Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet: Telematik

Hauptseminararbeit
Sommersemester 2003

Thema:

Analyse der CCM-Container und Vergleich mit EJB-Systemen

Bearbeiter : Elachouri Abdelkader – WI99
Betreuer : Dipl.-Inf. Thorsten Strufe

Inhaltsverzeichnis

1. Einführung

Die Verbreitung der Vernetzung und Verteilung führt zu einer zunehmenden Komplexität der Anwendungen, deshalb werden gut durchdachte und sauber geplante Konzepte und Softwareentwürfe immer wichtiger. Es ist wichtig, den Entwickler mit Hilfe einer höheren Abstraktion durch mehrschichtige Architekturen, möglichst wenig mit den vorhandenen Systemen in Berührung kommen zu lassen. Erfahrungsgemäß verbringen

> *„...Programmierer bei der Konstruktion verteilter Systeme über 50 Prozent ihrer Zeit damit, ihre Anwendung in die bestehende Infrastruktur zu integrieren, beispielsweise in die vorhandenen Kommunikationsmechanismen, Transaktionsmonitore, Sicherheitskonzepte, Multithreading-Bibliotheken oder Datenbanksysteme“* [Wagner_2000]

Die Installation verteilter Dienste auf mehrere Plattformen wird durch die Heterogenität heutiger Serverlandschaften mit unterschiedlichen Standards noch zusätzlich erschwert.
Im Interesse der Entwickler steht dabei die Interoperabilität, also in welcher Weise unabhängig voneinander entwickelte Komponenten zur Laufzeit miteinander arbeiten, die Sprachen- und die Plattformenabhängigkeit.
Zu den am weitesten entwickelten Komponentenarchitekturen zählen CORBA, die Enterprise Java Beans (EJB) von Sun sowie Microsofts COM+. CORBA ist im Gegensatz zu EJB nicht an eine spezifische Programmiersprache gebunden.
Die Object Management Group (OMG) stellt mit dem Komponentenmodell des neuen CORBA 3.0 Standards eine umfangreiche und offene Plattform für verteilte Anwendungen bereit. Das Komponentenmodell, und damit das CCM Container Modell, ist, im Gegensatz zu konkurrierenden Konzepten, nicht auf eine festgelegte Programmiersprache oder eine Betriebssystemplattform beschränkt.

2. Middleware als Kommunikationsalternative

Die „N-Tier-Architektur“ gibt an, wie viele funktionale Ebenen in einem gegebenen System existieren. Bei der Entwicklung webbasierter Anwendungen wird meistens eine 4-Tier-Architektur entworfen (Clients, Webserver, Applikationsserver und Datenbankserver), bei der die Anwendung gemäß ihrer Funktionalitäten auf verschiedenen Ebenen verteilt wird.
Damit die Webserver mit den Anwendungsservern (oder untereinander) kommunizieren können, wird die sog. Middleware-Technologie eingesetzt, mit deren Hilfe eine sichere und ziemlich einfache Entwicklung möglich gemacht wird. Eine bekannte Middleware-Technologie ist CORBA.

2.1 CORBA

2.1.1 Was ist CORBA?

CORBA setzt sich aus den Anfangsbuchstaben von **C**ommon **O**bject **R**equest **B**roker **A**rchitecture zusammen und ist eine von dem Standardisierungskonsortium Object Management Group (OMG) erarbeitete Spezifikation, welche

> *„die Zusammenarbeit von Softwaresystemen über die Grenzen von programmier-sprachen, Betriebssystemen und Netzwerken hinweg regelt .“* [Wang_2000]

CORBA stellt eine universelle Kommunikationsplattform dar, mit deren Hilfe Objekte zusammenarbeiten können, deren Implementierungen sich auf unterschiedlichen Plattformen

befinden. Mit CORBA ist es möglich, Softwareapplikationen Operationen auf verteilten Objekten aufzurufen, ohne sich darum kümmern zu müssen, wo sich die Objekte befinden, welche Programmiersprache, Kommunikationsprotokolle, Verknüpfungen, Betriebssystem oder Hardware verwendet werden. Von OMG werden CORBA ObjectServices spezifiziert, auf die mittels Standardschnittstellen alle Komponenten auf gemeinsame nutzbare Services wie z.B. Benennung (naming) oder Ereignisverständigung (event notification) zugreifen können.
Zentrales Organ dieser Architektur ist der Object Request Broker (ORB), der die Kommunikation und Interaktion von Objekten innerhalb vernetzter, meist heterogener Systeme übernimmt.

2.1.2 Wie funktioniert CORBA?

Um Objekte anzusprechen, wird ein ***Stub*** angeschaltet, dieser ermöglicht den Zugriff auf die Operationen und leitet diese zum ORB weiter. Das ***Skeleton*** ist die Umwandlung der Nachrichten in entsprechende Aufrufe für das angesprochene Objekt entsprechend seiner Schnittstellen, die via ORB weitergeleitet werden. Die *Stub*s und *Skeletons* werden automatisch durch den IDL-Compiler generiert und sind von Implementierung der Objekte und dem Betriebssystem abhängig. [CORBA1]

