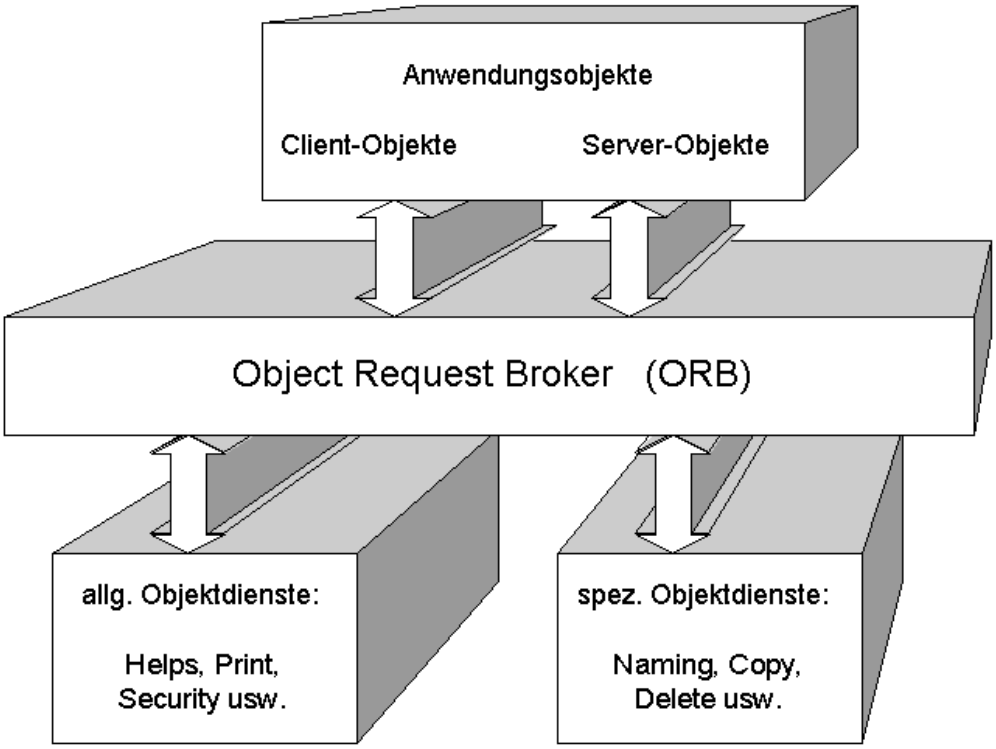
- **Systemtechnische Integration:** Diese Integrationsform bezieht sich vor allem auf die Gewährleistung der Verbindungsmöglichkeit von Komponenten hinsichtlich ihrer Kommunikationsfähigkeit. Ein Beispiel hierfür ist die ***Aggregation*** von Komponenten durch eine daten- oder steuerungsbezogene Interaktionsform. Ein Beispiel dafür ist in der folgenden Abbildung angegeben.



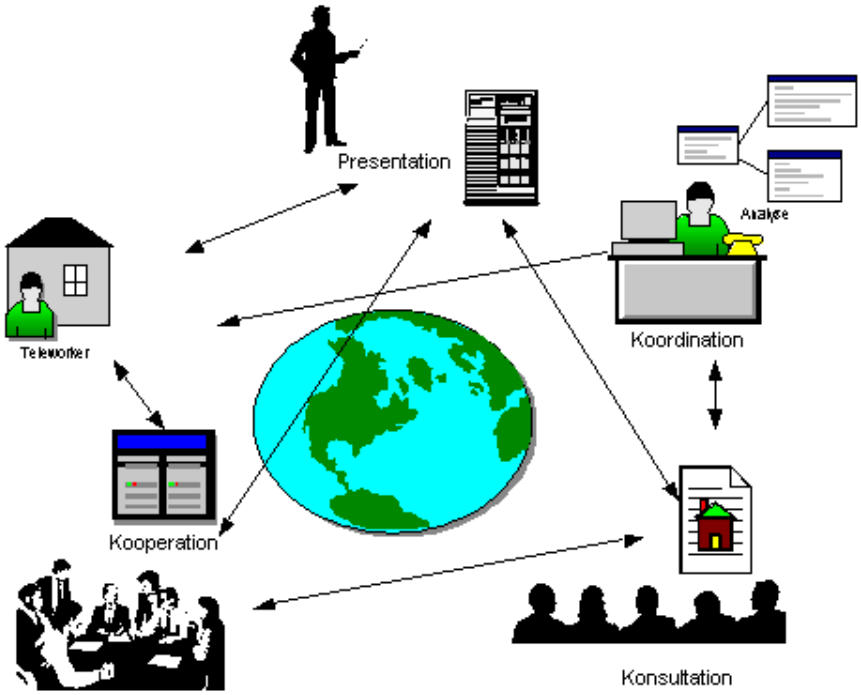
- **Modelltechnische Integration:** Hierbei wird eine Integration auf Modellebene vorgenommen, die eine Abbildung verschiedener oder eine Einordnung in ein übergreifendes Modell sein kann. Beispiele hierfür sind die ***Komposition*** der Komponenten durch eine einheitliche Präsentationsform oder die ***Assoziation*** der Komponenten durch verschiedene Vermittlungstechniken, wie *Middleware* oder *Wrapper-Techniken*. Ein Beispiel für eine Komposition zeigt die folgende Abbildung.



Und schließlich lautet eine Beispiel für eine Assoziation.



- **Prozesstechnische Integration:** Bei dieser Integrationsform sind übergreifende Arbeitsvorgänge bzw. Prozesse möglich. Ein Beispiel für diese Integrationsform ist schließlich die ***Kooperation*** von Komponentenprozessen zur Lösung gemeinsamer Aufgaben. Ein Beispiel zur Kooperation lautet schließlich.



Im Folgenden wollen wir uns nun den Entwicklungsformen von Softwaresystemen unter dem Komponentenaspekt zuwenden und dabei die unterschiedlichen Entwicklungsphasen der Modellierung bzw. Spezifikation und des Entwurfs betrachten.

Literaturverweise:

- [Griffel 98] Griffel, F.: *Componentware - Konzepte und Techniken eines Softwareparadigmas.* dpunkt-Verlag, 1998
- [Pfleeger 98] Pfleeger, S.R.: *Software Engineering - Theory and Practice.* Prentice-Hall Publ., 1998
- [Sametinger 97] Sametinger, J.: *Software Engineering with Reusable Components.* Springer Verlag, 1997

Komponentenbasierte Software-Entwicklung (CBSE)
4. Lehrhilfe: Entwurfskonzepte beim CBSE
FIN, IVS, AG Softwaretechnik

3. Entwurfskonzepte beim CBSE

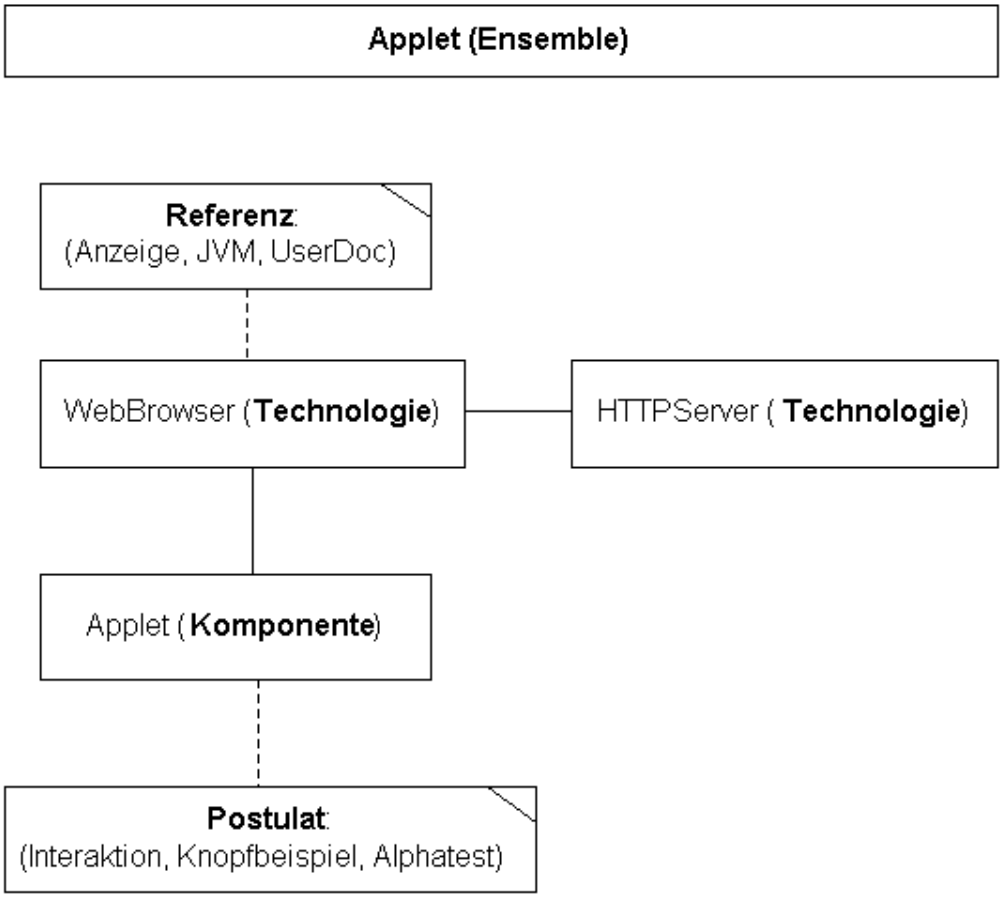
3.1 COTS-basierter Systementwurf (B-COTS)

Das Software Engineering Institute (SEI) in Pittsburgh startete bereits Mitte 90-er Jahre eine Initiative zu COTS und den damit verbundenen Technologien und Anwendungsmethoden [Wallnau 02].

Die Methode von Wallnau et al. als ***Building Systems from Commercial Components***, wir wollen sie **B-COTS** nennen, ist ein komponentenbasiertes CBSE und geht von folgenden Begriffsdefinitionen aus.

- **Kommerzielle Komponente**: Einheit einer Systemimplementation, die durch einen Vertreiber zur Verfügung gestellt wird (verkauft oder als Leasing), im Allgemeinen in kompilierter Form angeboten wird und über ein Interface mit anderen Komponenten verbunden werden kann;
- **Referenz/Zeugnis**: ist das Tripel < *property, value, howVerified* > , welches die Komponentbezeichnung (property), seinen speziellen Wert (value) in diesem Kontext und eine Bestätigung (howVerified) beinhaltet;
- **Postulat**: als Tripel < *property, value, howToVerify* > , welches insbesondere dessen noch ausstehende Prüfung definiert (howToVerify);
- **Ensemble**: ein Ensemble ist eine Menge von Technologien, Produkten und Komponenten, die über Interaktionen ein gewünschtes Systemverhalten gewährleisten;
- **Blackboard**: ist die eine Instantiierung eines Teiles eines Ensemble-Metamodells für einen speziellen Anwendungsbereich;
- **Komponentenraum**: ein *Component Space* ist eine Menge an Komponenten, die durch die drei Dimensionen der *Source of Components*, *Application/Infrastructure* und *Environments* charakterisiert werden.

Für eine kurze Methodencharakterisierung wollen wir uns erst einmal merken, dass wir COTS-basierte Systeme mittels Blackboards in der Art beschreiben, dass neben den eigentlichen Komponenten auch Technologien und andere Integrationsaspekte erfasst werden. Ein solches Blackboard hat beispielsweise den folgenden Inhalt.



Der weitere Verlauf unserer B-COTS-Methode besteht nun im Folgenden:

- die Aufstellung aller relevanten Blackboards für eine Systementwicklung,
- die Nutzung des Blackboards für die Angleichung bzw. Verfeinerung einer COTS-basierten Systemlösung,
- die Erschließung relevanter Komponenteneigenschaften,
- die Konzipierung von Kommunikationsverbindungen (beispielsweise CORBA-basiert mit Hilfe sogenannter Orblets),
- die endgültige Dokumentationsvorlage für eine COTS-basierte Systemlösung.

Wir wollen es bei dieser kurzen Charakteristik belassen, aber noch erwähnen, dass in [Wallnau 02] vor allem noch Hinweise angegeben sind, die eine allgemeine Umsetzung dieser Methodik in einem Unternehmensbereich betreffen. Dabei geht es um solche Fragen, wie der Bildung eines *Component Centers*, der Festlegung von Infrastrukturen zu dessen Wirksamkeit, der Abfassung eines *Enterprise Design Handbook* sowie Fragen der Zertifizierung in diesem Kontext (siehe COTS-Tutorial).

3.2 Asset-Bibliotheken

Wir erinnern uns an die *Software-Wiederverwendung* als hauptsächliche Intention des CBSE und zitieren eine Definition aus [Ezran 98].

> *"**Software reuse** is the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance."*

Diese Definition weist auf folgende grundlegende Merkmale der Software-Wiederverwendung hin:

- Der ausschließliche Bezug auf die Software-Entwicklung (die Wartung ist im gewissen Sinne als "Weiter"-Entwicklung einbezogen). Auf jeden Fall ist die Anwendung als ausschließliche Verwendung von Software ausgeschlossen.

Die Charakterisierung der Wiederverwendungskomponenten als *"stock of building blocks"* bezieht die beiden Komponentenarten bezüglich der Anforderungen einerseits und der Architektur (-komponenten) andererseits mit ein.
- Die Wiederverwendung wird als systematisches Vorgehen charakterisiert.
- Die Ziele der Wiederverwendung liegen also in einer höheren Produktivität, Qualität und Rentabilität.

