

Hase und Igel



Orientation in Objects

- **1840 veröffentlichte Wilhelm Schröder, geboren 1808 in Oldendorf bei Stade, im "Hannoverschen Volksblatt" erstmals das Märchen "Der Wettlauf zwischen dem Hasen und dem Igel auf der kleinen Heide bei Buxtehude" in niederdeutscher Sprache. Er hatte diese Geschichte, deren Ursprung schon im 17. Jahrhundert zu vermuten ist, in Bexhövede (in der Nähe von Bremerhaven), dem Wohnort seines Großvaters, des Amtsvogtes Wilhelm Krone, im Dorfkrug von einem alten Jäger gehört. Aber auch der Pastor im Dorf soll sie ihm erzählt haben. Den Austragungsort des sagenhaften Wettlaufs hat er letztlich jedoch nach Buxtehude verlegt. Die Gründe hierfür sind nicht bekannt. Unter der Nummer 187 wurde die Geschichte 1843 von den Brüdern Jakob und Wilhelm Grimm in die 5. Auflage der Kinder- und Hausmärchen aufgenommen.**

<http://www.fh-hamburg.de/pers/Meyer/buxte/hui-f1.html>



© 2000-2001 Papick Garcia Taboada - Dirk M. Sohn

1

Hase und Igel



Orientation in Objects



- **Zum vierundsiebzigstenmal aber kam der Hase nicht mehr zu Ende. Mitten auf dem Acker stürzte er zur Erde, das Blut floß ihm aus dem Halse, und er blieb tot auf dem Platze. Der Swineigel aber nahm seinen gewonnenen Golddukat und die Flasche Branntwein, rief seine Frau aus der Furche ab, und beide gingen vergnügt nach Hause, und wenn sie nicht gestorben sind, leben sie noch.**
- <http://millenniumcity.magnet.at/~g.nikles@magnet.at/haseigel.htm>



© 2000-2001 Papick Garcia Taboada - Dirk M. Sohn

2

- EJB und CORBA-Komponenten
 - Das „Moving Target“ Problem

Dirk M. Sohn

sohn@oio.de

Papick G. Taboada

taboada@adminSight.de

- „It (CCM) has not been given final approval by the membership of OMG, but in this case, that is considered a formality. The new standard will become official by the end of the year.“
- „I wouldn't expect the CCM standard to exert much influence on the component market next year.“
- „In other words, the OMG has not created another component model that will compete with MTS and EJB in 2000. Instead they have ... created a component solution for the future.“

- Diskussion vor 2 Jahren:

Was ist eine Komponente?



Ein Objekt ?

Eine Klasse?

- Diskussion heute:

Wie mache ich aus einer
Geschäftsidee sehr schnell
eine Anwendung?

IDEE



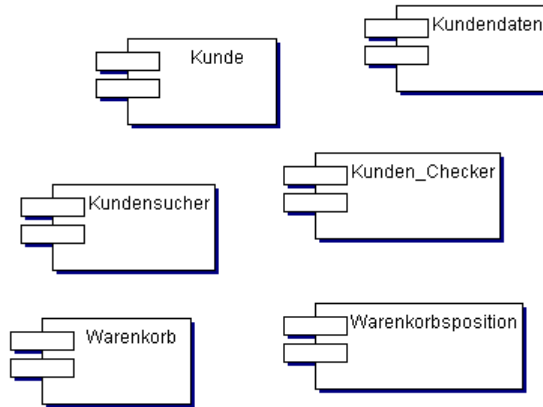
- Anforderungen heute:
 - Anwendung muss sehr schnell implementiert werden
 - Änderungen müssen kostengünstig realisierbar sein
 - Anwendung muss mit Wachstum der Geschäftsidee Schritt halten können
 - Client-Server Architektur wegen Internet zwingend erforderlich
 - u.s.w.
- Ziel
 - Ergebnisse aus den Software-Entwicklungsphasen ineinander ohne Aufwand fließen lassen!
 - Analoge Strukturen nutzen und nicht mehrmals implementieren

- Und welche Rolle spielen Komponentenmodelle?
 - Nehmen uns wiederkehrende Implementierungsarbeit ab
 - Framework für verteilte Anwendungen
 - Werden von Fachbereichsexperten für verteilte Anwendungen erstellt, fundiertes Know-How wird einfach benutzt
 - Praxisfrage
 - **Wer soll Komponenten entwickeln?**
 - Fachbereichsexperten?
 - Serverexperten?

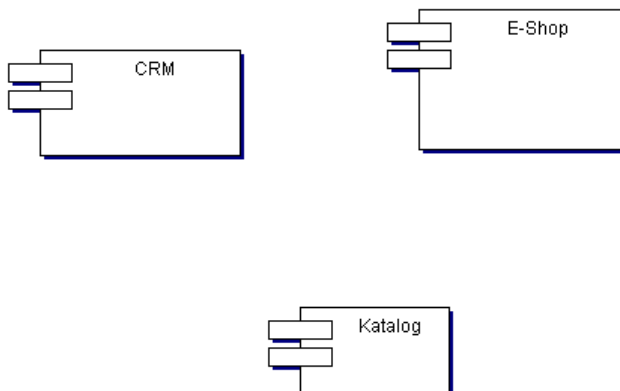
- Einführung (schon vorbei)
- **Komponentengröße**
- Das CORBA Komponentenmodell (CCM)
- Integration von CORBA Komponenten mit EJB 1.1
- Enterprise JavaBeans 2.0
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- Einschätzung des Reifegrades von CCM

- „Komponente“ ist ein dehnbare Begriff
- Für diesen Vortrag: „mittlere Größe“
 - nicht nur „Zeile einer Tabelle“
 - nicht Subsystem eines ERP-Systems

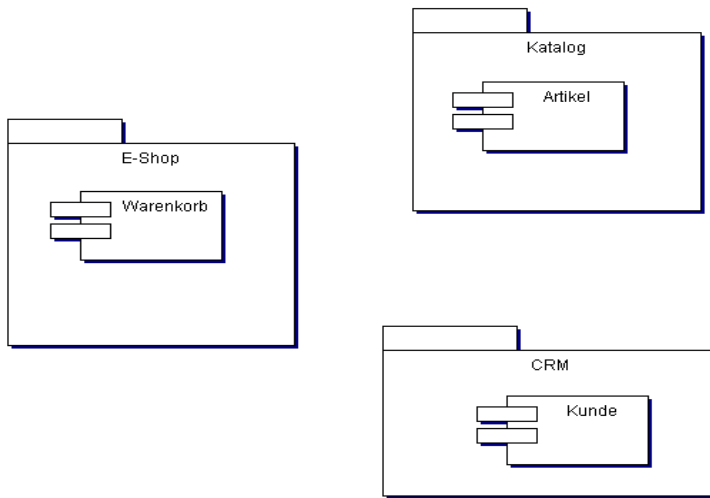
Kleine Komponenten



Große Komponenten



Mittlere Komponenten



Überblick

- Einführung (schon vorbei)
- Komponentengröße
- **Das CORBA Komponentenmodell (CCM)**
- Integration von CORBA Komponenten mit EJB 1.1
- Enterprise JavaBeans 2.0
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- Einschätzung des Reifegrades von CCM

- Das Modell sind 4 Modelle
- Geburt einer CCM-Applikation
- Client-Sicht
- CCM-compliant

- Das abstrakte Modell (AM)
 - Abstract Component Model
 - **Erweiterungen zur IDL und zum Object Model**
- Das Programmierungs-Modell (PM)
 - CCM Implementation Framework (CIDL)
 - Container Programming Model
 - **Sicht für Clients und Komponenten-Entwickler**
- Das Deployment-Modell (DM)
 - Packaging and Deployment Model
- Das Execution-Modell (EM)
 - Component Container Architecture
 - **Sicht für Container-Provider**

Das abstrakte Modell (AM) - Ziele

- Beschreibungsfähigkeit einer Komponente herstellen
 - Component API - was sehen die Clients
 - Abhängigkeiten - was wird von anderen Komponenten benötigt
 - Interaktionsverhalten (synchron, asynchron)
- Introspektion ermöglichen
- Modularität erhöhen

Kaffeemaschine Beispiel-Komponente

Component Facet



Kaffeemaschine



- Neuer IDL-Metatyp:

```
component Kaffeemaschine {  
};
```

- *Equivalent IDL*:

```
interface Kaffeemaschine : Components::CCMObject {  
};
```

- Neuer IDL Meta-Typ *component*
- Eine Komponente bietet ports an, das sind:
 - Facets, auch *provided Interfaces* genannt
 - Receptacles
 - Event Sources und Event Sinks
 - Attributes (Konfiguration)
- Client-Mappings (externe Features) werden in *equivalent IDL* beschrieben.