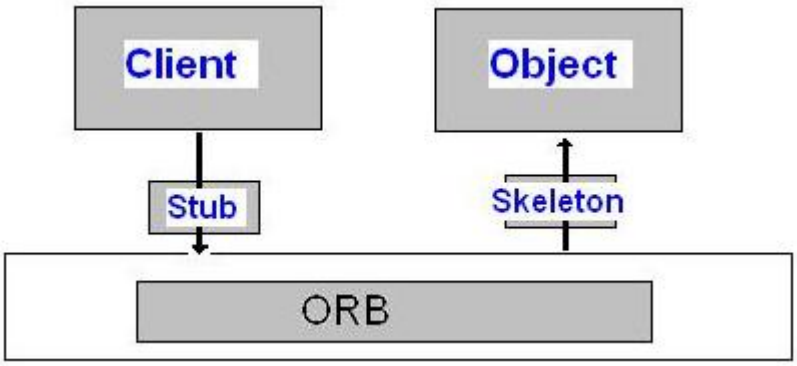

Abb.1 : Kommunikation via ORB

2.2 CORBA Components Model (CCM)

2.2.1 CCM: Definition & Bestandteile

Das CCM ist eine Spezifikation von OMG, die die Erstellung von serverbasierten, skalierbaren und plattformunabhängigen Anwendungen unterstützen soll [HuBerlin]. Es ist ein erweitertes CORBA Object Model und definiert Funktionalitäten und Services, die es Applikationsentwicklern ermöglichen, Komponenten zu implementieren, zu managen, zu konfigurieren und zu nutzen, die gemeinsame CORBA Services integrieren, wie z.B. Transaktion, Sicherheit, Dauerhaftigkeit und Ereignisbenachrichtigung.
Bis zur Version (2.x) gab es kein explizites Komponentenmodell für CORBA. Im Rahmen der Version 3.0 entsteht das CORBA Component Model (CCM).
Die CCM-Spezifikation sieht prinzipiell drei verschiedene Mechanismen vor [HuBerlin]:

- **Kommunikation über Schnittstellen**: Komponenten können mehrere unterschiedliche Schnittstellen (engl. facets) haben, die sie nach außen hin präsentieren. Umgekehrt können Komponenten auch Referenzen auf externe Agenten

enthalten (engl. receptacles), die es ihnen ermöglichen, auf deren Schnittstellen zuzugreifen. Die Kommunikation über Schnittstellen ist eine synchrone Kommunikationsform.

- **Kommunikation durch Ereignisse:** Komponenten sind einerseits Ereignisquellen, die typisierte Ereignisse versenden können und können andererseits auch als Ereignissenken fungieren, die solche Ereignisse empfangen und verarbeiten. Zwischen zwei typgleichen Endpunkten (d.h. einer Quelle und einer Senke des gleichen Ereignistyps) können so Ereigniskanäle (engl. event channels) aufgebaut werden. Z.B kann der Transport der Ereignisse durch den CORBA-Ereignisdienst realisiert werden, aber auch andere Mechanismen können in Frage kommen, solange die vorgeschriebene Ereignissemantik beachtet wird. Die Kommunikation über Ereignisse ist eine asynchrone Kommunikationsform.
- **Kommunikation über Attribute:** Für die Konfiguration einer Komponente sind in erster Linie die so genannten Attribute zuständig, sie funktionieren wie Variablen, die von externen Agenten über Zugriffsmethoden verändert werden können.

Die vorher erwähnten Endpunkte, die Facette, Ereignisquelle/-senke, Rezeptor und Home werden auch als Anschlüsse (engl. ports) bezeichnet.

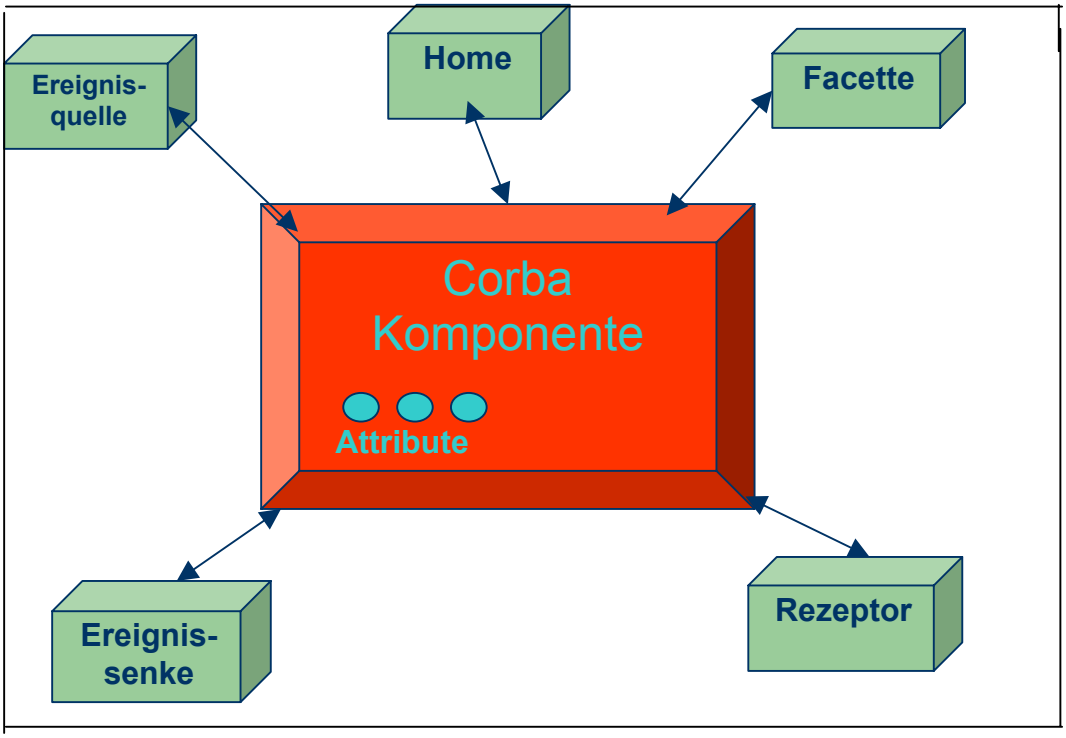

Abb.2 : Corba Components Model

Trotz dieser Erweiterungen bleibt die konventionelle CORBA-Infrastruktur grundsätzlich erhalten, die sich auf den Object Request Broker (ORB) stützt, an den über einen Objektadapter die Objekte angebunden werden. Unter CCM Spezifikation ist es auch wie bisher möglich, Objekte zu verwenden und auf die Verwendung von Komponenten zu verzichten. Aber auch die Interaktion von Komponenten und konventionellen Objekten ist denkbar.