Die systematische Wiederverwendung erfordert Änderungen im Software-Entwicklungsprozess sowie in der Business- und Organisationsstruktur eines Unternehmens. Das ergibt sich vor allem aus dem mit der Software-Wiederverwendung verfolgten Ziel des schrittweise Übergangs von einer komponentengestützten zu einer komponentenbasierten Entwicklungsform, die nahezu ausschließlich den Einsatz und die Verarbeitung von Komponenten für die Software-Erstellung vorsieht.

Für die Software-Komponenten hat sich im Rahmen der Software-Wiederverwendung die englische Bezeichnung Assets herausgebildet. ***Assets*** bezeichnen also das, was innerhalb der Software-Entwicklung oder -Wartung wiederverwendet werden kann. Die Software-Assets sind (ebenfalls nach [Ezran 98]) wie folgt definiert.

> ***Software Assets*** *are composed of a collection of related software work products that may be reused from one application to another.*

Ein technologisch zusammengefasste Menge solcher Assets bezeichnen wir schließlich als **Asset-Bibliotheken**. Ein spezielles Beispiel einer Asset-Bibliothek ist das folgende Angebot für Java-Programme unter

JAVA ASSETS

Weitere allgemeine Beispiele für Asset-Bibliotheken verschiedenster Art und Ausprägung sind:

- Statistik-Software
- Mathematische Software
- Informix-Anwendungen
- Software-Entwicklung
- Ada-Software
- Linux-Software
- Java-Middleware
- Active-X
- allgemeine Komponenten
- COTS-Y2K-Datenbank

Die kommerzielle Veränderung in diesem Bereich zeigt uns die alte Asset-Seite aus dem Jahre 2000 im folgenden Bild.

und die jetzige WWW.ASSET.COM . Im weitesten Sinne können wir natürlich alle OO-Bibliotheken nutzen, wie zum Beispiel:

- Design Pattern
- STL-Bibliothek
- X11-Bibliotheken
- Software-Bibliotheken
- Programmbibliotheken
- C++-Bibliotheken
- Java-Klassenbibliotheken
- C++-Klassenbibliotheken
- Data-Mining-Klassen
- Tools und Bibliotheken zur Datensicherheit
- Visual C++ Bibliotheken für ODBC

Für die eigene Softwareentwicklung ist es leicht vorstellbar, sich derartige Referenzen ad hoc oder permanent (als eigenes ***Entwicklungsportal***) zu erschließen.

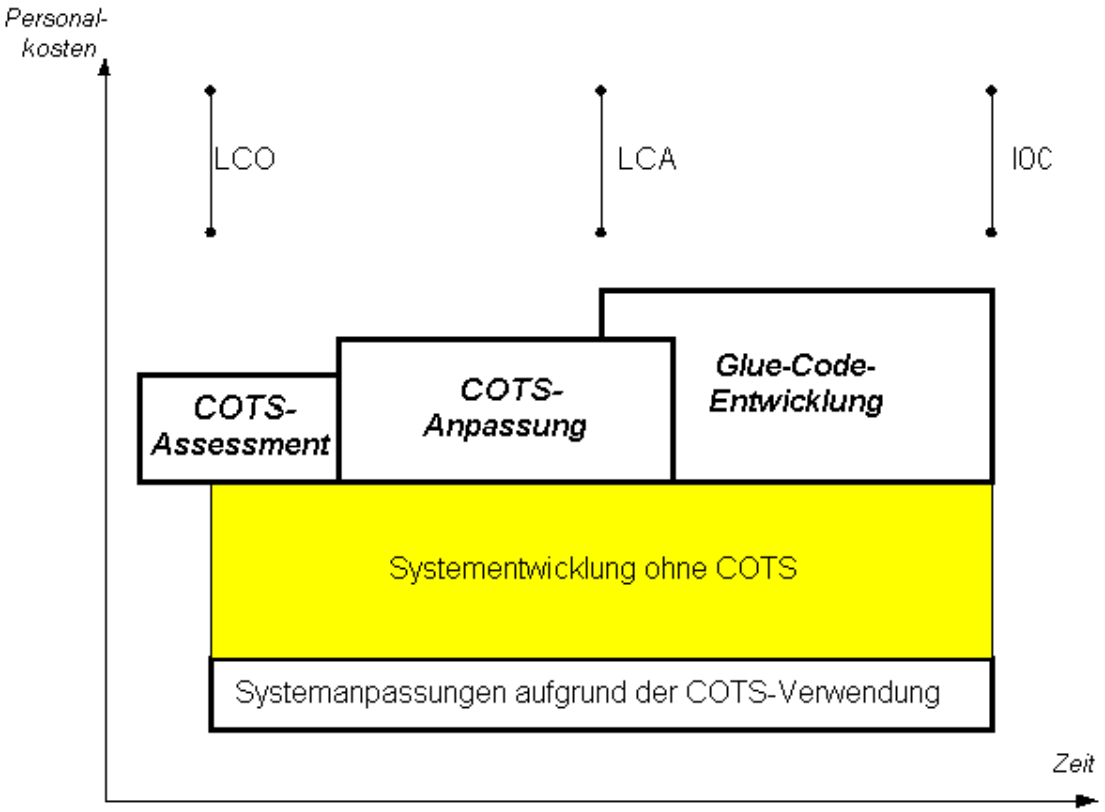
3.3 COCOTS

Das COCOTS (***COnstructive COTS***) spezielle Ausprägung des COCOMO II:2000 für COTS [Boehm 00]). Bei den COTS-Komponenten ist im Allgemeinen der Quellcode für den Nutzer nicht sichtbar bzw. explizit verfügbar. Ebenso hat der Anwender/Nutzer keinen Einfluss auf die (künftige) Entwicklung dieser Komponenten.

Beim COCOMO (*Constructive Cost Moldel*) geht es um die Kostenschätzung einer Softwareentwicklung. Bei den COTS stehen diese Angaben im Allgmeinen nicht zur Verfügung. Dafür kommen solche Kosten, wie *Lizenskosten, Integrationskosten (als Pre-Integration oder Post-Integration), Zertifizierungskosten, Gewährleistungskosten, Anpassungskosten* usw. hinzu. Speziell die Integration erfordert bei der COTS-Verwendung folgende Aufwände:

- Bewertung des COTS-Kandidaten,
- Zuschnitt der COTS-Komponente,
- Entwicklung und Test möglicher Anpassungssoftware (als sogenannter ***Glue Code***),
- Erweiterung der Systemprogrammbasis für eventuelle Effekte der COTS-Einbindung.

Die folgende Abbildung verdeutlicht die Aufwandsquellen und die damit verbundenen Kostenindikatoren beim COCOTS.



Die Abkürzungen LCO, LCA und IOC beziehen sich auf das sogenannte ***Prozessmodell RUP (Rational Unified Process)*** und kennzeichnen die Systementwicklungsstadien als *Anfang (inception)* (mit dem Ergebnis der Lifecycle-Ziele (*Lifecycle Objectives (LCO)*), der *Ausarbeitung bzw. Verfeinerung (elaboration)* (mit dem Ergebnis der Lifecycle-Architektur (*Lifecycle Architecture (LCA)*) und der *Entwicklung (construction)* (mit dem Ergebnis der Anfangsoperationsfähigkeit (*initial Operational Capability (IOC)*).

Der im obigen Bild gekennzeichnete gelbe Bereich kann durch das klassische COCOMO abgeschätzt werden. Siehe hierzu

COCOMO II : 2000

Für die anderen Bereiche wollen wir nun die Berechnungsformen und -inhalte näher betrachten. Dabei unterscheiden wir die vier Schätzbereiche des Assessment (assessment), der Anpassung (tailoring), der Integration (glue code) und der Angleichung (volatility bzw. als Systemanpassung in der obigen Abbildung bezeichnet).

ASSESSMENT:
COTS werden im Allgemeinen durch funktionale und nichtfunktionale Anforderungen, wie Performance, Kosten, Installation, Wartung, Zuverlässigkeit usw., bestimmt. Bei der Abschätzung des Aufwandes für das Assessment wird die folgende Formel angewandt:

Initial Filtering Effort (IFE) = Σ[*Anz*($COTS_{Systemklasse}$) * Ø$IFE_{Systemklasse}$]

Detailed Assessment Effort (DAE) = Σ[*Anz*($COTS_{Systemklasse}$) * Ø$DAE_{Systemklasse}$]

Final Project Assessment Effort (FPAE) = *IFE* + *DAE*

Ø kennzeichnet dabei den jeweiligen Durchschnittswert, der auf die Anwendung der jeweiligen Assessment-Attribute beruht. Dazu zählen:

- Correctness (accuracy, correctness),
- Availability/Robustness (availability, fail safe, fail soft, fault tolerance, inout error tolerance, redundancy, reliability, robustness, safety),
- Security (access related, sabotage related),
- Product Performance (execution performance, information/data capacity, precision, memory performance, response time, throughput),
- Understandability (documentation quality, simplicity, testability),
- Ease of Use (usability/human factors),
- Version Compatibility (downward compatibility, upward compatability),
- Intercomponent Compatibility (with other components, interoperability),
- Flexibility (extendibility, flexibility),
- Installation/Upgrade Ease (installation ease, upgrade/refresh ease),
- Portability (portability),
- Functionality (functionality),
- Price (initial ourchase or lease, recurring costs),
- Maturity (product maturity, vendor maturity),
- Vendor Support (response time for critical problem, support, warranty),
- Training (user trainig),
- Vendor Concessions (will escrow code, will make modifications).

Damit ist der erste Kostenanteil bei der Anwendung von COTS im Rahmen des COCOTS abgeschätzt.

ANPASSUNG:
Anpassungsaufgaben können beispielsweise eine Initialisierung von Parameterwerten, die Layout-Konfiguration der GUI, die Festlegung der Sicherheitstechniken usw. sein. Die formelmäßige Berechnungsgrundlage ist hierbei folgende:

Project Tailoring Effort (PTE) = Σ[*Anz*($COTS_{Systemdomäne}$) * Ø$PTE_{Systemdomäne}$]

Für die Abschätzung der ***Komplexität beim PTE*** wird folgende Tabelle angewandt.

Aktivität	very low	low	nominal	high	very high
Parameterspez.	1	2	3	4	5
Skriptschreiben	1	2	3	4	5
I/O Layout	1	2	3	4	5
GUI-Sepz.	1	2	3	4	5
Security	1	2	3	4	5
Availability	1	2	3	4	5

Auf dieser Grundlage ergibt sich dann eine Komplexitätseinschätzung in der Art:

- 5-7 Punkte: very low,
- 8-12 Punkte: low,
- 13-17 Punkte: nominal,
- 18-22 Punkte: high,
- 23-25 Punkte: very high.

Neben den eigentlichen Kosten kann somit für die Phase der Komponentenanpassung auch noch eine Charakterisierung der Komplexität dieser Aufgabe an sich vorgenommen werden.

INTEGRATION:

Bei der Integration bzw. dem Erstellen des sogenannten Glue Codes für die Komponenteneinbindung wird die Kostenschätzung nach folgender Formel vorgenommen.

*Glue Code Effort (GCE) = A * [(size)(1+CREVOL)]* ^{B} * $\prod$*(effort multipliers)*

wobei gilt

- A - lineare Skalierungskonstante,
- size - Umfang in LOC oder Function Points,
- CREVOL - prozentualer Anteil an Anpassungsarbeit aufgrund der (nachfolgenden) Systemanpassung,
- B - ein architekturbasierter nichtlinearer Skalierungsfaktor,
- effort multipliers - 13 Aufwandsjustierungsparameter.

Diese *Effort multiupliers* sind definiert als:

- personalbezogene "Kostentreiber":
  - ACIEP: COTS-Integrationserfahrung,
  - ACICP: Fähigkeiten des COTS-Integrationspersonals,
  - AXCIP: Erfahrung mit COTS-Integrationsprozessen,
  - APCON: Kontinuität des Integrationspersonals;
- COTS-Komponentenkostentreiber:
  - ACPMT: COTS-Produktgüte,
  - ACSEW: Bereitschaft zur COTS-Produkterweiterung,
  - APCPX: Interface-Komplexität,
  - ACPPS: Produkt-Support,
  - ACPTD: Trainings- und Dokumentationsunterstützung;
- Anwendungssystemkostentreiber:
  - ACREL: Zuverlässigkeitsbedingungen,
  - AACPX: Anwendungssysteminterfacekomplexität,
  - ACPER: Performance-Anforderungen,
  - ASPRT: Anwendungssystemportabilität;
- nichtlineare Faktoren:
  - AAREN: Anwendungsarchitektur-Engineering.

Damit haben wir auch eine allgemeine Kostenschätzung für den Integrations-Code vorgenommen.

ANGLEICHUNG:
Die Kosten für die Systemangleichung werden schließlich nach folgenden Formel berechnet.

*System Volatility Effort (SVE) = (application effort) *
*{[(1+(SCREVOL/(1+REVL)] E - 1} * (COTS effort multipliers)*

wobei gilt

- application effort - Aufwand für neuen Code zur COTS-Integration,
- SCREVOL - prozentualer Anteil an Systemanpassung,
- RVEL - prozentualer Aufwandsanteil für Anpassungen unabhängig von der COTS-Anwendung.

Die effort multipliers haben wir bereits oben definiert. Damit sind wir nun in der Lage den Gesamtkostenaufwand nach dem COCOTS für eine COTS-Anwendung zu schätzen.