- Problem: Klassenobjekte
 - Für normales OOP üblich
 - Bei verteiltem CORBA: wer kümmert sich darum
 - **wo Instanzen erzeugt werden**
 - **wo sie sind**
 - Beispiel: Beim Erzeugen einer Instanz soll ein Primärschlüssel mit der Instanz assoziiert werden -> `find_by_primary_key` wird ermöglicht (mini-Naming-Service für Primärschlüssel).
- Idee:
 - Factory, die alle Instanzen im Scope der `CCM_Runtime` verwaltet

- Lösung: Neuer IDL-Metatyp *ComponentHome*.
 - Standardoperationen für den Typ der Komponente:
 - create
 - finder
 - destruct
 - Vergleich
 - selbstdefinierte (z.B. queries)
 - Jeder Komponententyp muß mindestens einen Hometyp haben, kann aber mehrere haben (z.B. unterschiedliche Primärschlüssel)
- => Zum ersten Mal kann CORBA Objekt-Identität sichtbar für den Client mitverwalten.

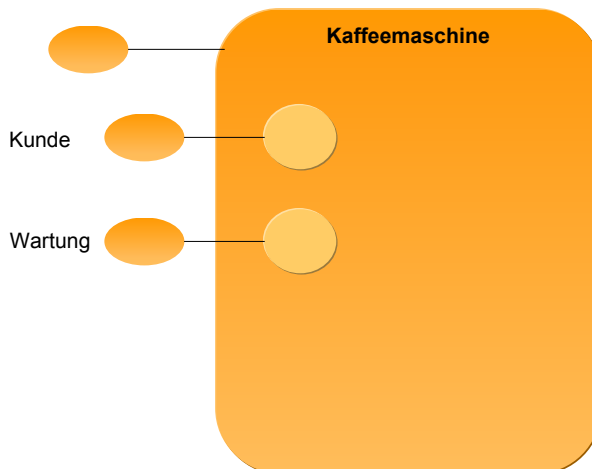
- Neuer IDL-Metatyp:

```
home KaffeemaschineHome manages Kaffeemaschine {  
    <explicit_operations>  
};
```

- *Equivalent IDL:*

```
interface KaffeemaschineHomeExplicit: Components::CCMHome {  
    <explicit_operations>  
};  
interface KaffeemaschineHomeImplicit: Components::KeylessCCMHome {  
    Kaffeemaschinecreate();  
};  
  
interface KaffeemaschineHome: KaffeemaschineHomeExplicit,  
    KaffeemaschineHomeImplicit {};
```

Component Facet



```
component Kaffeemaschine {  
  
    provides Kunde kunde;  
    provides Wartung wartung;  
};  
  
interface Kaffeemaschine : Components::CCMObject {  
  
    Kunde        provide_kunde();  
    Wartung      provide_wartung();  
};  
  
interface Kunde {...};  
interface Wartung {...};
```

- Facets (Facette, Aspekt) sind Objektreferenzen einer CC.
- Mehrere Facets pro CC möglich.
- Diese müssen nicht voneinander erben.
- Eine ausgezeichnete OR verweist auf das von der Komponente selbst definierte Interface
- *Equivalent Interface* zur Navigation über Facets und Ports

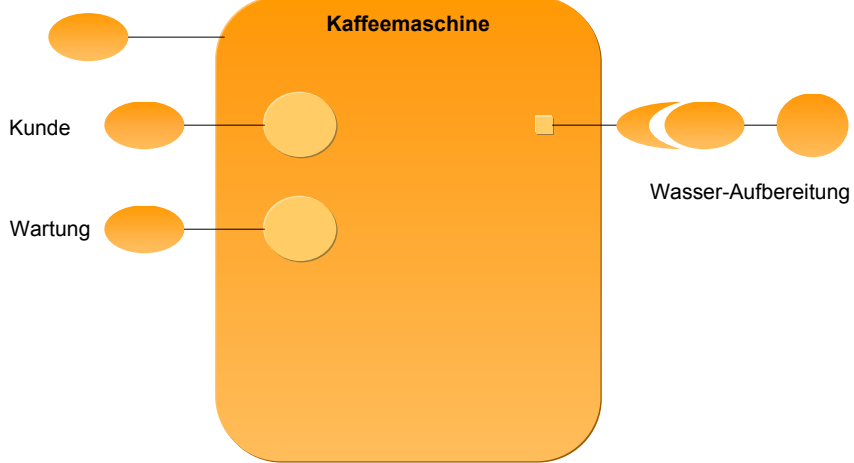
- Wegen Components::CCMObject : Components::Navigation

```
interface Kaffeemaschine : Components::CCMObject {  
  
    Kunde                provide_kunde();  
    Wartung              provide_wartung();  
  
    provide_facet (in FeatureName name) // von Navigation  
    Facets provide_all_facets();  
  
    boolean same_component (in Object ref)  
  
    ...  
};
```

- Navigation von jeder Facet zur Component Facet:
 - mit get_component() von CORBA::Object
- Navigation von der Component reference zu jeder Facet
 - mit generierten facet-spezifischen Methoden
 - provide_<facet-name>
 - mit generischen Navigations-Operationen auch dynamische Navigation und Vergleich von Komponenten möglich.

Kaffeemaschine - Receptacles

Component Facet



Kaffeemaschine - Receptacles

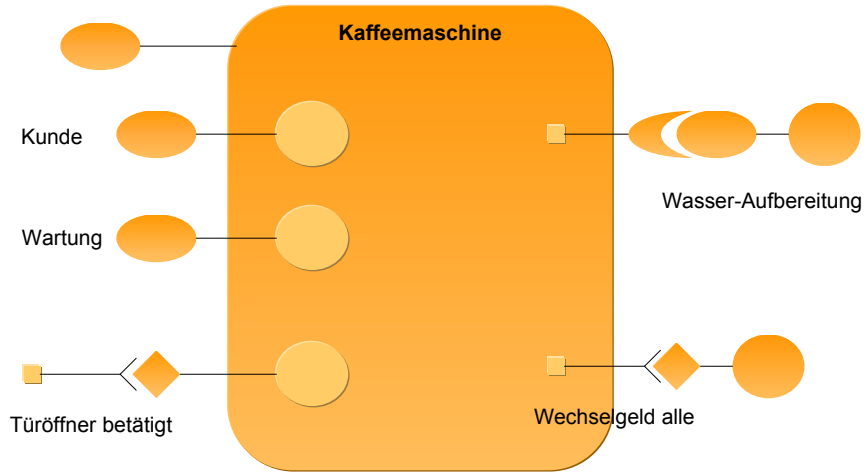
```
component Kaffeemaschine {  
  
    provides Kunde kunde;  
    provides Wartung wartung;  
  
    uses WasserAufbereitung wasser;  
};  
  
interface Wasseraufbereitung {...};
```

```
interface Kaffeemaschine : Components::CCMObject {  
  
    Kunde        provide_kunde();  
    Wartung      provide_wartung();  
  
    provide_facet (in FeatureName name) // von Navigation  
    Facets provide_all_facets();  
    boolean same_component (in Object ref)  
  
    WasserAufbereitung get_connection_wasser();  
  
    void connect_wasser(in WasserAufbereitung wa) rais..;  
  
    WasserAufbereitung disconnect_wasser() raises(...);  
};
```

- Möglichkeit zum deskriptiven Management von Komposition und Delegation
- Mögliche Nutzung für
 - Konfiguration zur Bildung eines assembly
 - zur Laufzeit

Kaffeemaschine - Events

Component Facet



Kaffeemaschine - Events

```
component Kaffeemaschine {  
  
    provides Kunde kunde;  
    provides Wartung wartung;  
  
    uses WasserAufbereitung wasser;  
  
    publishes GeldEvt geld;  
    consumes TuerEvt tuer;  
};
```

Kaffeemaschine - Events

```
valuetype GeldEvt : Components::EventBase{..};
valuetype TuerEvt : Components::EventBase{..};

module KaffeemaschineEventConsumers {
  interface GeldEvtConsumer : Component::EventConsumerBase {
    void push (in GeldEvt ga_evt);
  };

  interface TuerEvtConsumer : Component::EventConsumerBase {
    void push (in TuerEvt to_evt);
  };
};
```

Kaffeemaschine - Events

```
interface Kaffeemaschine : Components::CCMObject {

  Kunde provide_kunde();
  Wartung provide_wartung();

  provide_facet (in FeatureName name) // von Navigation
  Facets provide_all_facets();
  boolean same_component (in Object ref)

  void connect_wasser(in WasserAufbereitung wa) rais..;
  WasserAufbereitung disconnect_wasser() raises(...);
  WasserAufbereitung get_connection_wasser();

  Components::Cookie subscribe_geld
    (in KaffeemaschineEventConsumer::GeldEvtConsumer co) raises(...);

  KaffeemaschineEventConsumer::GeldEvtConsumer
  unsubscribe_geld (in Components::Cookie ck) raises (...);

  KaffeemaschineEventConsumer::TuerEvtConsumer
  get_consumer_tuer();
};
```