Die vier wichtigsten Object Services (CORBAservices) des ORB sind [HuBerlin]:

- **Transaction Service**: Ermöglicht es verschiedenen verteilten Objekten, an einer atomaren Transaktion teilzunehmen.
- **Security Service**: Stellt einen ganzen Satz von Sicherheitsmechanismen bereit. Er unterstützt z. B. Authentifizierung und Zugriffskontrolle.
- **Event Service**: Ermöglicht es Objekten, ihr Interesse an bestimmten Ereignissen dynamisch registrieren zu lassen.
- **Persistent Object Service**: Ermöglicht es Objekten, auch dann dauerhaft zu existieren, wenn die Applikation, die das Objekt erzeugt hat, beendet wurde

Die drei Hauptaspekte des Corba Component Models sind [JH&FJ]:

- Die Integration von EJB
- Die Festlegung eines Formates zur Verteilung von Komponenten
- *Die Spezifikation einer als **Container** bezeichneten Laufzeitumgebung.*

2.2.2 CCM Container Modell

Container, die das Corba Component Model unterstützen, sind für die Erzeugung, Lokalisierung und Zerstörung von Objekten, die Einbettung der Aufrufe von Objektmethoden in Transaktionen, Einhaltung von Sicherheitsaspekten und die Persistenz von Objekten und dessen Zustände zuständig. Diese Funktionen stehen den Entwicklern zur Verfügung, die Implementierung ist aber von den letzteren verborgen.
Der Container verwaltet die von Komponenten erzeugten und konsumierten Ereignisse und stellt Kommunikationskanäle bereit, ohne dass eine Komponente das Wissen über die Interaktion mit dem darunter liegenden System benötigt. Stattdessen wird durch einen Container eine Laufzeitumgebung für Komponenten bereitgestellt.

2.2.2.1 CCM Container Architektur

Ein Container weist zwei unterschiedliche Arten von Schnittstellen auf: Die External APIs sowie die Container APIs.

External APIs
External APIs können direkt durch andere Elemente des Modells (Clients, fremde Komponenten,...) genutzt werden. Hierzu zählen die Äquivalenzschnittstelle, die Facetten sowie der Home.

Container APIs
Bei den Container APIs unterscheidet man zwischen ***Internal Interfaces*** und ***Callback Interfaces.***
Über ***Internal Interfaces*** ist es der Komponente möglich, die vom Container angebotenen Dienste zu nutzen. Hierfür stehen Methoden bereit, um die Transaktions-, Sicherheits- oder Ereignisschnittstellen abzurufen.

Die ***Callback Interfaces*** sind dagegen von der Komponente zu implementieren, um sie den Container-Rückrufschnittstellen zur Verfügung zu stellen, über die beispielsweise eine Benachrichtigung über relevante Ereignisse erfolgt.
Die Nutzung dieser Schnittstellen ist natürlich von der Art der zu verwaltenden Komponente sowie der jeweiligen Aufgabe abhängig. Um Komponentenreferenzen mit oder ohne einem Persistenzstatus zu verwalten, existieren in dem Modell daher Container in zwei unterschiedliche Ausprägungen:

> Entity Container unterstützen persistente Komponenten, während Session Container für eine transiente Verwaltung der Komponenten zuständig sind.

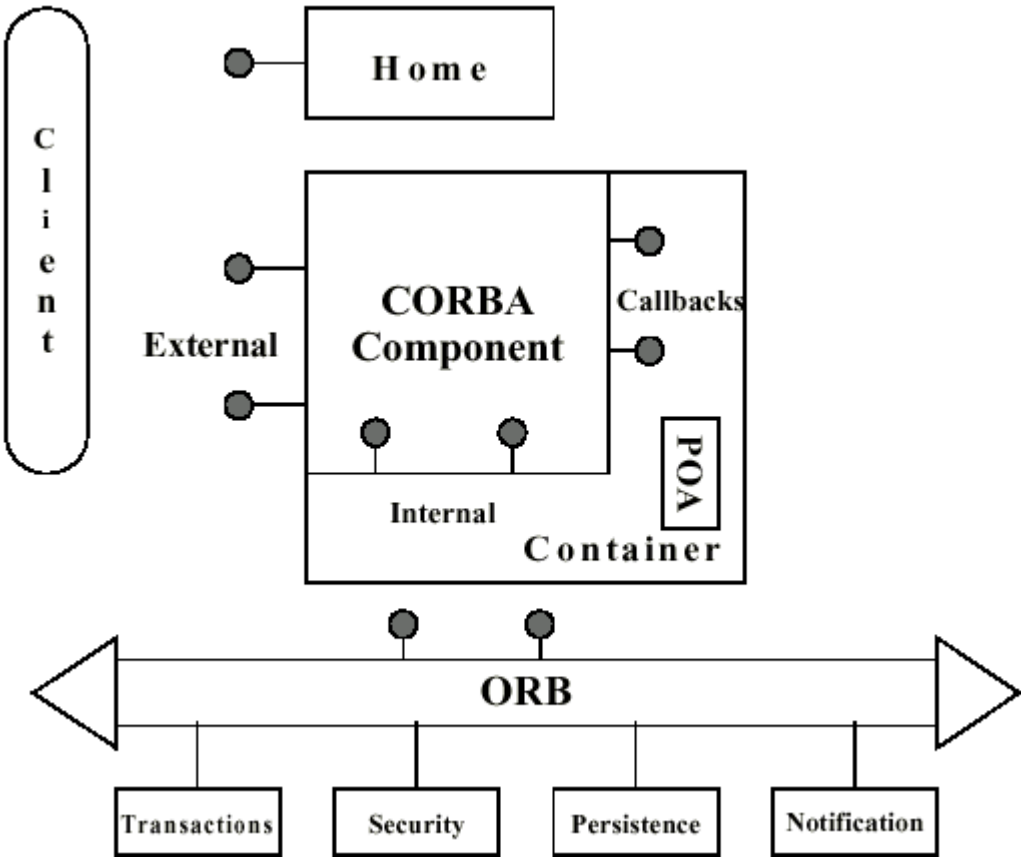

Abb. 3: Das Container-Modell [JoSi]