Gesamtkosten = FPAE + PTE + GCE + SVE

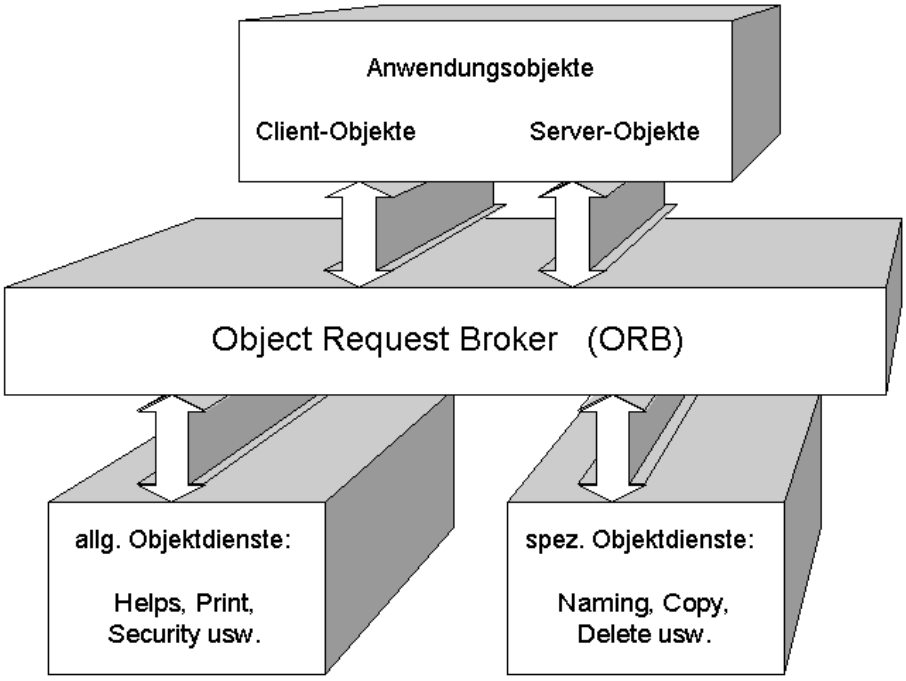
Diese Form der Kostenschätzung hat uns gleichzeitg einen guten Überblick zum Umfeld der COTS-Anwendung selbst gegeben.

3.4 Integrationstechnologien

Da die Integrationsansätze und -formen stets technologiegebunden sind, wollen wir hier zunächst nur eine allgemeine Charakterisierung angeben bzw. andeuten.

3.4.1 Middleware-Ansätze

Neben den unterschiedlichen Anpassungs- oder Überbrückungstechnologien hat sich vor allem die CORBA-Technologie etabliert. Das folgende Bild der OMA deutet diese Form an.



Hinsichtlich der speziellen Technologieform verweisen wir auf die Lehrveranstaltung Verteilte Systeme, in der diese Technologie ausführlich behandelt wird.

3.4.2 XML-basierte Technologien

Auch diese Form wollen wir nur ganz kurz charakterisieren. Mit Hilfe der XML werden hinsichtlich einer Komponentenintegration vor allem realisiert:

- die Definition eines Austauschdatenformates (so sind bereits erste Ansätze zur Ersetzung der *IDL* implementiert),
- die *semantische Integration* mit Hilfe des *Resource Description Framework (RDF)*,
- die Definition und Implementation einfachster Interoperatibilitätsfunktionen mit Hilfe der *DARPA Agent Markup Language (DAML)* unter Anbindung einer speziellen semantischen Schnittstelle als *Ontology Interface Language (OIL)*.

Zu den speziellen Techniken und Inhalten verweisen wir auf die Lehrveranstaltung Web Engineering, die sich dieser Problematik im besonderen widmet.

Literaturverweise:

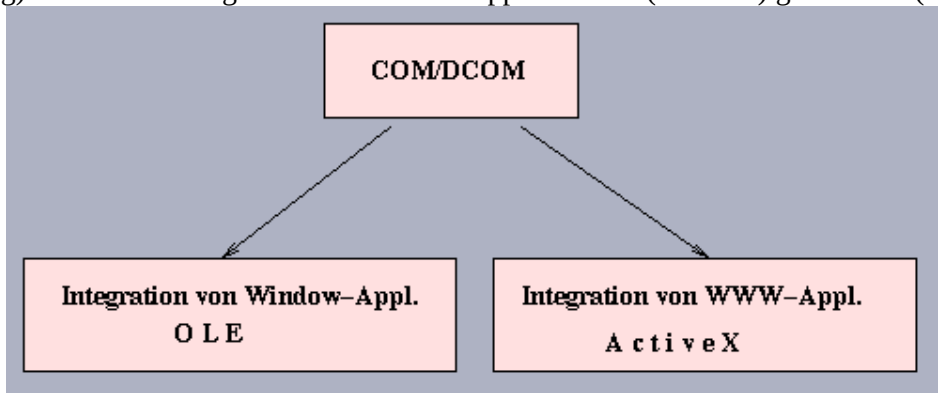
- [Boehm 00] Boehm, B.W. et al.: *Software Cost Estimation with Components.* Prentica Hall Verlag, 2000
- [Ezran 98] Ezran, M.; Morisio, M.; Tully, C.: *Practical Software Reuse: the essential guide.* Freelife Publ., Paris, 1998
- [Wallnau 02] Wallnau, K.C.; Hissam, S.A.; Seacord, R.C.: *Building Systems from Commercial Components.* Addison Wesley Verlag, 2002

Entwicklung (CBSE)
5.Lehrhilfe: Entwurfsstechnologien beim CBSE
FIN, IVS, AG Softwaretechnik

4. Entwurfstechnologien beim CBSE

4.1 COM/DCOM

Das ***(Distributed) Component Object Model*** (DCOM) wurde von Microsoft entwickelt für die Implementation verteilter Applikationen. Auf dieser Grundlage wurde die Technik für die Verbindung von Window-Applikationen (OLE: Object Linking and Embedding) und für die Integration von WWW-Applikationen (ActiveX) geschaffen (siehe DCOM-Tutorial).



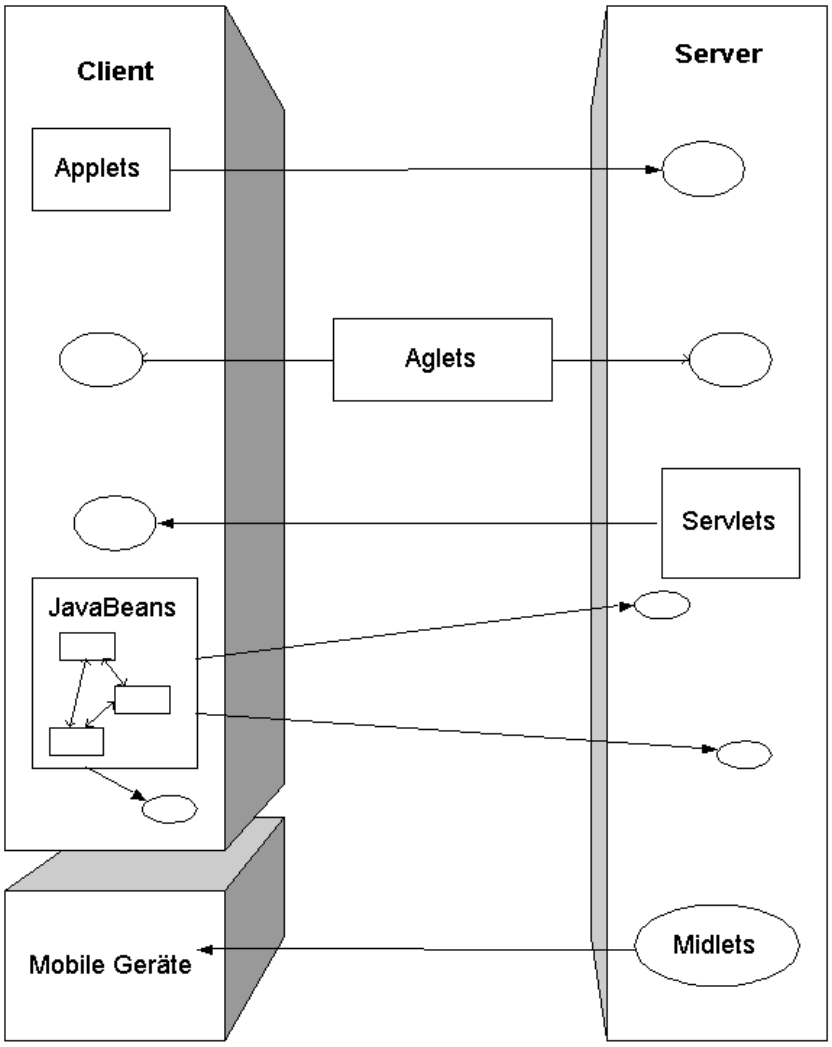
Wesentliche Merkmale dieser DCOM-Techniken sind

- die grundlegende Orientierung auf die RPC-Technik,
- die Verwendung eines Name-Systems analog dem X.500-Protokoll,
- die Verwendung von Public-Key-basierten Verschlüsselungstechniken für die Sicherheitsaspekte,
- die Verwendung einer speziellen Management-Komponente (MMC),
- die Realisierung der Wiederverwendung über Aggregation (nicht über Vererbung),
- die Transaktionsverarbeitung mittels eines speziellen Microsoft Transaction Servers (MTS).

Die DCOM-Techniken werden hauptsächlich von Digital, Hewlett Packard, SAP und Microsoft selbst verwendet.

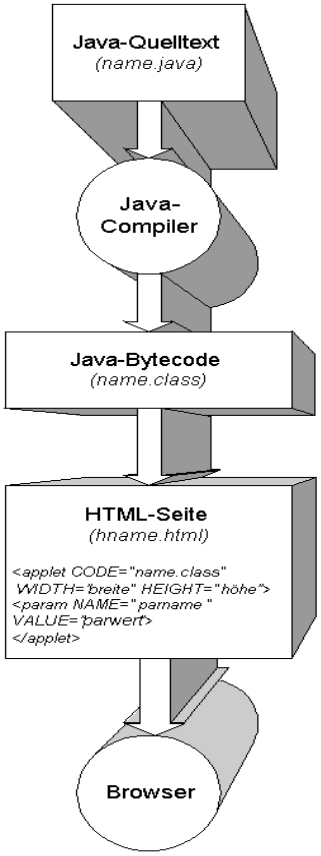
4.2 Java-Technologien

Für die weitere Dynamik in Web-Dokumenten können vor allem die "Java-lets"-Technologien verwendet werden. Dabei liegt als Architekturprinzip hierbei zumeist das Client/Server-Prinzip zugrunde. Die folgende Abbildung gibt einen Gesamtüberblick.



JAVA-APPLETS:

Als erstes wenden wir uns den Java-Applets zu. Das Anwendungsprinzip der Java-Applets zeigt uns das folgende Bild.



Darin zeigt uns die mittlere Komponente (als HTML-Seite) die allgemeine Charakterisierung einer Java-Applet-Anwendung. Ein *Java-Applet* hat folgende allgemeine Merkmale:

1. ein Applet ist kein unabhängiges Programm, d. h. es besitzt *keine Main-Methode*,
2. die Ein- und Ausgaben werden ausschließlich über die durch das Applet implementierte GUI realisiert,
3. ein Applet wird durch einen Web-Browser nach dem sogenannten *Sandkastenprinzip* (Sandbox) abgearbeitet, d. h. es hat keine Rechte für die Bearbeitung lokaler (Client) Files usw. (so kann man beispielsweise auch keine Druckausgaben in einem Applet definieren),
4. Applets werden im allgemeinen *ereignisgesteuert* konzipiert.

Die Programmierung eines Applet sehen wir uns an einem einfachsten Beispiel an. Wir wollen einfach eine Schaltfläche definieren, die anzeigt, wie oft sie betätigt wurde. Wir formulieren daher wie folgt:

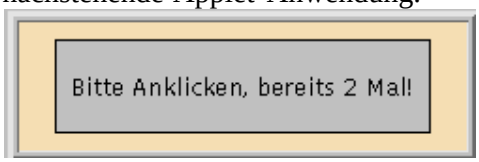
```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.event.*;

public class Anklicken extends Applet {
    private int n = 0;
    public void init() {
       addMouseListener(new Klicken( this));
    }
  public void doClick()
     { n++; repaint(); }
  public void paint(Graphics g) {
       g.drawRect(0, 0, getSize().width-1,
                  getSize().height-1);
       g.drawString( " Bitte Anklicken, bereits " +
                  n + " Mal!", 5, 30);
    }
 }
class Klicken extends MouseAdapter {
    private Anklicken app;
    Klicken(Anklicken ap)
       { app = ap; }
    public void mouseClicked(MouseEvent e)
      { app.doClick(); }
}
```

Genau diese Klasse übersetzen wir mit dem Java-Compiler und fügen sie mittels der folgenden Anwendungen

```
<APPLET CODE="Anklicken.class" WIDTH=200 HEIGHT=50> </APPLET>
```

in einen HTML-Text ein und erhalten die nachstehende Applet-Anwendung.