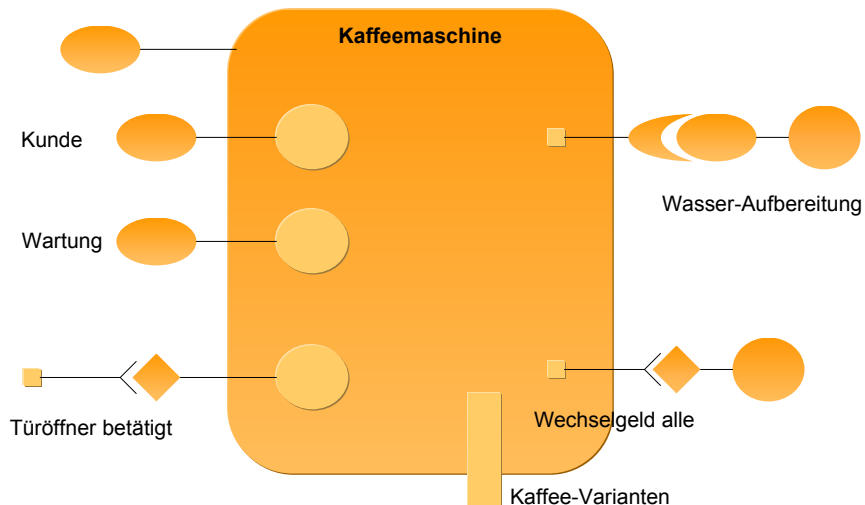
- Teilmenge des CORBA Notification Service
- Publish / Subscribe Modell
- Nur Push-Mode

- Event-Quellen:
 - publishes: Spezifischer Pfad zwischen Event-Quelle und beliebigen Empfängern
 - emits: Benutzung eines existierenden Kanals für Broadcasting

- Event-Senken
 - Registrierung (auch mehrfach) bei Event-Quellen

Kaffeemaschine Attribute

Component Facet



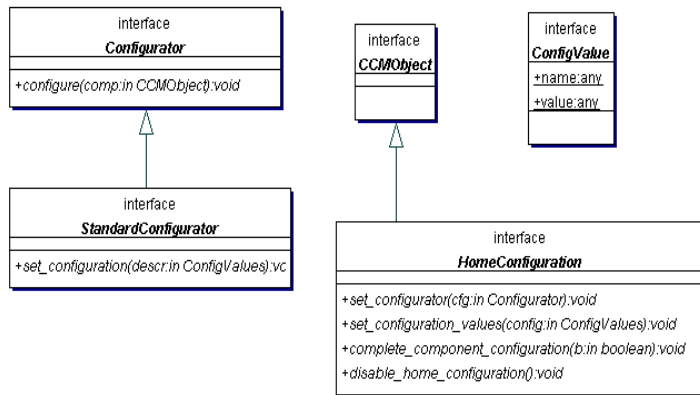
Kaffeemaschine - Attribute

```
• component Kaffeemaschine {  
  
    provides Kunde kunde;  
    provides Wartung wartung;  
  
    uses WasserAufbereitung wasser;  
  
    publishes GeldEvt geld;  
    consumes TuerEvt tuer;  
  
    attribute KaffeeVarianten products;  
  
};
```

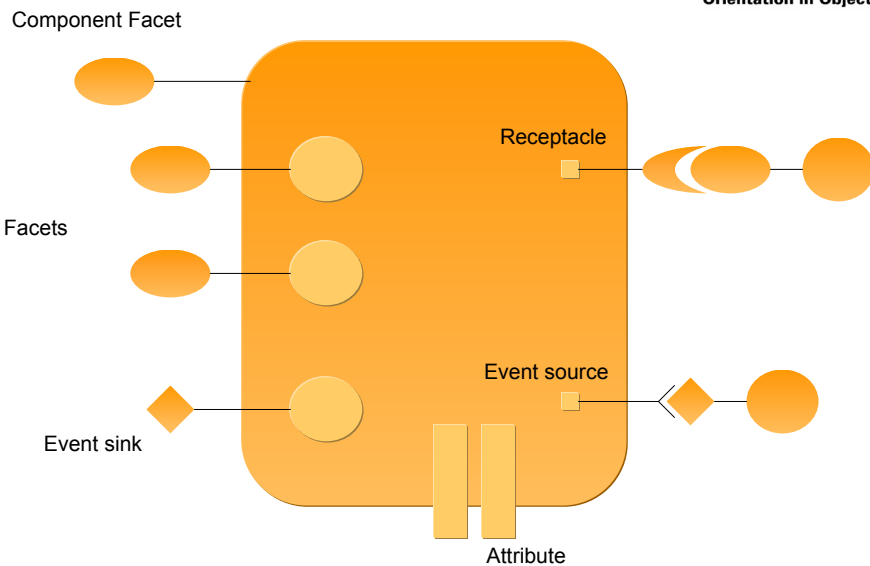
AM - Attribute

- Können Exceptions werfen
 - Verwendungszweck: Konfiguration
 - Standardkonfiguration durch Fabrik möglich
 - Konfigurations-Objekte:
 - **definiert je CC-Typ**
 - **erben von Components::Configurator**
 - **enthalten Konfigurations-Prozess und Attributwerte**
 - Konfigurationsablauf:
 - **Konfigurations-Objekt an Home übergeben**
 - **configure(CCMObject)**
 - **nach Abschluss Aufruf von configuration_complete()**
- => Die Konfiguration ist fest definierter Lebensabschnitt einer CC

AM - Konfiguration von Komponenten



AM - Komponente



- 2 Levels von Komponenten:
 - basic
 - „Komponentisierung“ bestehender CORBA-Objekte
 - das Äquivalent zur EJB 1.1-Funktionalität
 - keine Facets, Receptacles, Event-Quellen und -Senken also von den Ports nur Attribute
 - vermutlich früher verfügbar
 - extended
 - wesentlich mehr Möglichkeiten
 - vermutlich länger noch nicht verfügbar
- Levels nicht in der Sprache verankert wegen vermuteter Vorläufigkeit der Levels

- Definition der Implementierung (CIF + CIDL)
 - CIF - Component Implementation Framework definiert das Zusammenspiel von funktionalen und nicht-funktionalen Bestandteilen
 - CIDL - Component Interface Definition Language Deklarationssprache zur Beschreibung von Komponenten
 - Abstrakte Definition von Persistenz (PSS)
- Definition der Component-Container-Interaktion
 - Container API
 - Component Callback API
- Definition der Interaktion mit einer Komponente
 - Mapping von Extended OMG IDL auf bisheriges OMG IDL

PM - Kategorien von Komponenten

	Service	Session / EJB	Entity / EJB	Process
CORBA Usage Model	Stateless	Conversational	Durable	Durable
Container API Type	Session	Session / EJB	Entity / EJB	Entity
Primary Key	Nein	Nein	Ja	Nein

Kaffeemaschine - CIDL

```
composition entity Flur_Kaffeemaschine {  
  
    home executor KaffeemaschineHomeImpl {  
  
        implements KaffeemaschineHome;  
        manages KaffeemaschineImpl;  
    };  
};
```

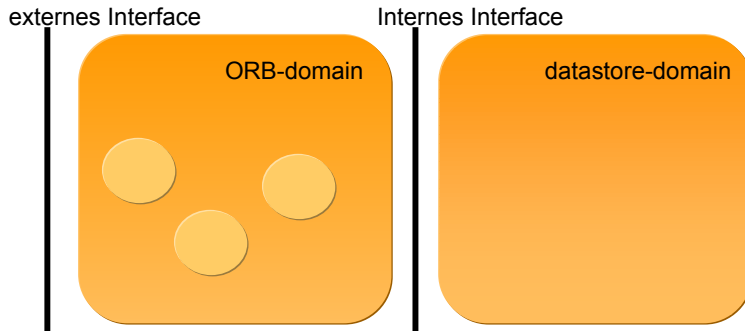
- Gesteuert mittels Policies im XML-Format
- Größtenteils durch Container implementiert, nicht durch die Komponente
- Policies verfügbar zur Steuerung von:
 - Servant Lebenszeit
 - Transaktionen
 - Sicherheit
 - Events
 - Persistenz
 - ...

