Technische Universität Ilmenau

Fakultät für Informatik und Automation
Institut Praktische Informatik

HAUPTSEMINAR
im WS 2002/2003

Das CORBA Component Model

CORBA Components vs. Enterprise Java Beans

vorgelegt von	:	Thomas Havemeister
geboren am	:	21.09.1978
E-Mail Adresse:	:	thomas@havemeister.net
Telefon:	:	03677 / 2059231
Studiengang	:	Wirtschaftsinformatik/M98
Vertiefung	:	Telematik
Matrikel-Nr.	:	27376

Betreuender Hochschullehrer	:	Dipl.-Inf. Thorsten Strufe
Abgabetermin	:	18.02.2003

Ilmenau, 1. Februar 2003

Inhaltsverzeichnis

Abkürzungsverzeichnis **iii**

Tabellenverzeichnis **iv**

Abbildungsverzeichnis **v**

1 Einleitung **1**

2 Komponententechnologien im Wandel **2**

2.1 Überblick 2

2.2 Defizite bisheriger Modelle 5

3 Das CORBA Component Model **6**

3.1 Begriffsbestimmung 6

3.2 Architekturkonzepte 7

3.3 Komponentenarten des CCM 9

3.3.1 Service Components 11

3.3.2 Session Components 11

3.3.3 Process Components 12

3.3.4 Entity Components 12

3.4 Einordnung in die Gesamtarchitektur 13

4 CORBA Components & Enterprise Java Beans **15**

4.1 Überblick zur EJB Spezifikation 15

4.2 Klassifikationsansätze 18

4.3 Gegenüberstellung beider Technologien 19

5 Fazit **22**

Literaturverzeichnis **23**

Erklärung über Hilfsmittel **25**

Abkürzungsverzeichnis

API	Application Programming Interface
B2B	Business to Business
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
EJB	Enterprise JavaBeans
ERM	Entity Relationship Model
GIOP	General Inter-ORB Protocol
GUI	Graphical User Interface
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
J2EE	Java2 Enterprise Edition
JDK	Java Development Kit
JMS	Java Messaging Service
JNDI	Java Naming & Directory Interface
LAN	Local Area Network
OMG	Object Management Group
ORB	Object Request Broker
POA	Portable Object Adapter
PSS	Persistent State Service
WAN	Wide Area Network

Tabellenverzeichnis

3.1 *Übersicht der CCM-Modelle* 15
4.1 *Klassifikationsansätze* 19
4.2 *Gegenüberstellung beider Technologien* 21

Abbildungsverzeichnis

3.1 *Skizze einer CORBA Component [CCM02]* 10
3.2 *CCM Architektur im Überblick* 13
4.1 *EJB Architektur im Überblick* 16

1 Einleitung

Die heutige Softwareentwicklung hat sich unlängst von dem Desktop auf den Server verlagert. Im Mittelpunkt stehen komplexe verteilte Systemumgebungen, mit äußerst hohen Connectivity- und Portabilitätsansprüchen.[1] Dabei werden Entwickler oft mit dem Problem konfrontiert, aufwändige IT-Infrastrukturen verbunden mit mehrdimensionalen Anforderungen wie Sicherheit, Unabhängigkeit und Asynchronität zu realisieren. Ein möglicher Ansatz wäre die vollständige Eigenentwicklung, um so die komplexen Integrationsaufgaben selber zu lösen. Der große Nachteil ist zweifelsohne der immense Kosten- und Zeitaufwand der bei diesem Ansatz unweigerlich entsteht. Demgegenüber steht die Verpflichtung gegenüber dem Kunden, den hohen Entwicklungsaufwand mit der Abbildung der Geschäftslogik zu rechtfertigen. Ein Großteil des Aufwandes steckt damit nicht länger in der Umsetzung der Kundenwünsche, sondern bei der Entwicklung einer geeigneten IT-Infrastruktur.

Das Ziel ist also der unternehmensweite Einsatz einer tragfähigen Technologie und die Steuerung kritischer Geschäftsprozesse mit Hilfe eines integrierten Gesamtkonzeptes, welches auch zukünftigen Anforderungen gerecht werden soll. Eine Lösung könnte der Einsatz von so genannten Middle-Tier-Komponententechnologien sein. Also eine Art „standardisierte Drehscheibe“, welche beliebige Multi-Tier Systeme im Kontext des Unternehmens verbindet.

Was ist jedoch wirklich neu an diesem Konzept? Ein wesentlicher Aspekt bei der Erstellung von verteilten Anwendungen ist die Tatsache, dass Entwickler sich primär an Diensten in unteren Schichten der Gesamtarchitektur aufhalten. Abschreckend wirkt auch die enorme Komplexität die mit dieser Aufgabe verbunden ist. So müssen neben den rein technischen Aspekten (Verbindungsaufbau, Datenübertragung, Ereignisbehandlung, Persistenz, ...) auch eine ganze Reihe von organisatorischen Problemen (Sicherheitsaspekte, Mehrbenutzerfähigkeit, Transaktionsbearbeitung, ...) gelöst werden.[2]

Im Rahmen dieser Arbeit soll daher ein Ansatz der Object Management Group (OMG) untersucht werden, welcher aus der Common Object Request Broker (CORBA)-Idee der letzten Jahre hervorgegangen ist. Als Bestandteil der CORBA 3.0 Spezifi-

[1] Vgl. [Stahl00] S.26 ff.
[2] Vgl. [Bartlett01]

kation[3], könnte das so genannte CORBA Component Model (CCM)[4] eine geeignete und universelle Komponententechnologie sein, um den angesprochen Problemen zu begegnen.