Die Klasse Applet hat folgende grundlegende Methoden:

destroy() : zerstören (beenden) des Applet,

init(): Initialisieren des geladenen Applet (im obigen Beispiel ausreichend, da ereignis- (maus-) gesteuert),

start(): explizites Starten eines Applet,

stop(): Anhalten eines Applet,

getParameterInfo(): Angabe der Applet-Parameter,

getAppletInfo(): Informationen zum Applet als Zeichenfolge,

isActive(): gibt den gegenwärtigen Bearbeitungszustand des Applet zurück,

getAudioClip(): ruft Audioclip über URL auf,

getImage(): stellt Bild über URL bereit,

play(): spielt einen Audioclip ab,

getCodeBase(): stellt des Applet-Code über angegebener URL bereit,

getDocumentBase(): entnimmt die Applet-Dokumentation über die angegebene URL,

getParameter(): stellt die im Applet angegebenen Parameterwerte bereit,

getAppletContext(): liefert den sogenannten Applet-Kontext,

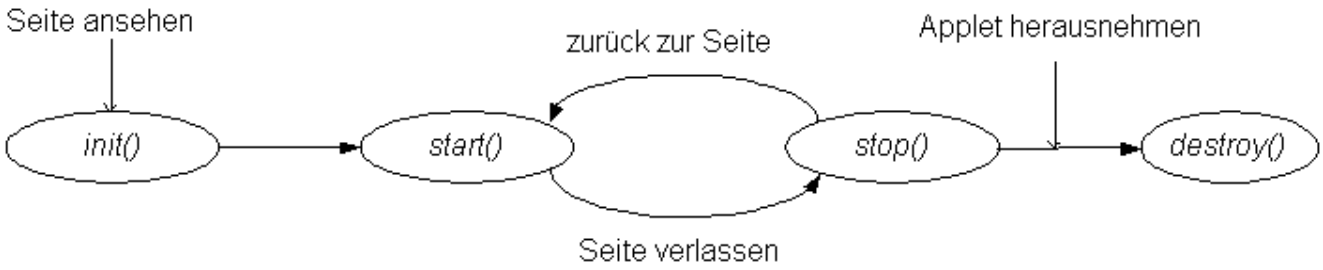
showStatus(): zeigt eine Nachricht über den Status im Applet-Kontext an,

getLocal(): stellt das Applet aus lokaler Adresse bereit,

resize(): verändert die Applet-Fläche im Browser,

setStub(): bildet einen Applet-Vertreter.

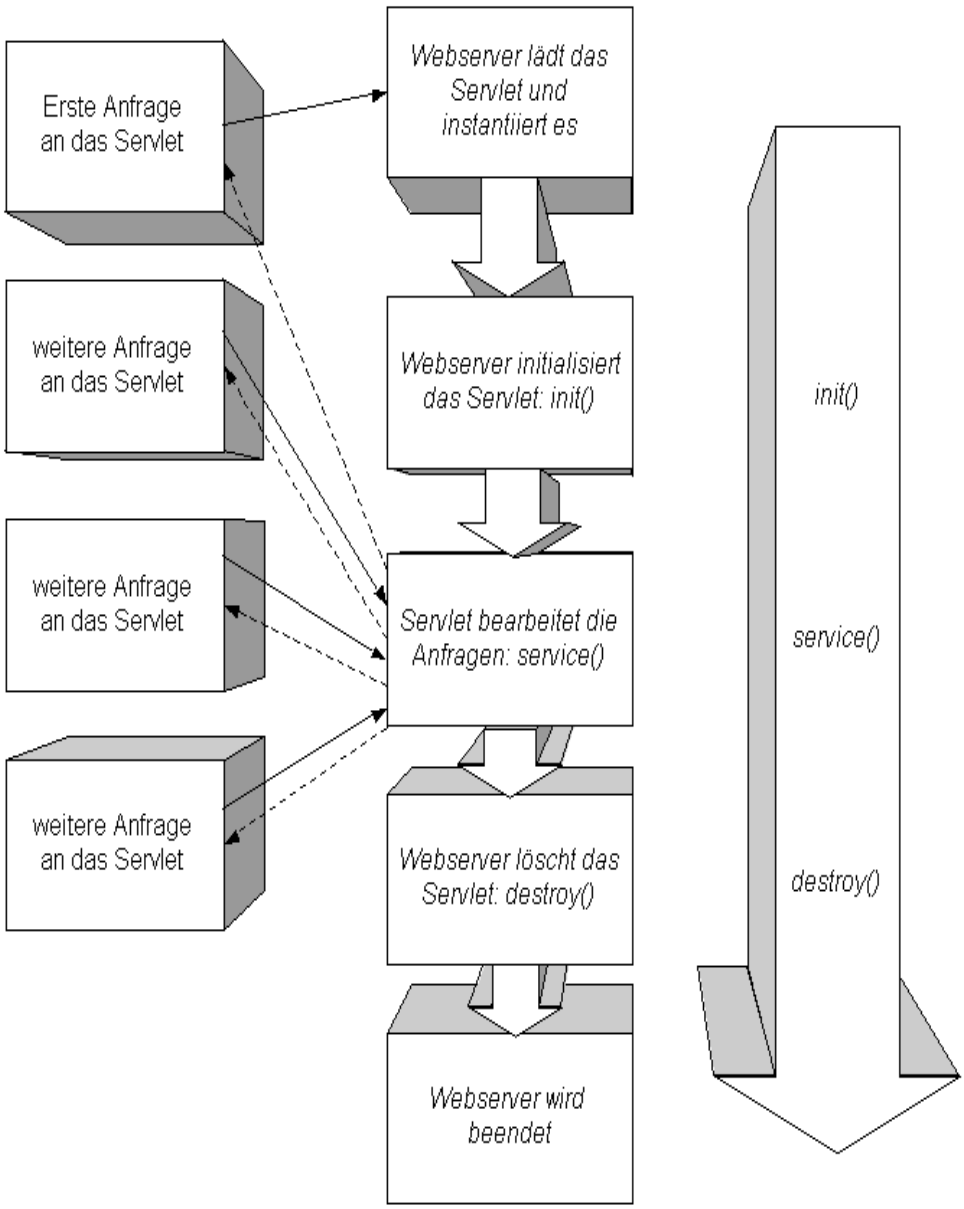
Hinsichtlich der Bearbeitungszustände haben wir bei einem Applet die typischen Prozesszustände bzw. -übergangsformen. Die folgende Abbildung kennzeichnet diese Statusformen bei einem Applet, wie sie sich in einer Anwendung in einem (HTML-) Browser ergeben.



JAVA-SERVLETS:

Als nächste Java-Technik wenden wir uns den Java-Servlets zu. Java-Servlets haben ihre Bezeichnung von den Applets, die "serverseitig" implementiert werden können und somit einen ständigen Dienst bereit stellen. Die

Entwicklungsumgebung ist durch das sogenannte *Java Servlet Development Kit (JSDK)* gegeben und stellt hierbei eine ***Servlet-Engine*** bereit, die die Testung der entwickelten Servlets gestattet. Servlets stellen also Java-Programme auf der Server-Seite dar, die beispielsweise den Webserver um *dynamische Komponenten* erweitern. Sie laufen auf dem Server und verursachen daher beim Anwender selbst keinen entsprechenden Ressourcen-verbrauch. Im folgenden geben wir den allgemeinen Lebenszyklus eines Servlets an.



Servlets besitzen unter anderem folgende Eigenschaften für die Implementation flexibler Web-Anwendungen:

1. Servlets können Daten aus dem sogenannten *Secure Hypertext Transfer Protokoll (SHTTP)* verarbeiten, wodurch beispielsweise auf Kreditkarten-informationen aus einer HTML Seite zugegriffen werden kann.

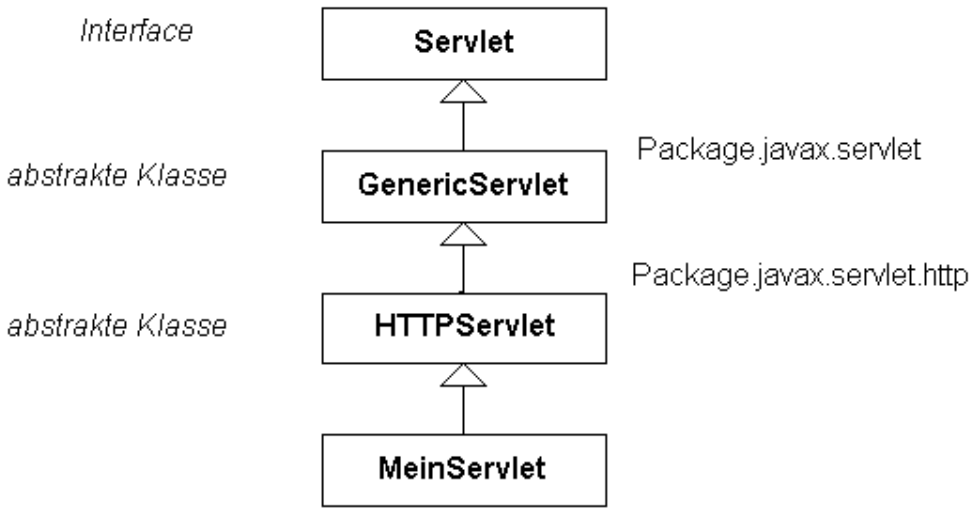
2. Die Threads der Servlets lassen sich synchronisieren. Damit können zum Beispiel Online-Konferenzen mit mehreren Clients implementiert werden.
3. Durch eine Aufgabenteilung hinsichtlich der Zusammenarbeit von mehreren Servlets auf verschiedenen Rechnern ist es möglich, Überlastsituationen abzufangen im Sinne eines allgemeinen Lastausgleichs.
4. Nach der Verbindungsaufnahme zwischen einem Client und dem Server über das HTTP-Protokoll kann über ein frei wählbares Protokoll kommuniziert werden.
5. Dynamische Code-Erweiterungen sind möglich, wobei Java-Bytecode zur Laufzeit auf den Server geladen und ausgeführt werden kann.

Schließlich wollen wir auch hier einen *Hello-World-Servlet* angeben in der folgenden Java-Implementation.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
           IOException {
  res.setContentType("text/html");
  PrintWriter out = res.getWriter();
  out.println("<HTML>");
  out.println(
        "<HEAD><TITLE>Hello World</TITLE></HEAD>");
  out.println("<BODY>");
  out.println("<BIG>Hello World</BIG>");
  out.println("</BODY></HTML>"); }
}
```

Das folgende Bild zeigt uns den einfachen Aufruf dieses Servlets und die entsprechende Ausgabe.

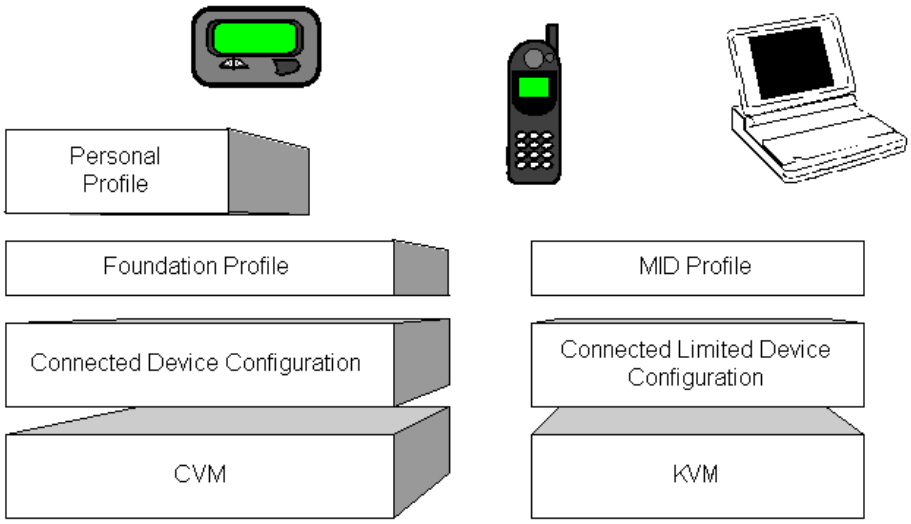
Einer Servlet-Implementation liegt die folgende Hierarchie zugrunde, die die Klassenstruktur einer derartigen Komponente darstellt.



Servlets werden häufig genutzt, um beispielsweise Besucheranzahlen oder andere Statistiken zu Web-Seiten anzuzeigen.

JAVA-MIDLETS:

Midlets (auch MIDlets geschrieben) stellt ein Sammelbegriff für die Java-Technologie für **mobile Geräte** *(Mobile Information Devices (MID)),*wie zum Beispiel Handies, PDAs, Handhelds usw. dar. Die Entwicklungskomponenten für diese Technik sind in der *Java 2 Platform Micro Edition (J2ME)* enthalten. Die allgemeine Architektur dieser Technologie zeigt uns die folgende Abbildung.



Das folgende Programmbeispiel dient der Darstellung einer temporären Zeichenfläche auf einem mobilen Gerät.