- Für den Entity-, Process und EJBEntity-Container
- Zwei Varianten:
 - self managed
 - container managed
- Jeder Modus kann Persistent State Service (PSS) oder einen eigenen Persistenz-Mechanismus benutzen (solange dieser das PSS-Framework unterstützt ;-))
 - CORBA-Komponenten mit CMP benötigen nur ein eingeschränktes PSS-API
 - => Container braucht für CMP keine vollständige PSS-Implement.

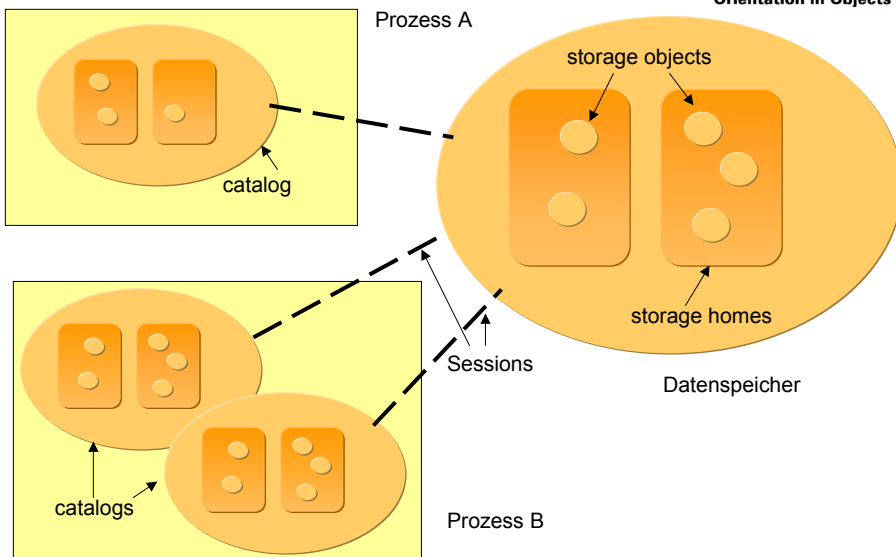
Exkurs - Persistent State Service (PSS) 2.0

Spezifikation der Interfaces für die Implementierung persistenter Objekte

beschreibt Schnittstelle von CORBA-Servants mit Datenspeichern, berührt nicht die Client-Schnittstelle der Servants



PSS



- Sprache zur Definition persistenter Datentypen
- Als Feature ist die direkte Definition in Java und C++ beschrieben - sogenannte „transparente Persistence“
- PSS-Anbieter muß den PSDL-Compiler stellen
- 2 Arten von PSDL-Compilern
 - „standalone“ Compiler erstellen nur Implementierung der erweiterten Persistencetypen und nutzen ansonsten IDL-Compiler des ORB-Anbieters
 - ORB-Anbieter erweitert IDL-Compiler so, daß er PSDL übersetzt

```
abstract storage-type KMState {  
    float    getFuellstand();  
    long     getBecherzahl();  
};  
  
storage-type PortableKMState implements KMState {  
    state float Fuellstand;  
    state long Becherzahl;  
};  
  
abstract storage-home KMStateHome {};  
  
catalog KMCatalog {  
    provides KMStateHome;  
};
```

```
S.125 Buch wahrscheinlich ausblenden oder Spec lesen.  
composition entity KaffeemaschineImpl {  
  
    uses catalog {KMCatalog  
    implements Kaffeemaschine;  
  
    home executor KaffeemaschineHomeImpl;  
  
};
```

abstract storagetype (deklariert Status und Verhalten)

```
abstract storagetype XYZ{  
    Object resolve(in AAA x) raises ...  
}
```

abstract storagehome (deklariert nur Verhalten)

```
abstract storagehome XYZHome of XYZ{  
    factory create();  
}
```

storagetype

```
storagetype ABC implements XYZ {  
    state List m_list  
}
```

- Spezifizierbar mit dem deployment-Deskriptor
- Container bietet API zum Zugriff auf *Security Current*

- Modus Container-managed:
 - Gesteuert mit dem Komponenten-Deskriptor
 - NOT_SUPPORTED
 - REQUIRED
 - SUPPORTS
 - REQUIRES_NEW
 - MANDATORY
 - NEVER
- Modus Komponenten-managed
 - mit dem UserTransaction API von Components::Transaction
- Nur flache Transaktionen

Deployment-Modell (DM) - Ziele

- Ziele:
 - Automatisches Deployment
 - Klärung der Paketierung heterogener Komponenten
 - Interoperabilität zwischen Deployment Tools und Containern
- Sprache
 - XML DTDs
 - softpkg angelehnt an Open Software Description (OSD, W3C, XML)

DM - Deskriptoren

- Component Descriptor
 - welche Features (Typ, Interfaces, Ports, Dienste, Segmentierung)
- Software Package Descriptor
 - Informationen über Implementierungen
 - welche System-Anforderungen (OS, ORB, JVM, Bibliotheken,...)
- Properties
 - beschreibt die initiale Konfiguration für eine Instanz
- Assembly
 - Menge von zusammengehörigen (uses/provides, emits/consumes) Component-Packages zusammengefaßt zu einer Deployment-Einheit
 - beim Deployment werden virtuelle Hosts auf physikalische abgebildet.

Kaffeemaschine - Komp. Package Descr.

```
<corbacomponent>
  <corbaversion> 3.0 </corbaversion>
  <componentrepid repid=„IDL:Kaffeemaschine:1.4“ />
  <homerepid repid=„IDL:KaffeemaschineHome:1.2“ />
  <componentkind>
    <entity>
      <servant lifetime=„method“ />
    </entity>
  </componentkind>
  <configurationscomplete set=„true“ />
  ...
</corbacomponent>
```

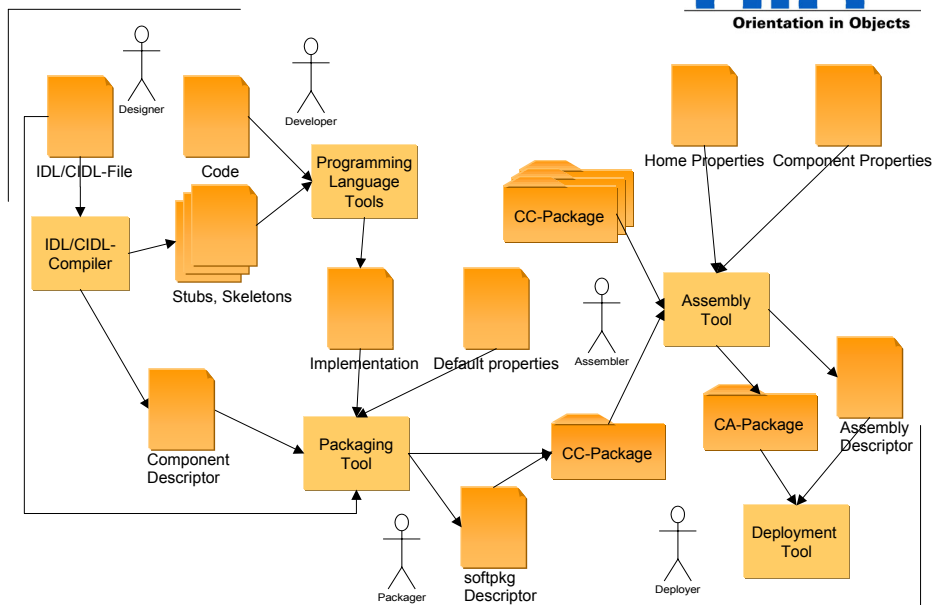
Kaffeemaschine - Softpackage Descr.

```
<softpackage name=„Kaffeemaschine“ version 1.1.4>
  <pkgtype> CORBAComponent </pkgtype>
  <title,..author,..description,..>
  <idl id=„IDL:Kaffeemaschine:1.4“>
    <link href=„http://www.oio.de/maschinenpark/kaffeemaschine.idl“/>
  </idl>
  <implementation id=„GC:0576554808“>
    <os name=„WIN98“/>
    <processor name=„x86“/>
    <compiler name=„jdk1.4.2.“/>
    <programminglanguage name=„Java“/>
    <code type=„JAR“>
      <fileinarchive name=„kaffeemaschine.jar“/>
    </code>
    ....
  </implementation>
  <implementation> <!-- another --> </implementation>
</softpkg>
```

DM - Geburt einer CCM-Applikation

- Spezifikation der Komponenten
- Implementierung der Komponenten
- Paketierung
- Assemblierung
- Auslieferung