Für die Interaktion von Containern mit den zugrundeliegenden CORBA-Diensten existieren die sogenannten „Container Implementation Types“. Bei der Verwendung eines *transienten Containers* ist nur eine zustandslose Interaktion möglich. Hierbei unterscheidet man zwei unterschiedliche Typen von Interaktionen in Abhängigkeit vom Verhältnis „Instanz zu Servant“.
Wenn ein Servant sämtliche Instanzen eines Komponententyps repräsentiert, dann spricht man von einer Interaktion des Typs „stateless“, wohingegen beim Typ „conversational“ zu jeder Instanz ein eigener Servant geordnet wird. Bei persistenten Containern gibt es nur einen Containerimplementierungstyp, der „durable“ genannt wird. Auch hierbei wird jede Komponente mit einer eindeutigen Referenz auf einen Servant verbunden, allerdings kommt hierbei eine persistente Referenz zum Einsatz.
Daraus ergeben sich 4 mögliche Komponentenkategorien *(Service, Session, Process and Entity*) als Kombinationen von Containertyp, Container Implementation Type sowie der Verwendung / Nichtverwendung von Primärschlüsseln für die Komponenten

1. **Service Components** sind zustandslos und damit nicht an einen Client gebunden.
2. **Session Components** besitzen einen transienten Zustand, verändern diesen aber in Abhängigkeit vom Clientaufruf, womit eine feste Assoziation mit dem Client für die Dauer einer Sitzung besteht.
3. **Process Components** besitzen einen persistenten Zustand, der Zugriff ist allerdings nur durch den Originalclient möglich.
4. **Entity Components** besitzen ebenfalls einen persistenten Zustand, durch die Verwendung eines Primärschlüssels ist der Zugriff jedoch von jedem Client aus möglich.

Komponentenkategorie	Containertyp	Containerimplementierungstyp	Primärschlüssel
Service	Transient	Stateless	Nein
Session	Transient	Conversational	Nein
Process	Transient	Durable	Nein
Entity	Persistent	Durable	Ja

Tabelle 1: Komponentenkategorien

2.2.3 CCM Container Dienste

Container verwalten auch den Lebenszyklus des Komponenten-Servants. Zu unterscheiden ist zwischen dem CORBA-Objekt (einem virtuellem Konzept) und seiner konkreten Implementierung, dem *Servant*. Im Laufe der Lebenszeit eines CORBA-Objekts können verschiedene *Servants* dasselbe CORBA-Objekt implementieren. Die Container kontrollieren mittels *Servant Lifetime* die Speichernutzung eines, für eine bestimmte Komponenten-facette, implementierenden Servants.

a) Servant Lifetime

Es sind vier verschiedene Richtlinien der Aktivierung und Deaktivierung von Komponenten definiert:

- **METHOD**:
  Die Komponente wird beim ersten Methodenaufruf aktiviert und ihre Deaktivierung erfolgt nach jedem Methodenaufruf.

- **TRANSACTION**:
  Die Aktivierung der Komponente erfolgt beim ersten Aufruf innerhalb der Transaktion, die Komponente wird bei Transaktionsende passiviert.

- **COMPONENT**:
  Die Komponente wird beim ersten Methodenaufruf aktiviert, eine Deaktivierung geschieht erst auf explizite Anfrage der Komponente.

- **CONTAINER**:
  Auch hier erfolgt die Aktivierung der Komponente mit dem ersten Methodenaufruf und wird passiviert, wenn der Container einen neuen Speicherplatz benötigt.

Die Richtlinie „METHOD“ ist nur bei Verwendung des Containerimplementierungstyps *„stateles*s“ anwendbar, während beim Typ *„conversationa*l“ oder *„durabl*e“ alle 4 Möglichkeiten Verwendung finden können.

b) Transactions

Komponentenentwickler haben die Wahl zwischen selbstverwalteten Transaktionen, durch Zugriff auf den CORBA Transactions Service, oder containerverwalteten Transaktionen, unter Zuhilfenahme des Komponentendeskriptors. Innerhalb des Komponentendeskriptors können folgende Attribute spezifiziert werden

- NOT_SUPPORTED
- REQUIRED
- SUPPORTS
- REQUIRES_NEW
- MANDATORY
- NEVER

c) Security

Die CCM Spezifikation sieht eine deklarative Sicherheitsimplementation mit Hilfe des Komponentendeskriptors vor. Der CCM Container verwendet den CORBA Security Service zur Zugriffskontrolle auf die Methoden einer Komponente.
Applikationen können Security Credentials auch zur Laufzeit überprüfen.

d) Notification/Events

Die Notification/Events in CCM Spezifikation basieren auf den Funktionen des CORBA Notification Services.
Der Container ist für den Aufbau des Event Channels und die QoS zuständig.
Events können, durch Spezifikation innerhalb des Komponentendeskriptors, sowohl transaktional sein, als auch Access Control Listen beinhalten.

e) Persistence

Der Persistenzdienst des CCM Containers unterstützt allein Process- und Entity-Komponenten. Eine Komponente kann ihren persistenten Zustand selbst auf eine Datenbank abbilden (Self-Managed) oder die Dienste des Containers in Anspruch nehmen (Container-Managed).
Container-Managed Persistence Komponenten werden in jedem Fall über den CORBA Persistence Storage Service abgebildet. Komponenten, die ihren persistenten Zustand selbst verwalten, haben die Wahl zwischen dem CORBA PSS oder alternativen Methoden (ODBC, JDBC).

3. Sun Microsystems Enterprise Java Beans

Microsoft hat Active Server Pages und Java hat Java Server Pages. Außerdem stellt Microsoft den MTS (Microsoft Transaction Server) her, einen Server für verteilte Transaktionen, der verteilte Objekte unterstützt. Enterprise JavaBean (EJB) scheint eine direkte Antwort auf MTS zu sein, bietet aber mehrere Funktionen, die es bei MTS nicht gibt.

EJB ist eine besondere Art von JavaBean, die der Durchführung serverseitiger Geschäftslogikoperationen dient. EJBs können Daten darstellen, die in einer Datenbank gespeichert sind. Sie können sogar mit einem EJB-Server Datenbankdaten automatisch in eine Bean einlesen und Bean-Daten automatisch in die Datenbank zurück schreiben. Das spart bei einer großen Datenbank viel Zeit.