```
import java.util.*;
import javax.microedition.lcdui.*;

public class MeinScreen extends Canvas {
  private Display  display;
  private Displayable next;
  private Timer  timer = new Timer();
  public MeinScreen( // Initialisierung
    Display display, Displayable next ){
    this.display = display; this.next = next;
    display.setCurrent( this );
   }
protected void keyPressed( int keyCode )
    {
     dismiss();
    }
protected void paint( Graphics g ){
  // Angabe der gewünschten Zeichenfunktion
  }
protected void pointerPressed( int x, int y )
  // Screen verschwindet beim Druecken einer Taste
    {
      dismiss();
    }
protected void showNotify(){
  // Screen verschwindet nach 5 Sekunden
     timer.schedule( new CountDown(), 5000 );
   }
private void dismiss(){
     timer.cancel();
     display.setCurrent( next );
   }
private class CountDown extends TimerTask {
  public void run(){
    dismiss();
   }
 }
}
```

Die Midlet-Komponentenart besitzt also einen mobilen Charakter über drahtlose Geräteverbindungen und ist im allgemeinen durch Platzrestriktionen gekennzeich-net. Die dabei erreichte Funktionalität ist ebenfalls durch die Einschränkung der KVM (s. u.) bestimmt.

JAVABEANS

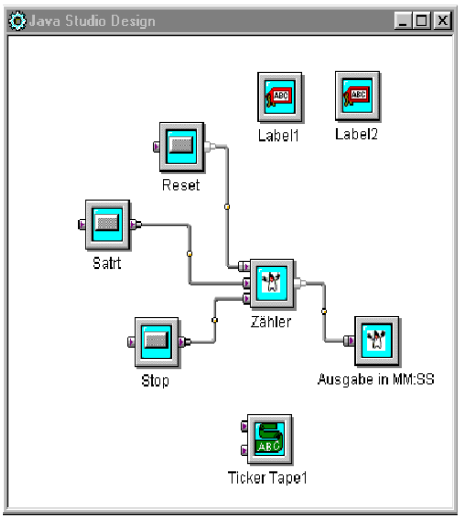
JavaBeans ist eine Technik, die *Komponentenbasierte Software-Entwicklung* auch für die Programmiersprache Java zu realisieren. Nach dem Entwickler *Sun* sind JavaBeans wie folgt definiert:

> *"JavaBeans components, or Beans, are reusable software components that can be manipulated visually in a builder tool. Beans can be combined to create traditional applications, ot their smaller web-oriented brethren,*

applets. In addition, applets can be designed to work as reusable Beans."

Das folgende Bild von Schmietendorf zeigt eine visualisierte Software-Architekturform entwickelt mit dem *JavaBuilder* und die Form der zugehörige Anwendung.

Entwicklung unter dem JavaStudio:

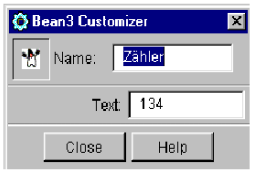


Verwendete Beans:

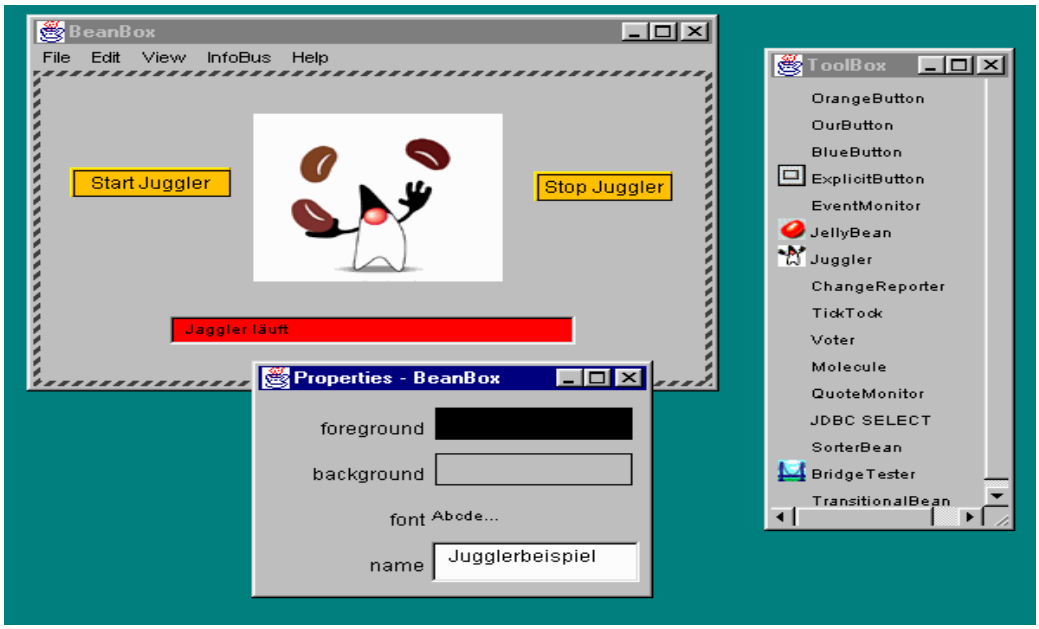
(visuelles Plazieren, angebotene Connectoren)

- Button-Beans (Start, Stop, Reset)
- Zähler (Impulsgeber alle 1 sek.)
- Ausgabe-Bean (Stopuhr)
- Label zur Beschriftung
- Laufschrift-Bean

Customize-Dialog:



In unserem Preprint zur EJB haben wir dieses Beispiel in seiner Entwicklung ausführlich bechrieben. Das folgende Bild deutet die Teilschritte mit dem JBuilder für eine Juggle-Animation noch einmal an.



Für die Aneignung grundlegender Kenntnisse zu den JavaBeans verweisen wir auf das folgende

JAVABEANS-TUTORIAL

Komponentenbasierte Software-Entwicklung (CBSE)

6. Lehrhilfe: Qualität von Software-Komponenten

FIN, IVS, AG Softwaretechnik

5. Qualität von Software-Komponenten

5.1 Erfahrungen mit Software-Komponenten

Für die Qualität bzw. die dadurch implizierte Nutzbarkeit und Eignung von Software-Komponten existieren bereits eine Reihe von Erfahrungen. Wir beginnen mit der Angabe einiger Links zu diesem Thema.

- Komponentenqualität
- COTS-Erfahrungen
- COTS-Kostenschätzung
- Middleware-Review
- JavaBeans-FAQ

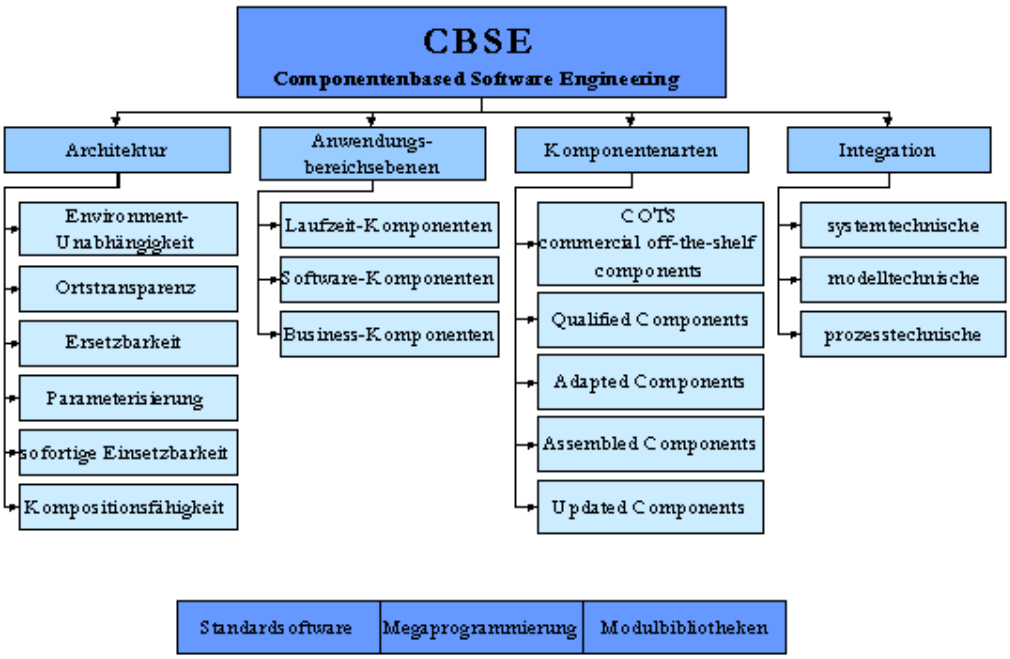
Allgemeine Erfahrungen als sogenannte ***Hypothesen*** definierten von Basili und Boehm [Basili 01] :

1. ***More than 99 percent*** *of all executing computer instructions come from COTS products. Each instruction passed a market test for value.*
2. ***More than half of the features*** *in large COTS software products* ***go unused***.
3. *The average COTS software product undergoes a* ***new release every eight to nine months****, with active vendor support for only its latest three releases.*
4. *CBS development and postdeployment efforts can scale as high as* ***the square of the number of independently developed COTS*** *products targeted for integration.*
5. *CBS postdeployment costs exceed CBS development costs.*
6. *Although* ***glue-code*** *development usually accounts for* ***less than half the total CBS development effort****, the effort per line of glue code averages about three times the effort per line of developed-applications code.*
7. *Nondevelopment costs, such as* ***licensing fees*** *, are significant, and projects must plan for and optimize them.*
8. *CBS assessment and tailoring efforts vary significantly by COTS product classes - operating system, database management system, user interface, device driver, and so forth.*
9. *Personnel capability and experience remain the* ***dominant factors*** *influencing CBS-development productivity.*
10. *CBS is currently a* ***high-risk activity****, with effort and schedule overruns exeeding non-CBS software overruns. Yet many systems have used COTS successfully for cost reduction and early delivery.*

Danach wollen wir uns nun den speziellen Qualitätsaspekten sowohl beim komponentenorientierten als auch beim komponentenbasierten CBSE zuwenden.

5.2 Qualitätsaspekte von Software-Komponenten

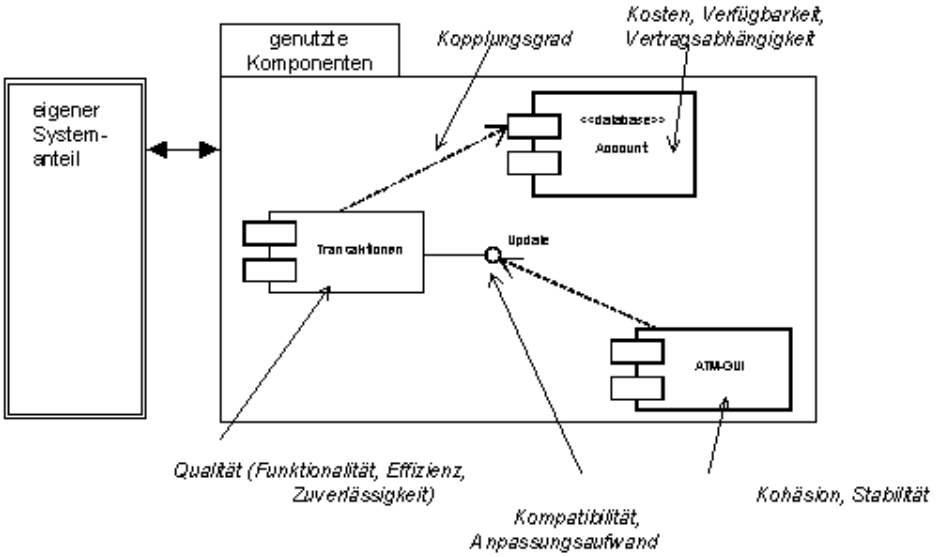
Zur Qualitätsbetrachtung geben wir noch einmal einer kurze Gesamtübersicht nach [Blazey 02] zu den allgemeinen Ausprägungen des CBSE an.



Daraus ergeben sich sich folgende allgemeine Ansätze bzw. Aspekte für die Qualitätssicherung beim CBSE:

- **Komponentenqualität:** mit den typischen Merkmalen der Programm- und Produktqualität,
- **Architekturqualität:** mit den Aspekten der modularen seiteneffektfreien Ersetzbarkeit bzw. der Ebenen- und Schichtenqualität für n-tier-Formen,
- **Intergationsqualität:** mit den Qualitätsmerkmalen der jeweiligen Intergrationstechnik selbst,
- **Anwendungsqualität:** mit den Merkmalen der qualitativen Auswirkungen im Anwendungskontext.

Die grundsätzlichen Ansatzpunkte für die komponentenbezogene Software-Messung bzw. deren Qualitätsbewertung zeigt uns die folgende Abbildung [Schmiet 02a].



In anderen Lehrveranstaltungen haben wir bereits zu den meisten dieser Aspektespezielle Maße oder Metriken kennengelernt. Siehe hierzu

- die Ausführungen zur Produktqualität in der Lehrveranstaltung Softwarequalitätsmanagement,
- die Links zur Softwaremessung überhaupt im Rahmen der Lehrveranstaltung Softwaretechnik I.