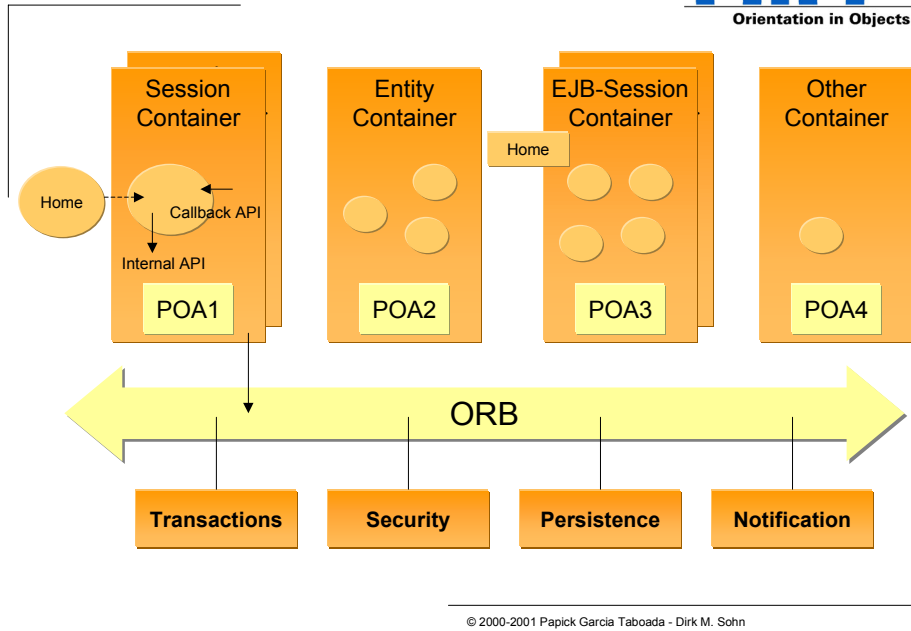
DM - Geburtshelfer



- Definition einer Ablaufumgebung für CC, besteht aus den notwendigen Containern + dem Component Server (der Prozeß für die Container)
 - Container-Manager=Container-Factory
- Unabhängigkeit des Container-Providers:
 - Ein CC-Container kann von einem ORB-Hersteller stammen, oder von einem Anbieter ohne eigenen ORB, der sich auf einen CORBA 3.0 ORB stützt

- Jeder Container hat sein Container-API, das die Interaktionen mit der Komponente sowie POA, ORB und CORBA-Services beschreibt.
- Es ist nicht Bestandteil der Spezifikation, wirklich verschiedene Container für die verschiedenen CC-Typen anzubieten, dies ist eine Design-Entscheidung des Container-Providers.

EM - Container und Server Architektur



EM - Container-Typen

- Service
- Session
- Process
- Entity
- EJBSession
- EJBEntity
- Empty

- method
 - aktiviert beim ersten Aufruf, passiviert nach Beendigung
- transaction (alle Container-Typen außer Service)
 - aktiviert beim ersten Aufruf, passiviert nach Ende der Transaktion
- component (alle Container-Typen außer Service)
 - aktiviert beim ersten Aufruf, explizit passiviert
- container (alle Container-Typen außer Service)
 - aktiviert beim ersten Aufruf, passiviert durch Container

- Home finden:
 - *resolve_initial_references* für einen HomeFinder oder
 - mit Namen bei Naming Service registrieren (wie bei EJB)
 - liefert *ComponentHome*
- Anlegen (Factory Pattern)
 - create aufrufen
- Entities Finden (Finder Pattern)
 - *find_by_primary_key*
 - oder mit Naming- oder Trading-Service
- Navigation, Gleichheit, Unterscheidung *Component<->Object*
- Transactions, Security, Events wie CCM
- Keine direkte Persistenz

- wird anfänglich ein häufiger Fall sein
- nutzt Komponenten über zugrundeliegende CORBA-Interfaces
- können nur supported interfaces
- keine Homes, keine HomeFinder, keine Navigation, ...
- keine Primärschlüssel -> finden Entities nur über Naming Service

- Ausweg: CCM-spezifische Anteile auf Server mit Wrapper

- CORBA ORB-Lieferant
 - Muß keine Komponenten anbieten, muß die Änderungen im Core für Komponentensupport einbauen

- CCM-Lieferant
 - Muß Basic-Level-Komponenten anbieten, kann Extended-Level anbieten
 - EJB-Clients auf CORBA-Komponenten optional

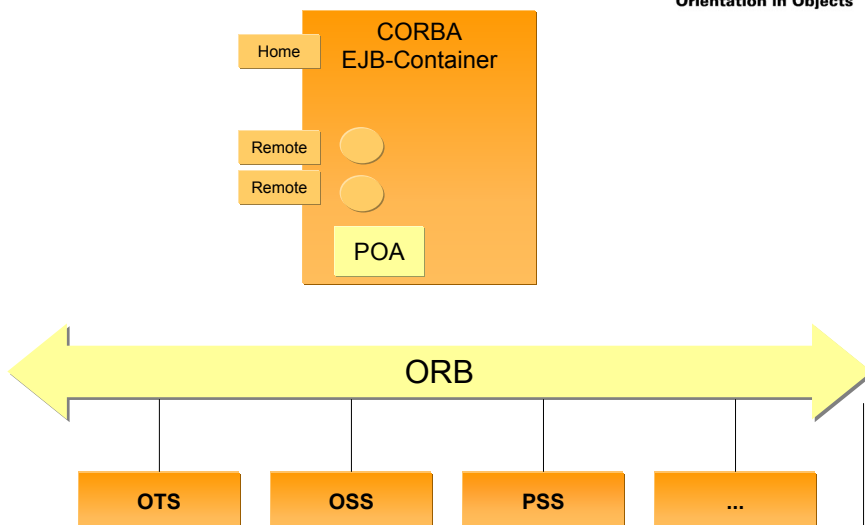
- CORBA COS-Lieferant
 - Änderungen in Lifecycle, Transaction und Security Services

- Basic-Level für Java heißt:
 - EJB 1.1 (inklusive EJB 1.1 XML DTD)
 - Java-to-IDL-Mapping (RMI/IIOP)
 - EJB to IDL mapping
 - Interoperabilität mit IIOP, CORBA Transactions und -Security
- Basic-Level sonst heißt:
 - IDL Erweiterungen fürs Komponentenmodell
 - CIDL (ohne multiple segments)
 - Container für Basic-Komponents
 - XML deployment Deskriptoren und ZIP-Files

- Einführung (schon vorbei)
- Komponentengröße
- Das CORBA Komponentenmodell (CCM)
- **Integration von CORBA Komponenten mit EJB 1.1**
- Enterprise JavaBeans 2.0
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- Einschätzung des Reifegrades von CCM

- Das Java language mapping der CCM APIs sind die EJB APIs
- Wird eine EJB in einen CC-Container deployed, ist sie eine CC
- Ein CORBA-Client sieht eine EJB als CC
- Ein EJB-Client sieht eine Basic-CC als EJB

Ein CORBA EJB-Container



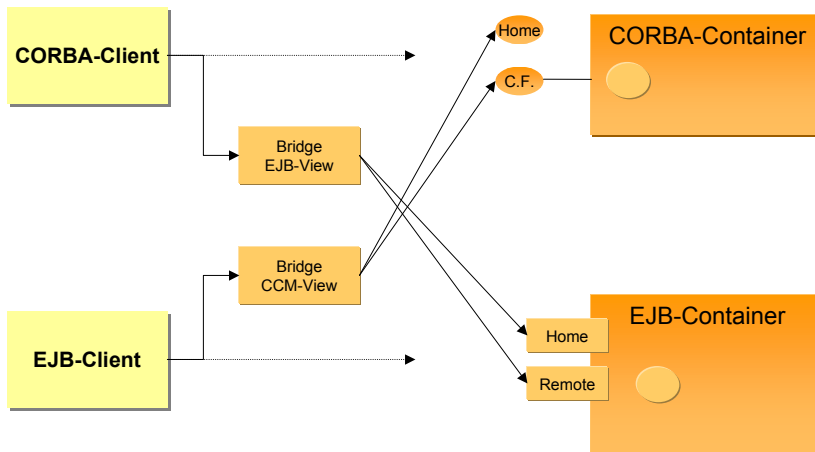
- Beispiel:
 - Unterschiede in der Implementierung eines CC-Session Containers und eines EJBSession Containers in der Component Container Architecture
- Studienobjekte:
 - Erzeugung von Objekt-Referenzen
 - Servant-Lifetime

- Erzeugung von Objekt-Referenzen (CC-Session-Container)
 - Bei Basic-CCs werden die ObjectIDs vom Container verwaltet.
 - Extended-CCs erzeugen Ihre eigenen Referenzen mit dem Container-API. Eindeutigkeit wird vom Container gewährleistet.
- Erzeugung von Objekt-Referenzen (EJBSession-Container)
 - EJB-Clients sehen nur EJBHome und EJBObject Sie werden vom Container erzeugt. Der Container-Provider muß:
 - **Interface-Definitionen für EJBHome und EJBObject erzeugen und sie im IR speichern**
 - **Einträge im Naming Service für die symbolischen Namen der Instanzen von EJBHome und EJBObject erzeugen**
 - **Implementierungen für EJBHome (Delegierung der create-Methoden ans Bean) und EJBObject (Delegierung der Geschäfts-Methoden ans Bean) erzeugen.**

- Servant-Lifetime (SessionCC-Container)
 - Der SessionCC-Container unterstützt alle Servant Lifetime Policies:
 - Method
 - Transaction
 - Component
 - Container

- Servant-Lifetime (EJBSession-Container)
 - EJBs verlassen sich auf die Java-GC für Servant-Lifetime. Das Äquivalent bei der Servant Lifetime Policy ist Container.