EJB-Architektur

Eine vollständige EJB Architektur besteht im Wesentlichen aus

- einem EJB Server
- einem oder mehreren EJB Containern, die in der Umgebung des EJB Servers laufen
- einer Anbindung an die Implementation weiterer Spezifikationen, wie dem Java Naming and Directory Interface (JNDI) und dem Java Transaction Service (JTS)

Die folgende Abbildung zeigt, wie ein Client mit den Interfaces *Home* und *Remote* einer EJB interagiert. Der Client erstellt mit dem Interface *Home* eine EJB und kommuniziert dann über das Interface *Remote.*

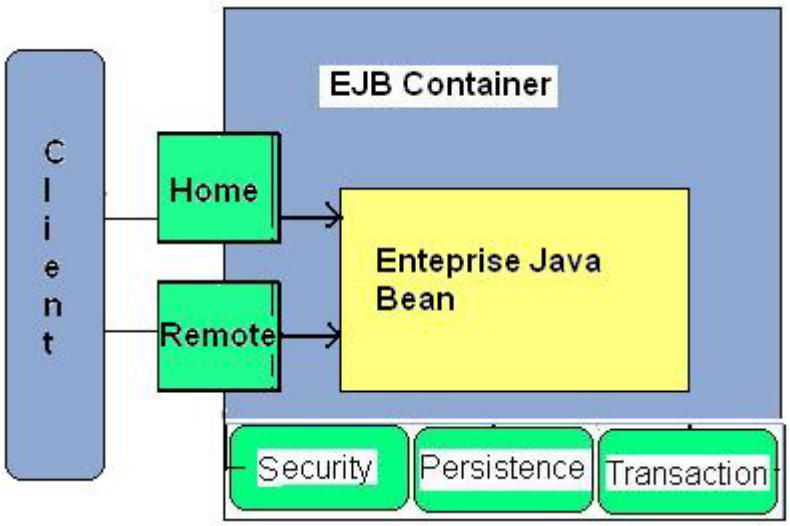

Abb. 4: Interaktion zwischen Client und EJB-Container

Es gibt 3 Arten von EJBs:

- Eine Session-Bean
- Eine Entity-Bean
- Eine Message-Bean.

3.1 Session- / Entity-Beans

3.1.1 Session-Bean

Session-Bean stellt eine Unterhaltung zwischen einem Client und einem Server dar, im Besonderen eine Unterhaltung zwischen Client-Anwendung und EJB-Container.
Im Normalfall gibt es pro Client eine einzige Sitzung und die Session-Beans sind keine persistenten Datenobjekte, d.h. eine Sitzung speichert man nicht in der DB.
Session-Beans implementieren die Geschäftslogik und arbeiten bei der Durchführung einer Operation häufig mit mehreren Entity-Beans zusammen, sie müssen aber nicht eine Entity-Bean benutzen.
Bei dem Entwurf von EJB-Anwendungen weiß man nicht, ob man mit der Bean beginnen und dann die Interfaces einrichten soll oder umgekehrt. Im besten Fall fängt man mit den Interfaces an: - Interface Remote
- Interface Home und
- Implementierungsklasse.

3.1.2 Entity-Bean

Eine Entity-Bean stellt ein persistentes Datenobjekt dar und muss persistent in einer Datenbank gespeichert werden. Sie kann von mehreren Clients verwendet werden ohne sichtbar zu sein und arbeitet mit den Session-Beans zusammen.
Eine Entity-Bean hat oberflächlich betrachtet Ähnlichkeit mit einer Session-Bean, beide haben Home- und Remote-Interfaces und eine Implementierungsklasse. Unterschiede gibt es im Bereich der Persistenz.
Die Entity-Beans stehen mehreren Anwendern gleichzeitig zur Verfügung, da sie Datenelemente repräsentieren, auf die mehrere Anwender zugreifen können. Der EJB-Container verwaltet die Konflikte, die entstehen können, wenn zwei Clients dieselbe Bean benutzen wollen.
Es gibt zwei Arten der Persistenz von Entity-Bean: Bean-Managed Persistence (BMP) und Container-managed Persistence (CMP):

i) BMP
BMP bedeutet, dass die Implementierungsklasse der Bean alle SQL-Operationen durchführt, die für das Laden der Bean-Daten aus der Datenbank, das Speichern und das Aktualisieren der Daten notwendig sind. BMP gibt also der Bean die Kontrolle über den Datenbankzugriff, damit hat die Bean mehr Verantwortung: sie muss alle Datenbankverbindungen, die sie verwendet, verfolgen und sicherstellen. Sie muss außerdem herausfinden, wann auf verbundene Datenelemente zugegriffen und wie die verbundenen Elemente gespeichert und aktualisiert werden sollen.
Eine Entity-Bean hat einen komplexen *Lebenszyklus*. Der EJB-Container erstellt eine Entity-Bean und setzt sie in einen *Bean-Pool.* Der Pool ist eine Gruppe unsichtbarer Beans. Der Container holt sich eine Bean aus dem Pool zur Bearbeitung, wenn die Clients (oder Client) fertig sind, erstellt der Container eine neue Entity-Bean-Instanz und gibt ihr mit *SetEntityContext* einen Kontext. Nachdem dieses Vorgehen beendet wurde, kommt die Bean in den Pool zurück.
Die folgende Abbildung zeigt den Lebenszyklus einer Bean und ihren Weg vom Pool zum aktiven Status und wieder zurück.