Dazu werden wir natürlich etwas üben. Speziell die Bewertung der **Wiederverwendung** wird vor allem nach dem Aspekt

vorgenommen, wie der Entwicklungsaufwand und die damit verbundenen Kosten gesenkt werden können. Das bedeutet die Einschätzung eines Kostenvorteils ***(cost benefits)*** bzw. einer Rentabilität der bei der Software-Wiederverwendung eingesetzten Mittel ***(return on investment (ROI)).*** Darüber hinaus ist natürlich auch ein möglicher Produktivitäts- und Qualitätsgewinn von Bedeutung. Die folgende Tabelle gibt einige Beispiele von Metriken zur Bewertung der oben genannten Kriterien an (nach [Poulin 97] und [Sodhi 99]).

Metrik	Messform	Berechnung
Amortization nach Gaffney	Berechnung	(RCR + RCWR/n –1) * R +1, n – Anzahl erwarteter Wiederverwendungen, R – Anteil wiederverwendeter Code im Produkt
COCOMO Modification nach Balda	Schätzung bzw. Messung	$LM = a_i N_i^b$ N_1= KDSI für einmalige Codeverwendung N_2= KDSI für geplante Wiederverwendung N_3= KDSI vom wiederverwendeten Code N_4= KDSI von modifizierten Komponenten (KDSI – Kilo delivered source instruction)
Cost Benefit nach Bollinger	Schätzung	Benefit = withoutReuse – withReuse - ReuseInvestment
Cost Benefit nach Malan	Schätzung	Costs= (withoutReuse – withReuse) - ReuseOverhead
RCA nach Poulin	Schätzung	ReuseCostAvoidance = DevelopmentCostAvoidance + ServiceCostAvoidance
RCR (relative reuse costs) nach Poulin	Schätzung	Aufwand für die Verwendung einer Komponente ohne Modifikation (black-box reuse)
RCWR (reuse costs for writing reuse component) nach Poulin	Schätzung	Aufwand für die Entwicklung einer für die Wiederverwendung vorgesehenen Komponente
Reusable Index nach Sodhi	Bewertung	Komponentenbewertung: 4 – most reusable, . . ., 1 – least reusable
Reuse Leverage nach Poulin	Schätzung	ProductivityWithReuse/ ProductivityWithoutReuse
Reuse Percent nach Poulin	Schätzung	ReusedSoftware/TotalSoftware
RVA nach Poulin	Berechnung	Reuse Value Added= (TotalSource Statements + SourceInstructionsReused ByOthers)/ (TotalSourceStatements- ReusedSourceInstruction)

Hierbei ist allerdings zu erkennen, dass beispielsweise die Metriken, die sich auf eine Gegenüberstellung einer mit und ohne Wiederverwendung realisierten Entwicklung beziehen, zwar für die Bewertung der Wiederverwendung insgesamt aber nicht für den Gebrauch bei der Entwicklung selbst anwendbar sind. Dennoch können auf der Grundlage der anderen Metriken Erfahrungsdaten gesammelt und für eine Trendanalyse bzw. Prozessbewertung eingesetzt werden.

5.3 Qualitätssicherung beim komponentenorientierten CBSE

Die komponentenorientierte Entwicklung berücksichtigt die Idee einer Komponentenbildung während der Systementwicklung selbst. Die dabei entstehenden sogenannten ***In-house-Komponenten*** sind so zu entwickeln, dass bei späteren Wartungsarbeiten insbesondere die Gewährleistung der Kompatibilität der in mehreren Systemen eingesetzten Komponenten gesichert ist. Quantitative Bewertungsmöglichkeiten sind zum Beispiel in [Dumke 97] und [Takeshita 97] angegeben.

Generell gelten für diesen Bereich die Qualitätssicherungsmethoden und -verfahren der Softwareentwicklung überhaupt. Wir verweisen daher noch einmal explizit auf das Skript der LV SQA .

Im wesentlichen sollen dabei allerdings die bereits im ersten Abschnitt angegebenen Anforderungen an Komponenten Berücksichtigung finden.

5.4 Qualitätssicherung beim komponentenbasierten CBSE

Hierbei ist die Situation, wie oben zum Teil schon beschrieben, in der Hinsicht problematisch, als das im Allgemeinen keine Entwicklungsdokumentation vorliegt bzw. die Informationen zu COTS generell lückenhaft sind.

Beispielsweise für den ***Software-Test*** *(testing component-based software)* bedeutet das eine höhere Anforderung an den Funktions- bzw. Black-Box-Test (siehe zum Beispiel [Weyuker 98]). Da bei den verwendeten Komponenten, wie zum Beispiel den COTS, der Quellcode nicht zur Verfügung steht, muss eine Erweiterung der Funktionstestfälle durch den Entwickler vorgenommen werden, die beispielsweise den Entscheidungsraum der Funktionalität der Komponenten nahezu abdeckt. Ein anderer Input für den Test von komponentenbasierten Systemen ist die Bewertung der Qualität durch indirekte Faktoren, wie beispielsweise das Prozess- und Ressourcenniveau der jeweiligen Komponentenhersteller.

Die besondere Problematik der COTS-Akquirierung und -Anwendung hat bereits beim CBSE zu einer speziellen Begriffsbildung, dem ***CURE*** *(COTS usage risks evaluation)* geführt. Auf die CURE-Technologie wollen wir allerdings hier nicht näher eingehen und verweisen auf die Web-Adresse *Tools Workshop* .

Eine ähnliche Ergänzung wie der Software-Test muss für das CBSE auch die ***Software-Wartung*** *(maintaining component-based systems)* erfahren (siehe [Jell 98] und [Voas 98]). Die Bearbeitung bzw. möglicherweise Ersetzung von Komponenten ist bei der Software-Wartung dadurch geprägt, dass es sich bei diesen Komponenten nicht nur um COTS, sondern auch um *Freeware*, *Public-Domain-Software* oder auch *Shareware* handeln kann, für die

- keine vertraglichen Regeln und damit auch keine Garantie- oder andere Haftungsformen bestehen,
- hinsichtlich der Funktionserweiterung oder Anpassung an neue Hard- und Software-Anforderungen kaum Einfluss genommen werden kann.

Einen allgemeinen praktischen Ansatz wollen wir mit der folgenden Auflistung von **Beurteilungskriterien der DB für Fremdsoftware** nach [Frenzel 99] angeben.

- ***Abdeckung der Anfoderungen:***
  - Grad der Abdeckung des Anforderungskataloges;
- ***Allgemeine Produktbetrachtung:***
  - Verbreitung im deutschsprachigen Raum,
  - Bestehen einer Benutzervereinigung,
  - Weiterentwicklungsstrategie des Herstellers erkennbar,
  - Versionsplanung des Herstellers,
  - Grad der Abhängigkeit vom Hersteller;
- ***Systemtechnische Betrachtung:***
  - Notwendigkeit zusätzlicher Hard- und Software,
  - Möglichkeit einer Testinstallation,
  - problemlose Durchführbarkeit von Versionswechseln,
  - Möglichkeit des Austausches von Moduln,
  - Behinderung durch parallel laufende Systeme,
  - Zeit- und Verbrauchsverhalten;

- ***Programmtechnische Betrachtung:***
    - Verwendung offizieller Schnittstellen,
    - Verwendung von Steuerdaten,