Interworking Architecture



- Einführung (schon vorbei)
- Komponentengröße
- Das CORBA Komponentenmodell (CCM)
- Integration von CORBA Komponenten mit EJB 1.1
- **Enterprise JavaBeans 2.0**
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- Einschätzung des Reifegrades von CCM

- J2EE Produkt Provider
- Application Content Provider
- Application Assembler
- Deployer
- System Administrator
- Tool provider

Spieler und Spielregeln

Client View Contract

- home interface
 - Erstellen, finden und entfernen von Beans
- remote interface
 - Business-Methoden aufrufen
- object identity
 - jede Bean hat in Ihrem "home" einen eindeutigen Schlüssel
- handle
 - ein langlebiges "handle" auf die entsprechende Bean
- meta-data
 - Laufzeitinformationen zur Bean (für dynamische Methodenaufrufe)

Bean Provider Contract

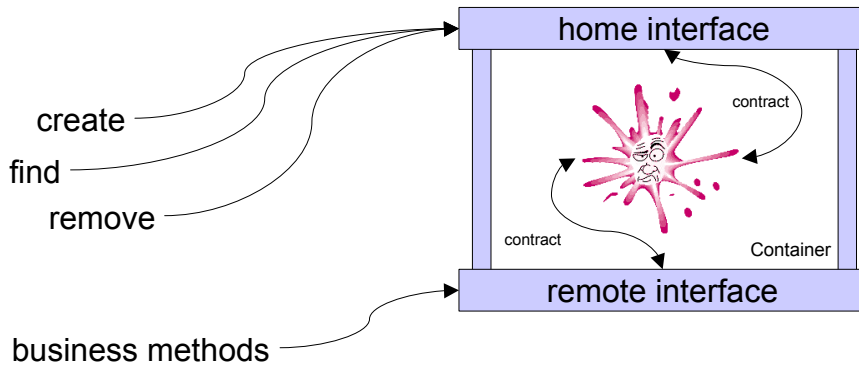
- Business-Schnittstelle implementieren (Remote Interface)
- Allgemeine Methoden implementieren
 - Home Interface
 - Container Callback Schnittstellen
- Beanspezifische Methoden
 - Message Driven Bean Methode "onMessage"
 - Entity Bean muss Persistenzvertrag implementieren
- Transaktionsmethoden bei Bean Managed Transaktion

Container Provider Contract

- Home Interface Methoden vermitteln...
- Business Methoden vermitteln
- Transaktionskontexte berücksichtigen
- Sicherheitsregeln berücksichtigen
- Instanzen erzeugen und verwalten (Poolmanagement, Lifecycle)
- Nachrichten vermitteln

EJB Komponentenmodell

- Eine bean wird niemals direkt angesprochen:
 - home interface
 - remote interface



EJB Komponententypen

- EJB 2.0 definiert 3 verschiedene Komponenten:
 - Session Beans
 - Entity Beans
 - Message Driven Beans (neu in EJB 2.0)

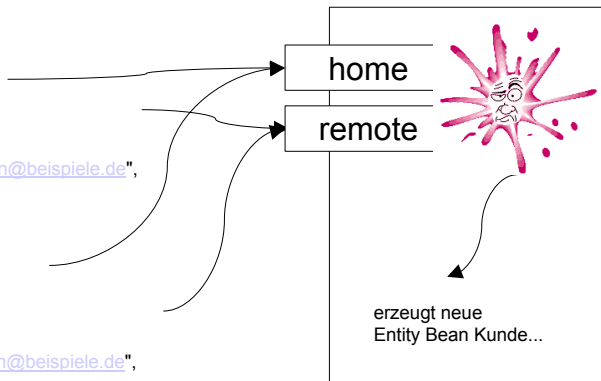
- Wird im Rahmen eines "Client"-Aufrufs ausgeführt
- Ist Transaktions-Aware
- Kann Datenbankaktionen ausführen
- Stellt keine Abbildung vorhandener Datenstrukturen dar
- Ist kurzlebig
- Zwei Arten:
 - Statefull bean
 - **Hat eigenen Zustand**
 - **Eine Komponente wird einem Client zugeordnet**
 - Stateless bean
 - **Hat keinen eigenen Zustand (einfacher)**

Stateless Session Bean Beispiel

Anwendungsfall "Neuen Kunde registrieren"

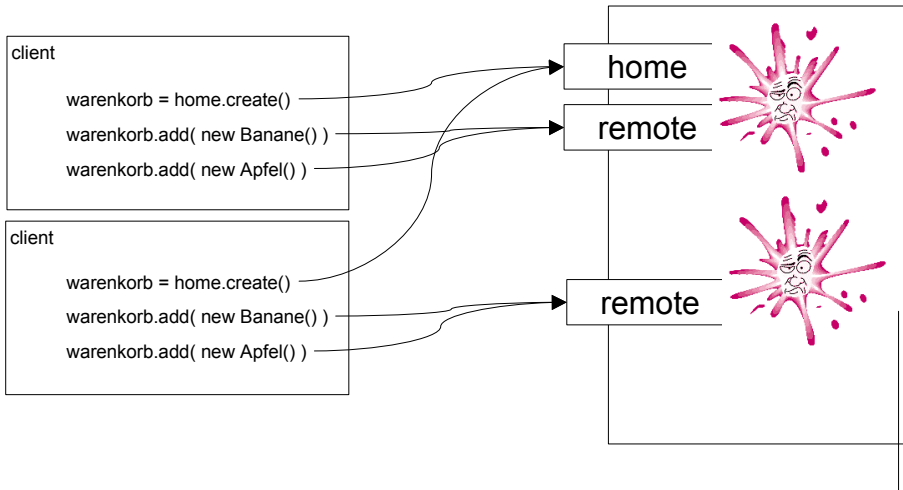
```
bean = home.create()
bean.createCustomer(
    "Peter",
    "Mustermann",
    "peter.mustermann@beispiele.de",
    "29.2.1960",
    "kenn1wort")

bean = home.create()
bean.createCustomer(
    "Peter",
    "Mustermann",
    "peter.mustermann@beispiele.de",
    "29.2.1960",
    "kenn1wort")
```



Statefull Session Bean Beispiel

Anwendungsfall "Kunde füllt Warenkorb"



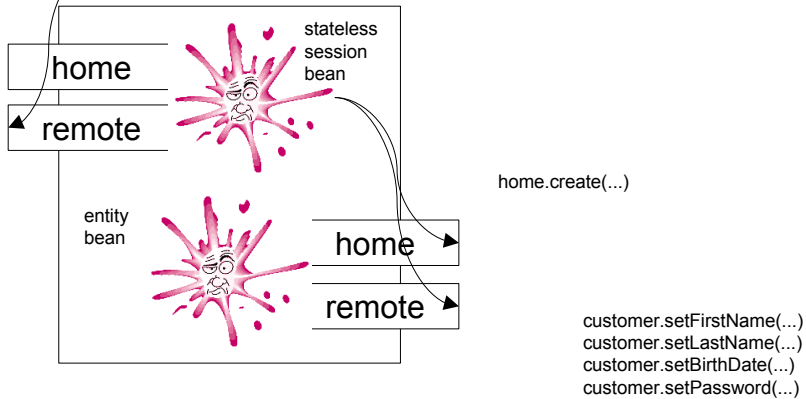
Entity Bean

- Darstellung persistenter Daten (Datensätze in DBMS) durch Komponenten
- Ermöglicht gleichzeitigen Zugriff durch mehrere "Clients"
- Lange Lebensdauer
- Zwei Arten
 - Bean Managed Persistence
 - **Persistenz wird durch Bean selbst abgewickelt**
 - Container Managed Persistence
 - **Persistenz wird durch den Container abgewickelt**

Entity Bean Beispiel

Neuer Kunde wird angelegt/ registriert...

bean.createCustomer(...)