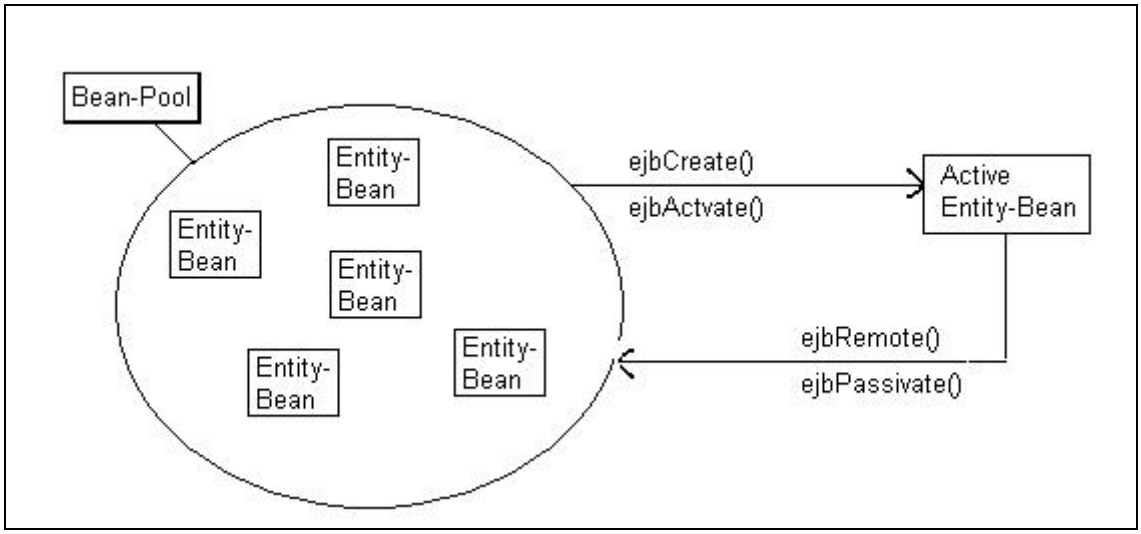

Abb. 5: Bean-Lebenszyklus

Der Container ruft die Methode ejbCreate auf um eine neue Entity-Bean zu erstellen und teilt der Bean mit der Methode ejbActivate mit, dass sie nun aktiv ist.
Wenn der Server bzw. Container die Instanz nicht mehr benötigt, wird die Methode ejbRemove() aufgerufen um die Bean aus dem Speicher zu entfernen. An dieser Stelle sollte die Bean-Instanz Aufräumarbeiten wie das Schließen offener Ressourcen durchzuführen.

ii) CMP
CMP bedeutet, dass der EJB-Container weiß, wie die Bean und die Datenelemente geladen und gespeichert werden. Der Entwickler spart viel Zeit bei der Implementierung von Geschäftslogik, da er den Code für den Datenbankzugriff nicht schreiben muss.
Wenn man eine Bean bereitstellt, die die CMP benutzt, genügt die Mitteilung der Container, wo die Bean gespeichert werden soll.
Die Fähigkeit der CMP hängt von der Implementierung des Persistenzmanagers ab, d.h der EJB-Container hat einen Persistenzmanager, der die CMP durchführt. Die EJB-Spezifikation gibt keine Information, wie ein Persistenzmanager arbeitet oder wie man die Felder einer Bean einer Datenbank zuordnet. Der Persistenzmanager ist hier der Entscheidungsträger.

3.1.3 Message-Bean

Eine Message-Bean hört Nachrichten ab, die von Java Message Services (JMS) stammen. Eine Message-Bean hat keine Clients, man kann mit ihr nur interagieren, indem man ihr JMS-Nachrichten sendet. Message-Beans wurden EJB erst kürzlich hinzugefügt und nicht alle EJB-Container verfügen über die Message-Beans-Dienste.

3.2 EJB Container Dienste

Was ist ein EJB-Container?
„…in J2EE ist ein Container so etwas wie ein Mini-Server, er stellt die Laufzeitunterstützung für die Elemente bereit, die er enthält.“[MaWu]
Der EJB-Container enthält EJB und ist verantwortlich für die Bildung eines Verbindungs-Pools sowie die Transaktionsverarbeitung. Dagegen enthält ein Servlet-Container Servlets und JSP.
Ein Server kann viele Container besitzen, die Objekte in den einzelnen Containern sind voneinander isoliert, können aber miteinander kommunizieren.
Der Container stellt grundsätzlich Funktionalitäten zur Verwaltung der EJB Komponente zur Verfügung. Das Enterprise Bean wird im Container eingesetzt und dort auch ausgeführt. Dabei wird es vom Container mit wichtigen Informationen während des Lebenszyklus versorgt. Hauptaufgabe des Containers ist es, Transaktionsregeln und die Sicherheitsaspekte während der Laufzeit einer Bean zur Verfügung zu stellen.

Container Sevices

Die EJB Container Dienste sind:

- Persistence
- Transactions
- Security

Component Lifecycle, Resource Pooling und Load Balancing werden von der jeweiligen Containerimplementation, für den Komponentenentwickler völlig transparent, erledigt.

a) Persistence (Laufzeitumgebung)

Enterprise Java Beans Entwickler können den Zustand entwickelter Komponenten ***deklarativ*** (d.h mit Hilfe von „Container Managed Persistence“, hier wird die Verantwortung für die persistente Ablage der Attribute dem EJB-Container übertragen) oder ***programmatisch*** (d.h als „Bean Managed Persistence“, hier kann eine Entity Bean selbst die persistente Ablage ihrer Attribute realisieren) auf eine Datenbank abbilden.

b) Transactions (Dienst und Laufzeitumgebung)

Transaktionen unterstützen die Entwickler besonders bei Fehlerbehandlungen, die durch den gleichzeitigen Zugriff mehrerer Benutzer auf bestimmte Daten auftreten können. Für jede Enterprise-Beans Komponente muss im Deployment Deskriptor spezifiziert werden, ob sie Transaktionen deklarativ, d.h. „Container Managed“, oder programmatisch, also „Bean Managed“ (JTS/JTA), verwendet.