    - Vorhandensein einer Testumgebung;
- ***Schnittstellen:***
    - Vorhandensein von Schnittstellen für spezielle Unternehmensbelange,
    - Vorhandensein von Schnittstellen zu anderen Systemen;
- ***Service und Wartung:***
    - Servicebereitschaft an Wochenenden und Feiertagen,
    - Telefonservice und Ferndiagnose,
    - Zeitraum der Fehlerbehebung,
    - Vorhandensein eines Frühwarnsystems;
- ***Sicherheitsanforderungen:***
    - Möglichkeiten der Kontrolle, Steuerung und Protokollierung;
    - Einhaltung des Datenschutzes,
    - Möglichkeit der Vergabe von Zugriffsberechtigungen,
    - Ablaufsicherheit,
    - Vollständigkeit, Nachvollziehbarkeit der Verfahren und Ergebnisse,
    - Einhaltung der Grundsätze ordnungsgemäßer Buchführung,
    - Möglichkeit der Verschlüsselung von Daten,
    - Möglichkeit der Anpassung an gesetzliche Bestimmungen,
    - Datensicherheit;
- ***Benutzeroberfläche:***
    - Ergonomie des Maskenaufbaus,
    - Vorhandensein von Hilfefunktionen,
    - Ergonomie der Menüsteuerung;
- ***Dokumentation:***
    - Aktualität und Vollständigkeit der Dokumentation,
    - Eindeutigkeit, Verständlichkeit und Widerspruchsfreiheit,
    - Sprache der Dokumentation,
    - Vorhandensein von Daten- und Funktionsmodell;
- ***Schulungsmaßnahmen:***
    - Umfang, Dauer, Sprache und Kosten der Schulungen;
- ***Kostenbetrachtung:***
    - Lizenskosten, Einmalkosten,
    - Wartungskosten,
    - Kosten der Umstellungsmaßnahmen im Unternehmen;
- ***Hersteller/Vertreiber:***
    - Rechtsform, Sitz,
    - Eigenkapital, Gewinn- und Verlustrechnung,
    - Qualitätsmanagement,
    - Bekanntheitsgrad, Ruf,
    - Produktpalette,
    - Referenzkunden;

Vertragsinhalte:

- Nutzungsrechte,
- Übergabe bzw. Hinterlegung des Quellcodes,
- Eindeutigkeit der Definition von Wartung und Erweiterung,
- Haftungsregeln, Vertragsstrafen,
- Datenschutz, Geheimhaltung,
- Gerichtsstand,
- Vertragsdauer, Konventionen.

Diese Auflistung impliziert natürlich noch eine weitere Bearbeitung für die Anwendung dieser Kriterien, die insbesondere in der Auswahl bzw. Erarbeitung von ***Maßen oder Metriken*** besteht. Eine ähnliche Untersuchung auf diesem Gebiet - speziell für den Integrationsaufwand - ist in [Wartmann 98] beschrieben.

Literaturverweise:

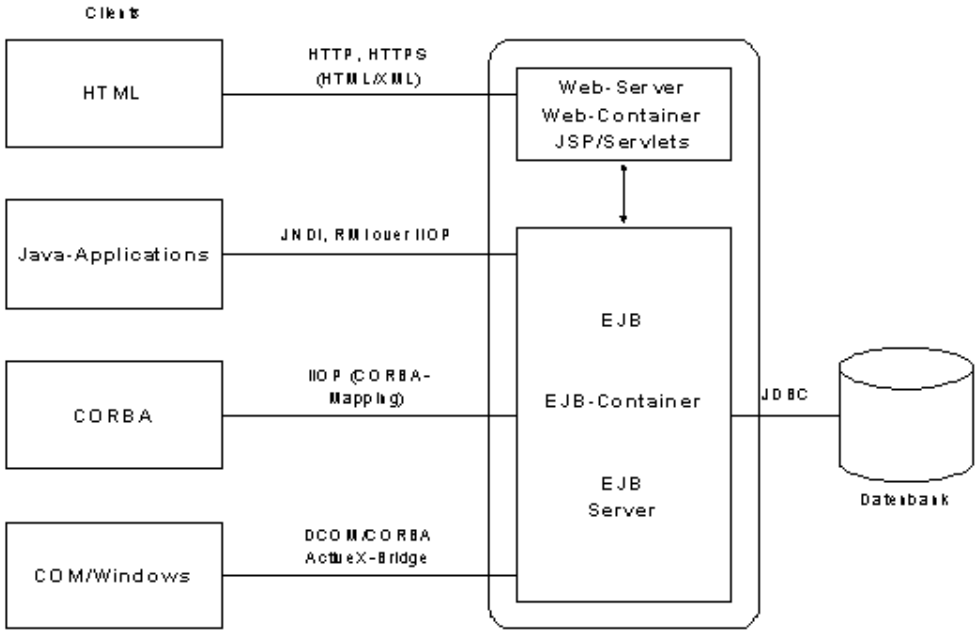
- [Basili 01]: Basili/Boehm: *COTS-Based Systems Top 10 List*. IEEE Computer, May 2001, S. 91-95
- [Blazey 02] Blazey, M.: *Qualitätssicherung für EJB-basierte Softwaresysteme.* Diplomarbeit, Uni Magdeburg, 2002
- [Dumke 97] Dumke, R.; Winkler, A.: *Managing the Component-Based Software Engineering with Metrics.* Proc. of the 5th Int. Symposium on Assessment of Software Tools, Pittsburgh, Juni 1997, S. 104-110
- [Frenzel 99] Frenzel, K.: *Integration einer Standardsoftware ...*, Diplomarbeit, Uni Magdeburg, 1999
- [Jell 98] Jall, T.:*Component-Based Software Engineering.* Cambridge University Press, 1998
- [Poulin 97] Poulin, J.S.: *Measuring Software Reuse.* Addison Wesley Verlag, 1997
- [Schmiet 02a] Schietendorf, A.; Dumke, R.; Dimitrov, E.; Nakonz, S.: *Bewertungsaspekte der komponentenorientierten Softwareentwicklung am Beispiel von Java-Komponenten.* Preprint Nr. 5, Fakultät für Infortmatik, Uni Magdeburg, 2002
- [Sodhi 99] Sodhi, J.; Sodhi, P.: *Software Reuse - Domain Analysis and Design Process.* McGraw Hill Verlag, 1999
- [Takeshita 97] Takeshita, T.: *Metrics and Risks of CBSE.* Proc. of the 5th Int. Symposium on Assessment of Software Tools, Pittsburgh, Juni 1997, S. 91-93
- [Wartmann 98] Wartmann, M.: *Analyse und Abschätzung des Aufwandes bei der Software-Integration.* Diplomarbeit, Uni Magdeburg, 1998
- [Voas 98] Voas, J.: *Maintaining Component-Based Systems.* IEEE Software July 1998, S. 22-27
- [Weyuker 98] Weyuker, E.J.: *Testing Component-Based Software: A Cautionary Tale.* IEEE Software, Sept. 1998, S. 54-59

6. Enterprise JavaBeans

6.1 EJB-Grundlagen

Da wir zum Teil einen Preprint zur Verfügung haben und vor allem unser Fachbuch die Grundlage bildet, können wir uns im Skript auf allgemeine Hinweise bzw. Prinzipskizzen beschränken. Die aktuellen Dokumente können unter dem Link Sun Specification direkt von angesehen werden.

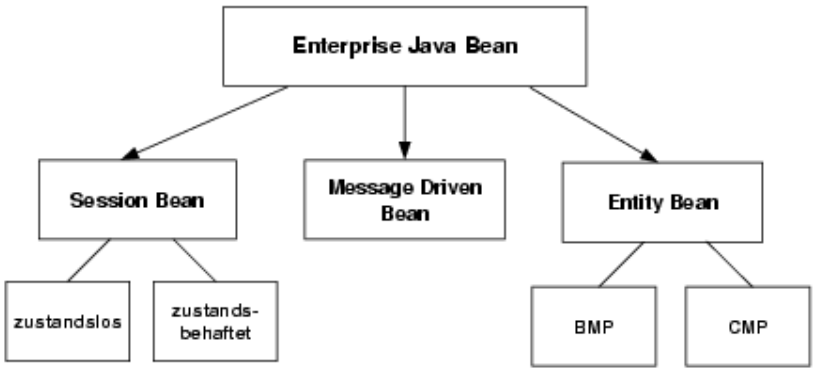
Die erste Abbildung zeigt uns die Möglichkeiten, die wir bei der Bildung EJB-basierte Architekturen im Allgemeinen zur Verfügung haben.



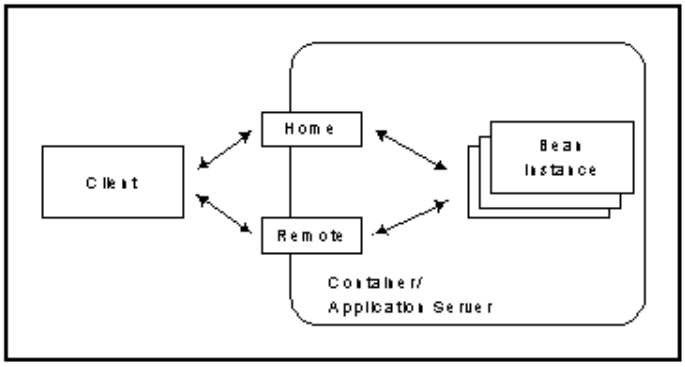
Die EJB-basierte Systementwicklung ist eingebettet in ein Gesamtkonzept von Sun zur Unterstützung der Implementation von Unternehmenssoftware als ***Java 2 Enterprise Edition (J2EE)*** Hierbei können neben den EJB's weitere kompatible Technologien zur Anwendung kommen, wie

- ***JNDI*** : als Java Naming und Directory Interface für dieSchaffung einheitlicher Namens- und Verzeichnisdienste,
- ***JDBC***: als Java Database Connectivity für den Zugruff auf relationale Datenbanken,
- ***JMS*** : als Java Message Service API unter anderem für die asynchrone Kommunikation,
- ***JTA/JTS***: als Java Transaction API für eine transaktionsorientierte Verarbeitungsform,
- ***Java IDL***: als Java Interface Definition Language zur einheitlichen Schnittstellenbeschreibung zwischen den Applikationen,
- ***JavaMail***: als Java Mail API für die einheitliche Verwendung verschiedenster Internet-Mail-Protokolle.

Die Arten von Enterprise JavaBeans-Komponenten zeigt uns die folgende Abbildung.



Hinsichtlich der schnittstellenbezogenen Verbindung müssen wir unser Bild von den JavaBeans her korrigieren. Während die JavaBeans gewissermaßen *Client-Komponenten* darstellen, handelt es sich bei den EJB's um *serverseitige Komponenten*. Dabei wird als Nutzerschnittstelle nicht die *java.awt* verwendet, sondern ein "Home-Remote-Interface"-Konzept. Die folgende Abbildung deutet diese Technologie an.



Insgesamt ist das EJB-Komponentenmodell in folgender Weise allgemein charakterisiert:

- ***Component Pooling und Lifecycle Management*** : die EJB's werden in sogenannten Pools gespeichert und durch den J2EE-Server ge-managed.
- ***Client Session Management***: durch einen Client referenzierte EJB's behalten ihren aktuellen Zustand bei, auch wenn sie durch andere Client verwendet wurden,