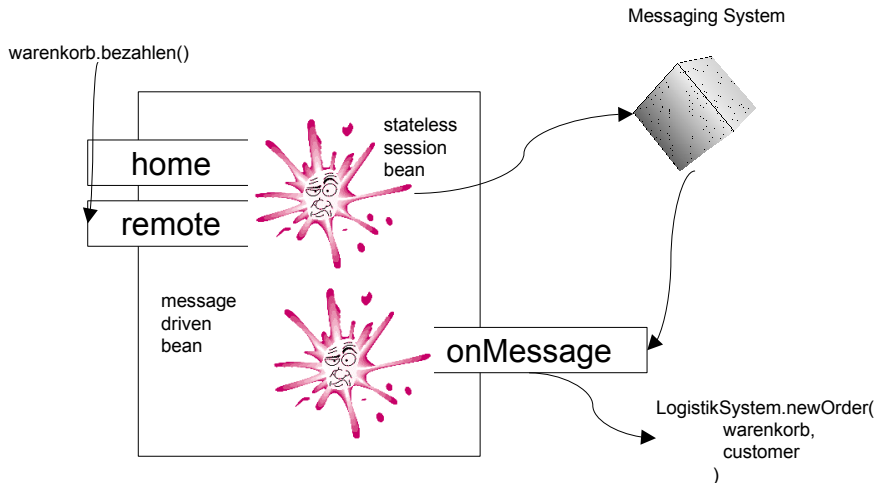


Message Driven Bean

- Wird bei Ankunft eines Ereignisses/ Nachricht ausgeführt
- Kann in Transaktionskontext eingebunden sein
- Kann Datenbankaktionen ausführen
- Stellt keine Abbildung vorhandener Datenstrukturen dar
- Ist kurzlebig
- Überlebt einen "Container-Absturz" nicht
- Ist zustandslos

Message Driven Bean Beispiel

Szenario "Kunde hat bezahlt"



Persistenz

- Bean Managed Persistence
 - Bean verwaltet eigenständig persistente Daten
 - Verbindungen zu den Ressourcen werden über den Container verwaltet
- Container Managed Persistence
 - EJB 1.1
 - **Persistente Felder werden durch öffentliche Attribute in der Bean gekennzeichnet**
 - **und im Deployment Descriptor beschrieben**
 - **Beziehungen zwischen Entities müssen selbst kodiert werden**
 - EJB 2.0
 - **Persistente Felder werden durch abstrakte getter & setter Methoden gekennzeichnet**
 - **und im Deployment Descriptor beschrieben, wobei jetzt auch die Beziehung zwischen Entities von dem Container bzw Persistenzmanager abgewickelt werden**

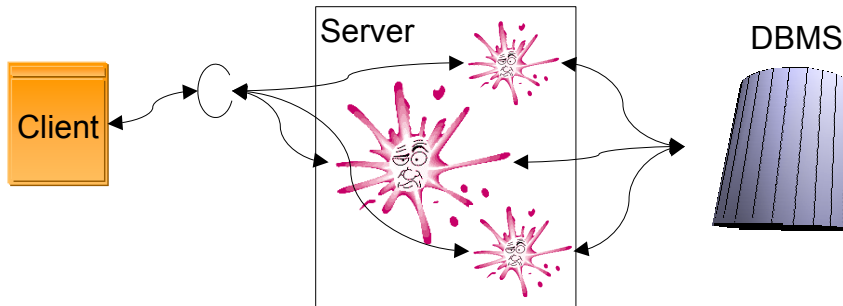
- Deklarative Sicherheit
 - Komponenten werden frei von "Sicherheitsmechanismen" entwickelt
 - Im Deployment Descriptor werden
 - **Sicherheitsrollen definiert**
 - **Rollen den verschiedenen Methoden einer Bean zugeordnet**
 - Benutzern werden Rollen zugeordnet (systemspezifisch)
- Programmatische Sicherheit
 - Für "sicherheitsbewusste" Komponenten gibt es die Möglichkeit, Sicherheitsinformationen abzufragen
- Es ist Aufgabe der einzelnen J2EE Produkte die Sicherheitskriterien des Deployment Deskriptors zu respektieren und die Sicherheitsmechanismen zu implementieren

- Sowohl deklarativ als auch programmatisch
 - Methoden werden Transaktions-Eigenschaften zugeordnet
 - Komponenten können programmatisch selbst ihr Transaktionsverhalten implementieren
 - Ein EJB-Container reagiert auf Exceptions und löst "rollback" aus
 - Der Transaktionskontext wird auch verteilt weitergegeben, basierend auf OMG's OTS

- Durch Anwendung von Industriestandards
- Durch enge Spezifikation wird Portabilität zwischen allen J2EE Produkten erreicht
 - Betriebssystem-Portabilität
 - Sicherheitssystem-Portabilität
 - Datenbanksystem-Portabilität
 - Application-Server-Portabilität
- Kommunikation zur "Außenwelt"
 - Internet (web) Clients: durch HTML & XML
 - Komponenten in Fremdsprachen: durch CORBA/IIOP
 - Legacy Enterprise Systeme: durch J2EE Connector

- Einführung (schon vorbei)
- Komponentengröße
- Das CORBA Komponentenmodell (CCM)
- Integration von CORBA Komponenten mit EJB 1.1
- Enterprise JavaBeans 2.0
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- Einschätzung des Reifegrades von CCM

- CCM und EJB sind Komponentenmodelle für Serveranwendungen

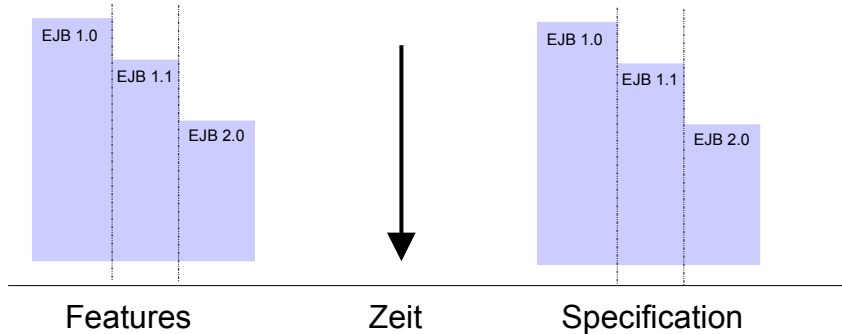


EJB Entstehung

- EJB 1.0
 - Definition von Komponenten (Session Bean; Entity Bean)
 - Definition von Rollen
 - Verantwortlichkeiten
 - Formate
- EJB 1.1
 - "Entity Beans"-Pflicht
 - Rollendefinitionen vertieft
 - nähere Spezifikation für das Zusammenstellen und Installieren von Komponenten
- EJB 2.0
 - Einführung der "Message Driven Beans"
 - neues Persistenzmodell
 - Interoperabilität mit Legacysystemem/ CORBA

EJB Entstehung

- Einfaches Konzept wird sehr eng spezifiziert
- Einzelne Themenbereiche werden in gesonderten API's sehr genau spezifiziert
- Verfeinerung der Spezifikation in folgenden Releases
- Einführung neuer Features



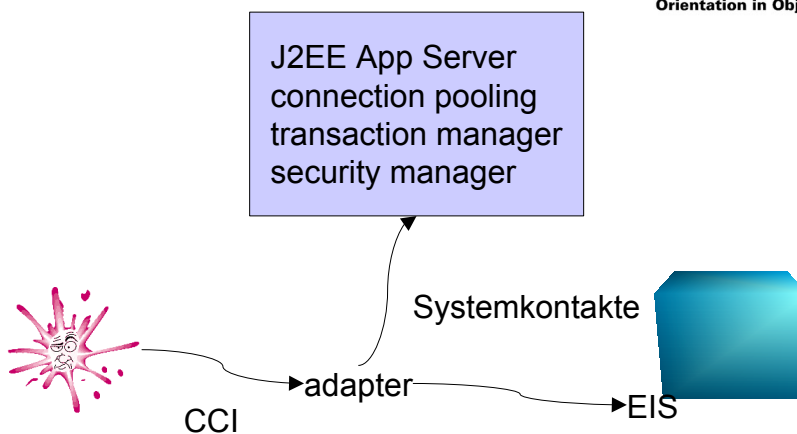
EJB - Komponentenumfeld

EJB Umfeld: J2EE !!!