Der Container muss andere Transaktionen solange daran hindern, verwendete EJB Komponenten zu benutzen, bis die gerade stattfindende Transaktion übergeben bzw. abgebrochen wird. Dies bedeutet jedoch, dass eine langlebige Transaktion die „Entity EJB“ sperren könnte.
Das Transaktionsverhalten des Containers wird bei deklarativen Transaktionen im Transaktionsattribut des EJB Deployment Deskriptors vermerkt [Sun00].

c) Security (Laufzeitumgebung)

Die Spezifikation verpflichtet den EJB-Container, den Enterprise-Beans ein Sicherheits-management als Bestandteil der Laufzeitumgebung bereitzustellen. Sie ist rein deklarativ und zielt darauf ab, sicherheitsspezifischen Code in EJB Business Methoden zu vermeiden.
Es besteht die Möglichkeit, Benutzerrollen zu definieren. In jeder Enterprise-Beans können einer bestimmten Rolle verschiedene Rechte erteilt werden. Die Rechte legen fest, ob der Benutzer die Methoden einer Enterprise Java Bean aufrufen darf.
Die entsprechenden Rollen und Zugriffsrechte werden innerhalb des Deployment Deskriptor’s festgelegt.

4. CCM Container vs. EJB-Systeme

Am Anfang stand das Ziel der CORBA-Komponentenspezifikation die problemlose Kompatibilität mit Enterprise JavaBeans. Daher sind java-basierte CORBA-Komponenten bei Beachtung kleinerer Einschränkungen problemlos in EJB-Containern nutzbar. Umgekehrt repräsentieren EJB-Komponenten gleichzeitig CORBA-Komponenten und sind demzufolge in CCM-Containern verwendbar. Wegen der häufigen Verwendung von CORBA als Kommunikationsbasis von EJB-Produkten ist auch die Interoperabilität auf dieser Ebene gewährleistet. Insgesamt sind die CORBA- und EJB-Container somit sehr ähnlich. Die enge Verwandtschaft hat auch Synergieeffekte zur Folge, sobald Produkthersteller ihre bisherigen EJB-Produkte zu vollständigen CCM-Implementierungen erweitern wollen.

Component Lifecycle Support

CCM Container sowie auch EJB Container bieten Funktionalitäten zur Verwaltung des Lebenszyklusses einer Komponente. Sie bieten den Komponentenentwicklern darüber hinaus,

mit Hilfe der im Deployment Deskriptor festgelegten Servant Lifetime, direkten konfigurierbaren Einfluss auf die Verwaltung des Lebenszyklusses einer Komponente.

Resource Management

EJB implementiert Strategien zur Verwaltung von wertvollen Ressourcen, diese Strategien sind nicht explizit Teil der Spezifikation, sondern finden sich in den einzelnen Dienstschnittstellen (JDBC Resource-Pooling) wieder.
Für das CCM Container Modell wird die Nutzung von Resource Pooling für externe Ressourcen nicht erwähnt. Es beruht auf CORBA Implementationen, so dass die Nutzung einer entsprechenden (nicht standardisierten) CORBA Implementation denkbar ist.

Transactions Management

Sowohl bei EJB als auch bei CCM Container haben Komponentenentwickler die Möglichkeit, Transaktionen rein deklarativ zu nutzen oder diese selbst zu verwalten. Bei deklarativer Programmierung werden im Deployment Deskriptor entsprechende Attribute gesetzt.

EJB Container	CCM Container
TX_REQUIRES_NEW	REQUIRES_NEW
TX_REQUIRED	REQUIRED
TX_SUPPORTS	SUPPORTS
TX_NOT_SUPPORTED	NOT_SUPPORTED
TX_MANDATORY	MANDATORY
TX_BEAN_MANAGED	NEVER

Tabelle 2: EJB/CCM Transaktionsattribute der Deployment/ Komponentendeskriptoren

EJB setzt bei programmatischen Transaktionen die JTA/JTS Schnittstelle ein.
Das Transkationsverhalten eines CCM Containers ist identisch mit dem Verhalten eines EJB Containers.

Persistence Services

Die beiden Technologien unterstützen die Möglichkeiten containerbasierter (und auch komponentenbasierter) Persistenz für bestimmte Komponentenkategorien.
Der Persistenzdienst des CCM Containers unterstützt allein Process- und Entity-komponenten, die EJB-Entwickler können „Container Managed Persistence“ oder „Bean Managed Persistence“ verwenden.

EJB Container	CCM Container
Bean Managed Persistence	Entity
Container Managed Persistence	Process

Tabelle 3: Komponentenkategorien mit containerbasierter Persistenzunterstützung

Security Services

Die beiden Spezifikationen (EJB/ CCM Container) besitzen ein rein deklaratives Sicherheitsmodell in welchem Rollen und Rechte für den Zugriff auf Methoden einer Komponente im Deployment Deskriptor festgelegt werden.

Event/Notification Services

Die Notification/Events in CCM Spezifikation basieren auf den Funktionen des CORBA Notification Service und nutzen dabei die ContainerAPI – Event, Policies werden dabei über den Deployment Deskriptor festgelegt.
EJB System unterstützt asynchrone Methodenaufrufe erst ab der EJB 2.0 Spezifikation mit Hilfe von Message Driven Beans.

Tabellarische Zusammenfassung:

	EJB Container	**CCM Container**
Component Lifecycle Support	Container Managed	Component Deskriptor Dependant
Resource Management	JDBC Resource Pooling	---
Transaction Management	Declarativ or programmatic (JTA/JTS)	Declarativ or programmatic (CORBA Transaction Service)
Persistence Services	Declarativ or programmatic (JDBC,SQLJ)	Declarativ or programmatic
Security Services	Declarativ	Declarativ or programmatic (CORBA Security Service)
Event/Notification Services	(Messaged Driven Bean from EJB 2.0 on)	CORBA Notification Service

Unterschiede in der Implementierung eines CC-Session Containers und eines EJBSession Containers in der Component Container Architecture.

Interoperating zwischen EJB- und CCM Container

Es ist möglich die EJBs und Corba components in derselben Applikation zu integrieren. Aufbauend auf diese Aussage, beinhaltet die CCM Spezifikation eine allgemeine (vorwärts und rückwärts) Mapping der EJB und CCM. Die Interaktion ist äußerst geschlossen, eine schmale
Brücke kann diese Lücke umschließen. Das schließt nicht nur die „method invocations“ ein, sondern auch Container, Factory, Finder und andere. Wird eine EJB in einen CCM-Container integriert, ist sie eine CCM component. Ein CORBA-Client sieht eine EJB als CCM component umgekehrt sieht ein EJB-Client eine Basic-CCM component als EJB.