- ***Database Connedction Pooling***: die Datanbank-Ressourcen und deren Verbindungen werden bei Bedarf automatisch zugeteilt,
- ***Transaktions Management***: die Methoden der EJB's können unter die Transaktionssicherung des Application-Servers gestellt werden,
- ***Security***: Realisierung der Authentifizierung und Zugriffskontrolle,
- ***Persistence***: Gewährleistung einer dauerhaften Speicherung der jeweiligen Daten durch den Container selbst.

Literaturhinweis:

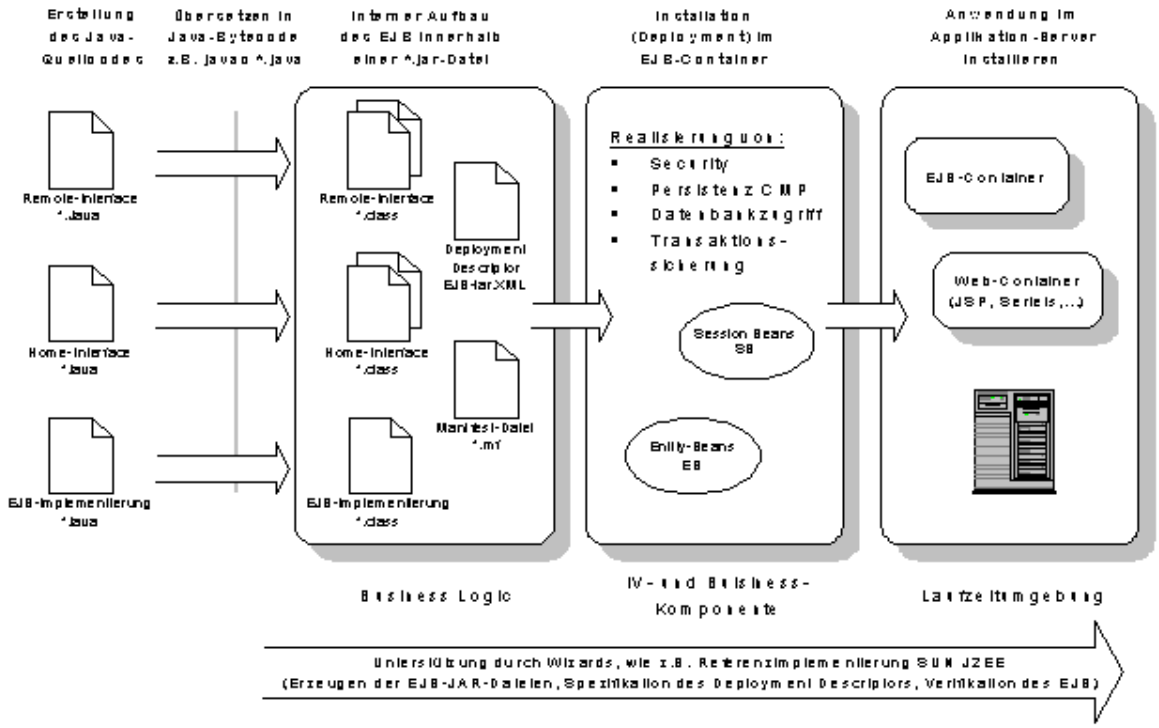
- [Adatia 01] Adatia, R. et al.: *Professional EJB.* Wrox Press, 2001
- [Schmiet 02a] Schietendorf, A.; Dumke, R.; Dimitrov, E.; Nakonz, S.: *Bewertungsaspekte der komponentenorientierten Softwareentwicklung am Beispiel von Java-Komponenten.* Preprint Nr. 5, Fakultät für Infortmatik, Uni Magdeburg, 2002
- [Schmiet 02] Schmietendorf, A.; Dimitrov, E.; Dumke, R.: *Enterprise JavaBeans.* MiTp Verlag, 2002

6.2 EJB-Applikationsentwicklung

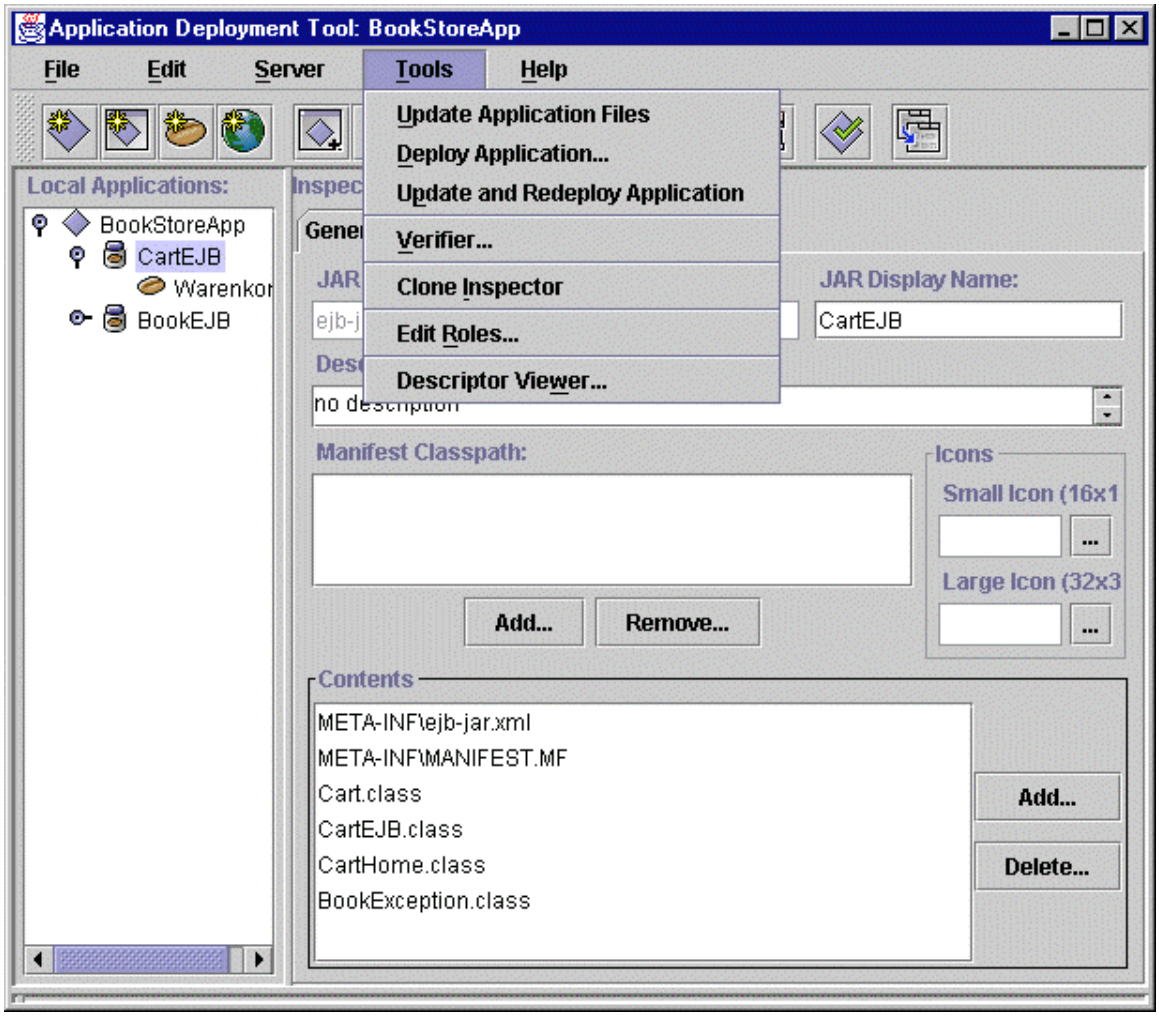
Für die Entwicklung eines EJB's-basierten Softwaresystems ist ein spezielles ***Rollenmodell*** initiiert, welches folgende Entwicklerarten vorsieht:

- ***Server Provider***: entwickelt und stellt bereit den Application Server, der die Laufzeitumgebung des EJB-Containers darstellt,
- ***Container Provider***: entwickelt Laufzeitumgebungen für EJB's - als Container, die relativ unabhängig vom Application Server arbeiten,
- ***Bean Provider***: implementiert die eigentlichen EJB-Komponenten,
- ***Application Assembler***: als sogenannter Domain-Experte "montiert" er die Applikationen aus den jeweiligen Beans-Komponenten,
- ***Bean Deployer***: er installiert die jeweiligen Applikationen mit den jeweils notwendigen Ressourcenanbindungen,
- ***Persistence Manager Provider***: diese erst mit der EJB 2.0 eingeführte Rolle dient dem "Mapping" der Persistenzanforderungen mit den jeweils vorhandenen Datenbasen,
- ***Systemadministrator***: er ist schließlich für die Infrastruktur unserer EJB-Applikation(en) verantwortlich.

Die allgemeine Vorgehensweise bei einer EJB-basierten Systementwicklung zeigt die folgende sehr kompakte Darstellung.



Für die Entwicklung EJB-basierter Applikationen stehen im Allgemeinen Entwicklungsumgebungen zur Verfügung. Das folgende Bild deutet die dabei notwendigen Tool-Funktionen an.



In der Übung werden wir eine derartige Entwicklung an einem Beispiel kennen lernen bzw. die Entwicklungsschritte einmal unmittelbar umsetzen.

Literaturhinweise:

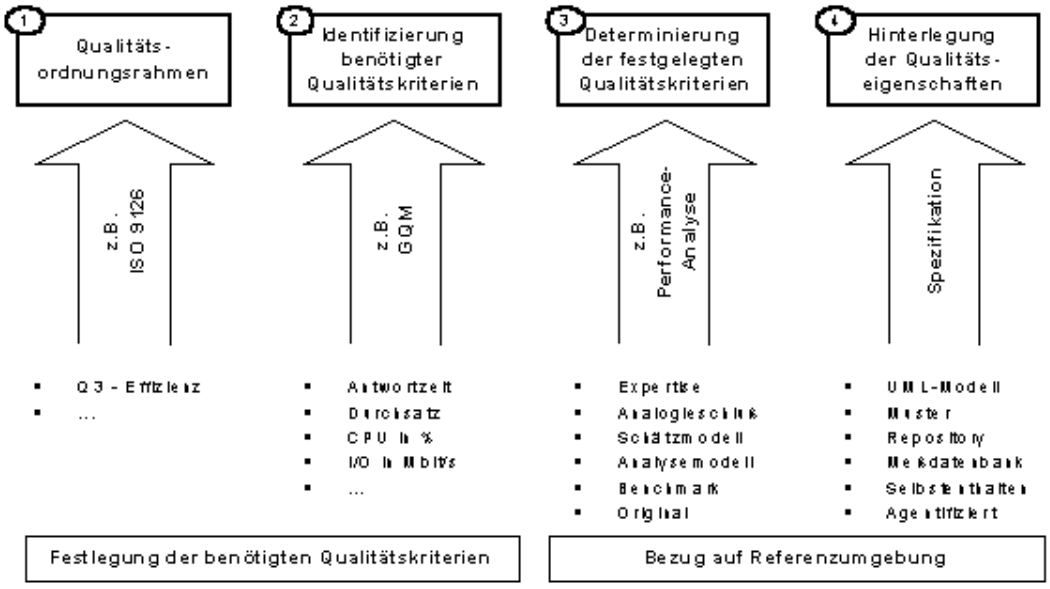
- [Adatia 01] Adatia, R. et al.: *Professional EJB.* Wrox Press, 2001
- [Schmiet 02a] Schietendorf, A.; Dumke, R.; Dimitrov, E.; Nakonz, S.: *Bewertungsaspekte der komponentenorientierten Softwareentwicklung am Beispiel von Java-Komponenten.* Preprint Nr. 5, Fakultät für Infortmatik, Uni Magdeburg, 2002
- [Schmiet 02] Schmietendorf, A.; Dimitrov, E.; Dumke, R.: *Enterprise JavaBeans.* MiTp Verlag, 2002

6.3 Qualitätssicherung

6.3.1 Grundlegende Bemerkungen

In der 6. Lehrhilfe haben wir bereits grundlegende Formen und Methoden einer Qualitätssicherung für Software-Komponenten kennen gelernt.

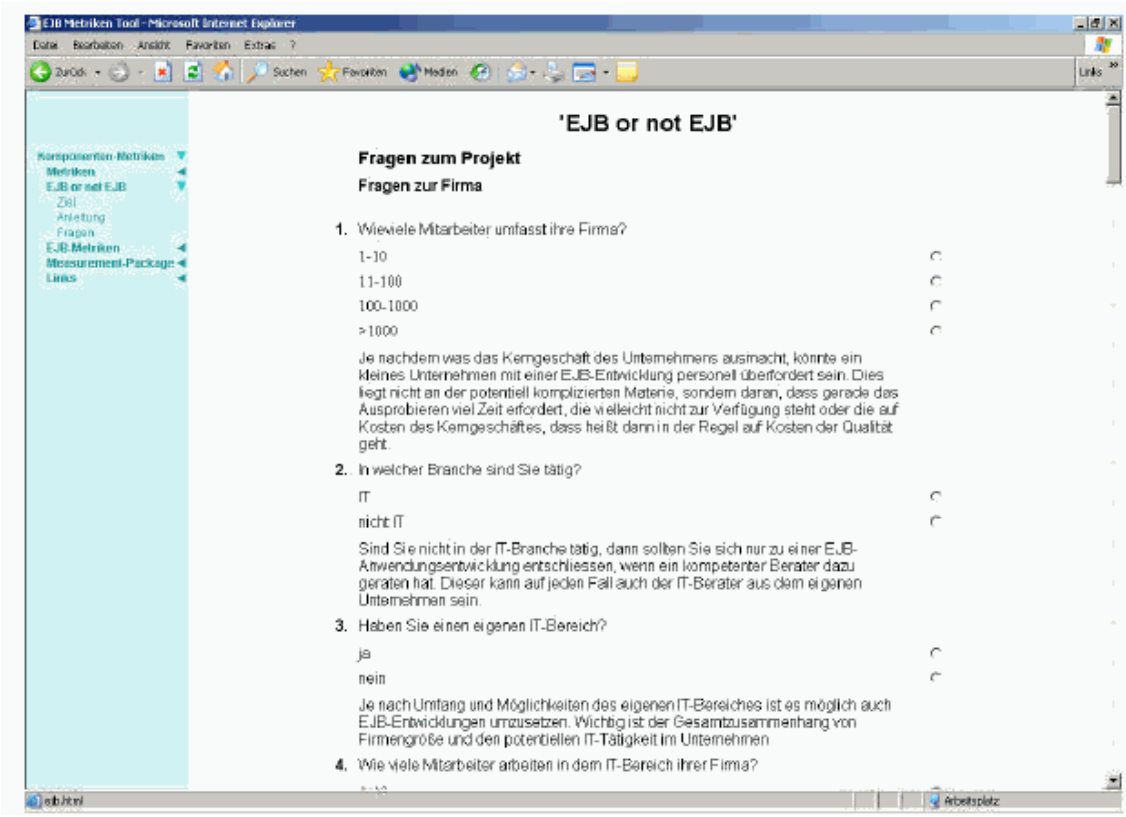
Die allgemeinen Ansatzpunkte und Verfahren für die EJB-Qualitätssicherung fasst die folgende Abbildung zusammen.



Daraus ergeben sich eine Reihe von Fragestellungen bzw. Ansatzpunkte für EJB-Applikationsentwicklung:

6.3.2 Die Qualität der EJB-Entwicklung

Die Bewertung des Entwicklungsprozesses selbst, d. h. die Überprüfung des Sinnfälligkeit der Anwendung der EJB-Technologie, stellt eine wichtige Ausgangsprämisse dar. In [Blazey 02] ist daher ein Anfrage-Tool entwickelt worden, welches sich dieser Problemstellung widmet. Es hat hierbei das folgende Layout



eine mögliche Beantwortung beginnt dann wie folgt aus

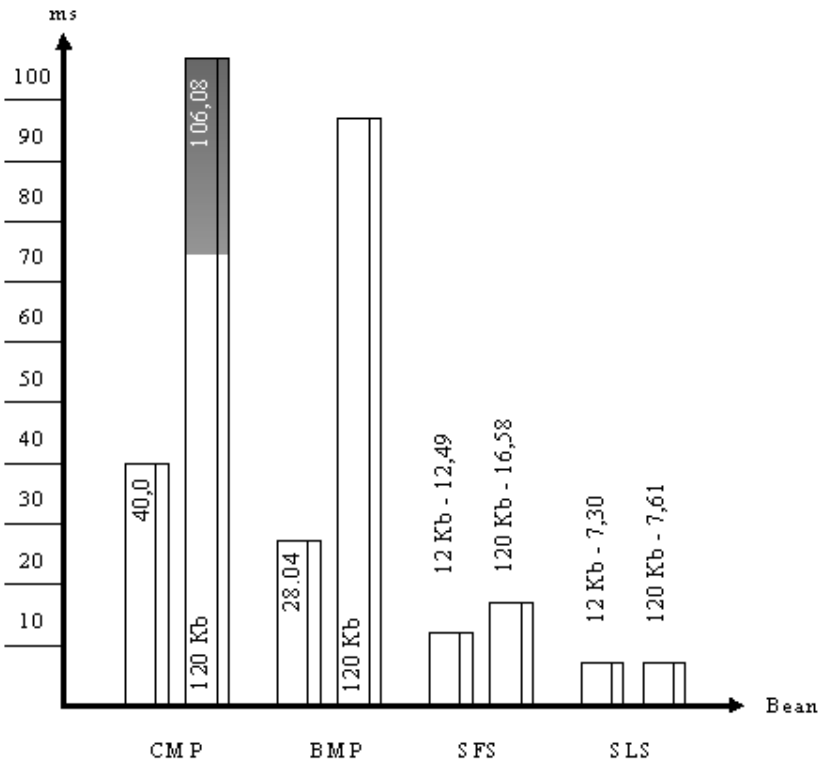
6.3.3 Die Qualität der EJB-Architektur

Die Qualitätsaspekte der Komponenten untereinander bzw. miteinander ist Inhalt dieses Bereiches. Hierbei geht es um solche Fragen, wie die sinnvolle Größe einer Komponenten, die maximale Kopplungsform oder die strukturelle Verknüpfung von Session- und Entity-Beans.

Im folgenden sind einige ausgewählte Ziele für eine ***Performance-Analyse*** dargestellt:

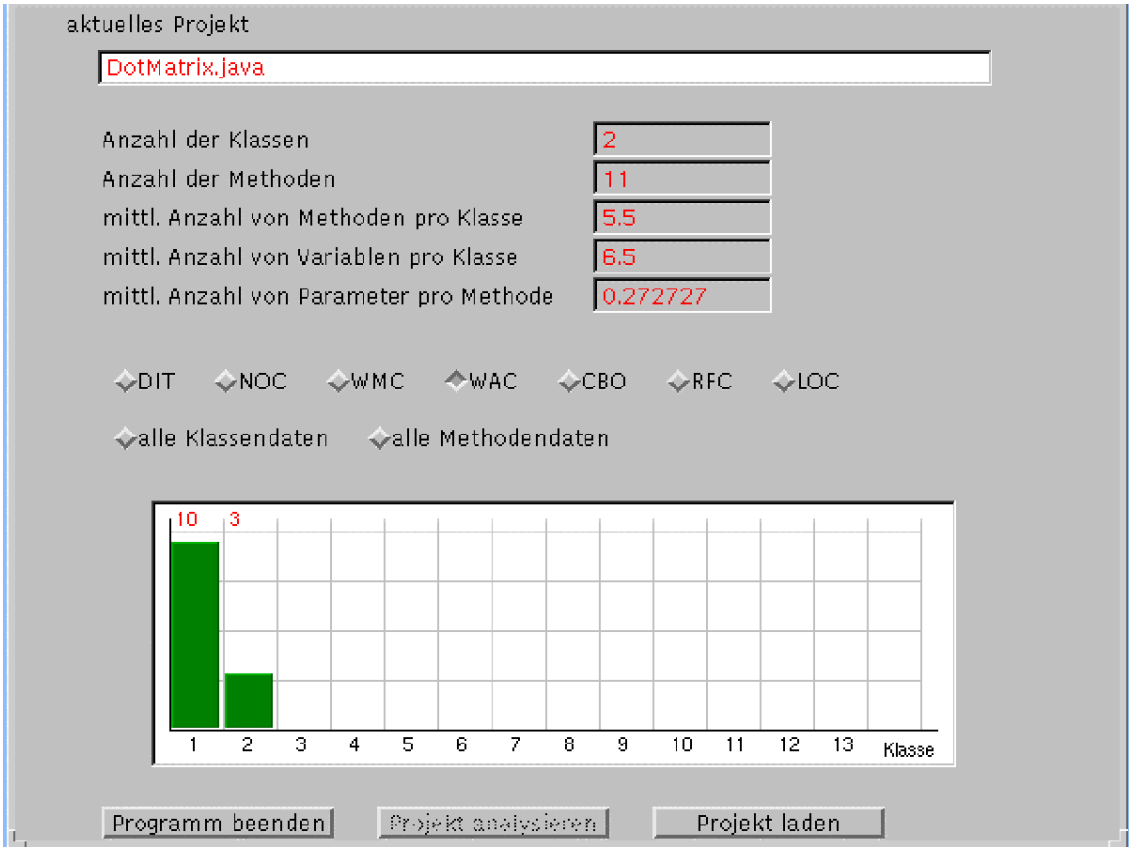
- Wie verhalten sich EJB-Server unter der Last steigender Client-Zugriffe und wie werden dabei die verfügbaren Hard- und Software-Ressourcen ausgenutzt?
- Welche Einflüsse auf die Performance hat die Verwendung der unterschiedlichen EJB-Typen (Entity-Beans, Session-Beans, Message Driven Beans)?
- Welchen Einfluß hat die Lokalität des Methodenaufrufs auf die Performance. Zu Berücksichtigen sind Methoden-Aufrufe im gleichen Container/EJB-Server, zu verschiedenen Containern innerhalb des selben EJB-Servers oder aber zu anderen EJB-Servern.
- Bringt es Vorteile den EJB-Server als Teil eines Datenbankservers zu integrieren, und wenn ja wie können diese quantifiziert werden?
- Wie können Hard- und Software-Systeme für die EJB-Plattform bemessen werden um den Nutzeranforderungen gerecht zu werden?

Im Folgenden geben wir eine konkrete Messung von Initialisierungzeiten an.



6.3.4 Qualitätssicherung des Komponenten-Codes

Bei der Bewertung der Qualität der Komponenten selbst kommen die Qualitätsmaße und -bewertungsformen für den Java-Code überhaupt zur Anwendung. Als visuelles Beispiel deuten wir hier die Java-Messung mit einem entsprechenden Tool an.



Hier sind also allgemeine OO-Metriken anzuwenden und für die Qualitätsbewertung und damit -sicherung anzuwenden.

Literaturhinweise:

- [Adatia 01] Adatia, R. et al.: *Professional EJB.* Wrox Press, 2001
- [Blazey 02] Blazey, M.: *Qualitätssicherung für EJB-basierte Softwaresysteme.* Diplomarbeit, Uni Magdeburg, 2002
- [Schmiet 02a] Schietendorf, A.; Dumke, R.; Dimitrov, E.; Nakonz, S.: *Bewertungsaspekte der komponentenorientierten Softwareentwicklung am Beispiel von Java-Komponenten.* Preprint Nr. 5, Fakultät für Infortmatik, Uni Magdeburg, 2002
- [Schmiet 02] Schmietendorf, A.; Dimitrov, E.; Dumke, R.: *Enterprise JavaBeans.* MiTp Verlag, 2002