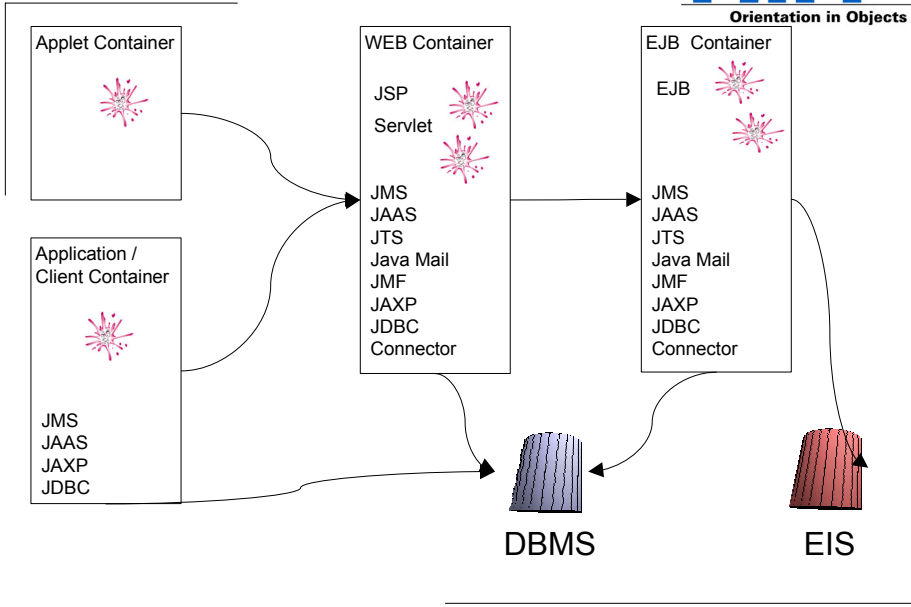
- Minimalvoraussetzung: JVM
- Festlegung aller Bestandteile
 - **J2SE Application Clients**
 - **Applets**
 - **JSP - Java Server Pages**
 - **Servlets**
 - **EJB - Enterprise Java Beans**
- Rollen in der Entwicklung des Systems
- Festlegung alle notwendigen Schnittstellen (API's)
- Definition von Sicherheits-, Transaktions- und Namensdienste
- Definition der Interoperabilität mit "legacy systems"

- J2EE API
 - EJB 2.0
 - JDBC 2.0
 - Servlets 2.3
 - JSP 1.2
 - JMS 1.0
 - JTA 1.0
 - JavaMail 1.2
 - JAF 1.0
 - JAXP 1.1
 - Connector 1.0
 - JAAS 1.0
- J2EE SPI
 - Network Protocols
 - Internet Protocols (TCP/IP, HTTP, SSI, TLS)
 - OMG Protocols (Corba 2.3.1 IIOP, CSiv2, COSNaming)
 - Java Technology Protocols (JRMP)
 - Data Formats (HTML3.2, GIF, JPEG, JAR, CLASS)
 - Deployment Descriptors

J2EE Connector Architecture

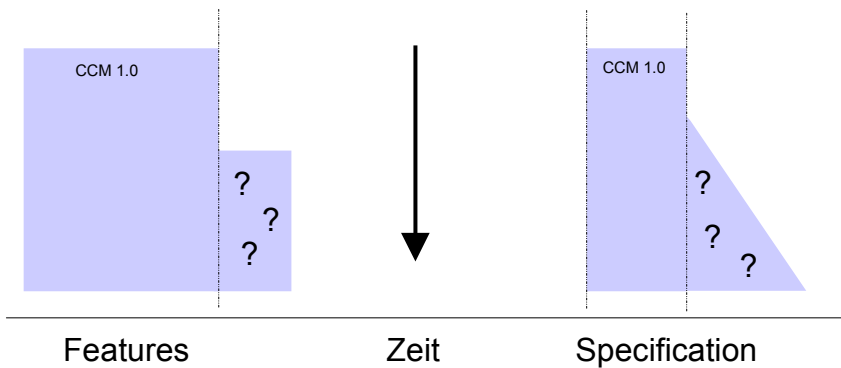


Übersicht



CCM Entstehung

- Erste Spezifikation noch nicht "final" (2 ½ Jahre)
- Sehr umfangreiche features
- Features nicht ausreichend spezifiziert



- CCM Umfeld:
 - keine Festlegung auf Plattformvoraussetzungen außer CORBA
 - 2 etablierte Komponentenmodelle zur Orientierung
 - möglichst Integration von EJBs
 - CORBA Services, Facilites,
 - Rollen in der Entwicklung des Systems
 - Über MOF Verzahnung möglich mit:
 - **UML**
 - **XMI**
 - **Workflow Specification**
 - **OMG Business Objects**

- Einführung (schon vorbei)
- Komponentengröße
- Das CORBA Komponentenmodell (CCM)
- Integration von CORBA Komponenten mit EJB 1.1
- Enterprise JavaBeans 2.0
- Vergleich des Vorgehens bei der Entwicklung der Modelle
- **Einschätzung des Reifegrades von CCM**

- „The CORBA Components Specification is not finalized. As adopted, it is severely underspecified and unimplementable, and nobody has stepped forward to fill in the missing bits and pieces. Currently, the deadline for the CCM FTF report is December 31, 2000, but there are plans to extend it this week. „

Martin v. Loewis 12/12/2000
in comp.object.corba

- „It (CCM) has not been given final approval by the membership of OMG, but in this case, that is considered a formality. The new standard will become official by the end of the year.“

Nicht verabschiedet bis min. 02/2001

- „I wouldn't expect the CCM standard to exert much influence on the component market next year.“

d.h. bis Mitte 2002

- „In other words, the OMG has not created another component model that will compete with MTS and EJB in 2000. Instead they have ... created a component solution for the future.“

nicht die allernächste Zukunft
Paul Harmon Oct. 13, 1999
<http://210.73.90.65/~wangweihan/CORBA/advisory.htm>

OMG TC Work in Progress:

- FTF Recommendation and Report Dec 31, 2000
- Veto Power May 16, 2001

- Veto Power List
 - BEA Systems June 16, 2001

last updated 13-Dec-2000 13:11:18 EST

http://www.omg.org/techprocess/meetings/schedule/Components_FTF.html

- „It (CCM) has not been given final approval by the membership of OMG, but in this case, that is considered a formality. The new standard will become official by the end of the year.“

Nicht verabschiedet bis min. 06/2001

- „I wouldn't expect the CCM standard to exert much influence on the component market next year.“

d.h. bis Ende 2002

- „In other words, the OMG has not created another component model that will compete with MTS and EJB in 2000. Instead they have ... created a component solution for the future.“

nicht die nächste Zukunft

Paul Harmon Oct. 13, 1999

<http://210.73.90.65/~wangweihan/CORBA/advisory.htm>

- „I wouldn't expect the CCM standard to exert much influence on the component market next year.“
- „I wouldn't expect the CCM standard to exert any significant influence on the component market before 2001 at the earliest.
- „In other words, the OMG has not created another component model that will compete with MTS and EJB in 2000. Instead they have ... created a component solution for the future.“
- ... it's hard to imagine CCM will ever capture a large segment of the server-side component market.

gar nicht?

Paul Harmon Jun, 2000

<http://www-106.ibm.com/developerworks/library/corba-comp.../?dwzone=component>

- Die Entscheidung für ein Komponentenmodell ist stark bindend: ein grundsätzlicher Wechsel ist aufwendig / nicht möglich (= Neuentwicklung!)
- Komponentenmodelle müssen sofort stabile Basisfunktionalität bieten
- Anwendungen wachsen unter Umständen über die Grenzen eines Applikation-Servers hinweg: Portierung auf performantere Produkte darf nicht aufwendig sein!

- EJB bereits sehr etabliert
 - verschiedene Produkte mit brauchbarer Interoperabilität
 - der Prozeß der Internet-Öffnung der Geschäftsmodelle ist fest in der Hand von EJBs
 - Java ist erste Wahl für diese Projekte, EJB-Sprachfestlegung kein Problem
 - Integration möglich über CORBA oder Connectoren
- CCM noch nicht als Spezifikation verabschiedet
 - keine Implementierung verfügbar
 - EJB-Integration wird in Hasenrolle bleiben
 - Interessant für Unternehmen, die im EAI-Bereich bereits auf CORBA-Infrastruktur setzen
 - bei komplexen Modellierungsaufgaben stärkeres Modell

- Jon Siegel, CORBA 3 Fundamentals and Programming, ISBN: 0-471-29518-3
- Vogel et. al., Java Programming with CORBA, ISBN: 0-471-37681-7
- Zahavi, EAI with CORBA Component...., ISBN: 0-471-32720-4,

- <http://www.ditec.um.es/~dsevilla/ccm/>
- <http://www.cs.wustl.edu/~schmidt/PDF/middleware2000.pdf>
- http://www.omg.org/techprocess/meetings/schedule/Components_FTF.html
- http://cgi.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html
- <http://openorb.exolab.org/extensions.html>