*“**Basic Component** has only a single interface and my use transactions, security and simple persistence for a single segement. It relies upon the container to manage the construction of Corba Object references, and use only a single Thread.*
***An extended components** hat die Funktionalitäten der Basic component, plus multiple facets and advenced persistence assigned individually to multiple segments.”*

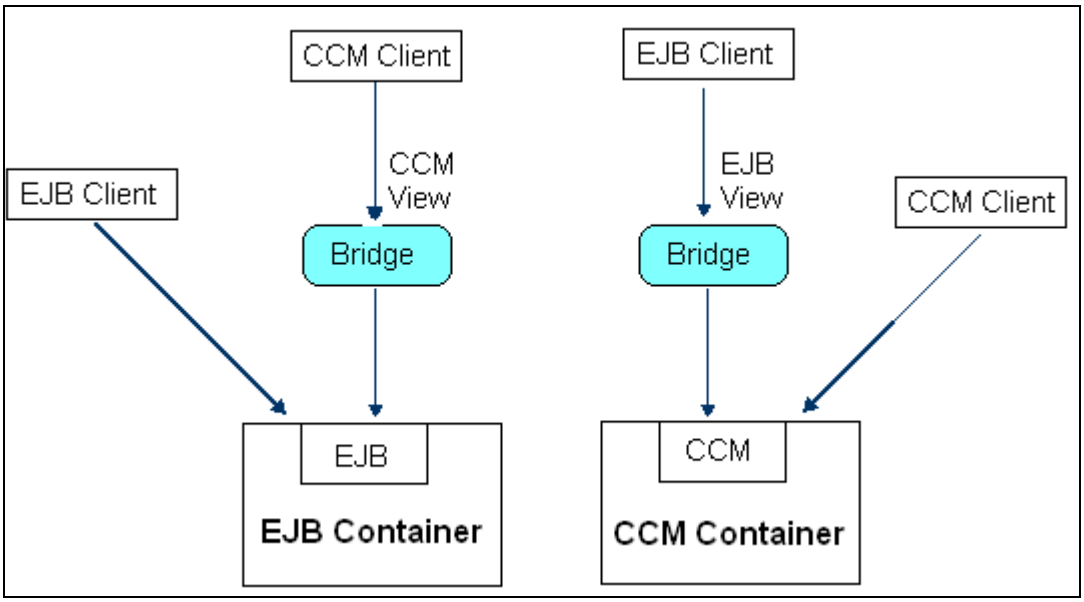

Abb. 6: Interoperating zwischen EJB- und CCM Container

5. Fazit

Der Container ist eine Ablaufumgebung für Komponenten. Er kümmert sich um Transaktionen, Sicherheit, Threading, etc., was als „technische Belange“ in der entsprechenden Domäne identifiziert wird. Der Komponentenentwickler muss also nicht mehr alles selbst implementieren.
Für die Realisierung mehrschichtiger Anwendungen ist es erforderlich, dass die Komponenten im Container über das Netz von Clients oder anderen Komponenten aus erreichbar sind. Ein Komponentenbus transportiert Operationsaufrufe über das Netz unter Benutzung der traditionellen OO-RPC Mechanismen wie CORBA/IIOP, RMI/JRMP oder DCOM/RPC.
Dabei können Ressourcenprobleme entstehen, da nicht zu jeder Zeit alle Instanzen einer Komponente aktiv im Speicher des Containers sind. Ein Container arbeitet normalerweise mit ***virtuellen Instanzen:*** Die logische Identität einer Komponente und die physikalische Komponenteninstanz werden konzeptionell getrennt. Der Client (bzw. eine als Client tätige andere Komponente) kommuniziert üblicherweise nur mit dem Komponentenproxy, dadurch kann der Container selbst die Identitäten wechseln oder Komponenten reaktivieren.
Der Container ist für den Lebenszyklus der Komponenten zuständig. Damit der Container eine Komponente in ihrem Lebenszyklus steuern kann, muss sie dem Container ***Lebenszyklusoperationen*** zur Verfügung stellen.
Der Container ruft diese auf, um zum Beispiel einen Wechsel der logischen Identität einer Instanz herbeizuführen, oder sie z.B. in einen persistenten Speicher auszulagern.
Ein Zusammenwachsen von CORBA mit Enterprise JavaBeans ist wahrscheinlich, dabei ist es fragwürdig, ob sich das CCM gegenüber EJB durchsetzen wird.

Literatur

[JH&FJ]…..... Hofmann J. ; Jobst F. & Schabenberger R. : Programmieren mit COM und CORBA. Hanser Verlag 2001
MaWu…… Wutka, M. : J2EE Developper’s Guide
2002 by Markt + Technik Verlag

ToLa……...Langer, T. : Verteilte Anwendungen mit Java
2002 by Markt + Technik Verlag

Wang_2000....Wang, Nanbor, Schmidt, Douglas C., O´Ryan, Carlos: An Overview of the CORBA Component Model. Eingereicht als ein Kapitel in: Component-Based Software Engineering: Putting the Pieces Together, George Heineman und Bill Councill, Addison-Wesley (2000)

HuBerlin……http://www.informatik.huberlin.de/Institut/struktur/systemanalyse/diplom/musk e03/x348.html [Aufruf: 10.06.03]

[JoSi].....Jon Siegel: Corba 3 fundamentals and Programming.
2000, 2th edition by John Wily & Sons

Sun00.....Enterprise JavaBeans
http://java.sun.com/products/ejb/docs.html

Sun01.....Java 2 Platform Enterprise Edition BluePrints
http://java.sun.com/j2ee/blueprints/index.html

DnsPei……Denninger, S. und Peters, I. : Enterprise JavaBeans
2000 by Addison-Wesley Verlag.