Für eine richtige Einordnung der Thematik in die Welt der Komponententechnologien wird in Kapitel 2 ein allgemeiner Überblick erarbeitet. Dabei wird zunächst bewusst auf Details spezifischer Technologien verzichtet. Diese werden später nur zum Zweck der Darstellung typischer Defizite bisheriger Modelle herangezogen.

Nach dieser Grundlagenbildung soll im Kapitel 3 das CCM beschrieben werden. Dazu werden Grund- und Architekturkonzepte vorgestellt und Begriffe definiert. Für ein besseres Gesamtverständnis werden zusätzlich die unterschiedlichen Modellsichten des CCM dargestellt und die vier wichtigsten „CORBA Components“ erläutert, welche gewissermaßen die Bausteine der Architektur symbolisieren.

Im anschließenden Kapitel wird ein Vergleich zwischen dem CCM und dem bereits bewährten Enterprise Java Beans (EJB)[5]-Ansatz von Sun Microsystems gezogen. Zu diesem Zweck wird vorher ein Überblick zur EJB Spezifikation erarbeitet und mögliche Klassifikationsansätze für einen Vergleich vorgestellt. Im letzten Teil folgt dann eine tabellarische Gegenüberstellung der beiden Technologien.

2 Komponententechnologien im Wandel

2.1 Überblick

Mit den derzeitig am Markt erhältlichen Werkzeugen für die Softwareentwicklung (GUI Builder, Editoren, Frameworks), ist es mittlerweile spürbar einfacher aufwändige Präsentationskomponenten zur Verfügung zu stellen, als es noch vor wenigen Jahren der Fall war. Damit wird dieser Teil der Arbeit gewissermaßen zur „Pflichtübung“ für Anwendungsprogrammierer. Einher geht jedoch eine deutliche Verschiebung der Betrachtung vom Client hin zum Server. Dies hat mehrere Gründe, welche aus dem veränderten Verhalten der Stakeholder (Geschäftsinteressierte) resultieren. Als Konsequenzen dieser Entwicklung für die IT-Branche nennt [Merle02] folgende Punkte:[6]

[3] Vgl. CORBA 3.0.1 in [CORBA02]
[4] Vgl. CORBA Component Model in [CCM02]
[5] Vgl. Enterprise Java Beans 2.0 in [EJB01]
[6] Vgl. zum folgenden Absatz [Merle02] S.6

- **Time-to-Market.** Im heutigen Geschäftsalltag kann es ein entscheidender Erfolgsfaktor sein, wenn man selber, früher als die Konkurrenz, in der Lage ist mit innovativen Produkten auf den Markt zu treten. Dies erfordert jedoch eine deutliche Produktivitätssteigerung während der Implementierungsarbeiten. Deshalb ist es gerade an dieser Stelle notwendig, die Stärken von Server-orientierten Ansätzen auszunutzen, um schnell lauffähige Ergebnisse vorführen zu können.

- **Baukastenartige Softwareentwicklung.** Die erwähnte Problematik „etwas schneller zu entwickeln“ basiert ganz wesentlich auf der Idee, hochwertige Produkte mit „Fertigbauteilen“ auszustatten. Dabei steht der Grundsatz im Mittelpunkt, bereits entwickelte Module, Teilprogramme oder Funktionsgruppen beliebig zu verbinden. Die Idee ist an sich nicht neu und wurde bereits durch die mächtigen Mittel objektorientierter Sprachen formuliert. Neu ist hingegen, die lose Kopplung der so genannten Komponenten[7].

- **Kundennutzen.** Das primäre Ziel aller Bemühungen sollte ein hoher Kundennutzen sein. Gerade mit einer umfassenden Client/Server-Architektur wird es möglich, die Vorteile verteilter Systeme auszunutzen.

In jüngster Zeit entfalten sich jedoch eine Reihe neuer Problemfelder und Herausforderungen, die ein Anwendungsentwickler ohne eine umfassende Technologie nur schwerlich, wenn nicht sogar unmöglich lösen kann. Diese sollen im Folgenden überblicksartig skizziert werden:[8]

- **Mehrbenutzerfähigkeit.** Ein System, welches mehrere Benutzer erlaubt, muss auf unterschiedlichste Weise Koordinations- und Verteilungaufgaben wahrnehmen können. Dazu gehören Aspekten wie Multithreading, Transaktionorientierung oder Benutzerverwaltung und die damit verbundenen Sicherheitsfragen.

- **Skalierbarkeit.** Wenn Unternehmen wachsen (bzw. rationalisieren) ist es oft notwendig, die bestehenden Computeranlagen dem Umfeld anzupassen. Zum Beispiel bedeutet ein größerer Kundenkreis im Zweifel eine Ausweitung der Kapazitäten.

[7] Vgl. Kapitel 3.1 dieser Arbeit zum Begriff „Komponenten“
[8] Vgl. zum folgenden Absatz [DenPet02] S.13 ff.

- **Verfügbarkeit.** In kritischen Phasen, wo hohe Ansprüche an die Stabilität des Systems gestellt werden, ist es oft besonders wichtig, entsprechende Vorbereitungen zu treffen, um Engpäße zu vermeiden und die Arbeitsfähigkeit sicherzustellen.

- **Verbindung mit der Außenwelt.** Schnittstellen zum System gewähren eine erhöhte Zugriffsvielfalt. Bei hohen Ansprüchen an die Erreichbarkeit des Systems (Internet, Mobilfunk,...), müssen besondere Mechanismen berücksichtigt werden.

- **Integration mit anderen Anwendungen.** Die Integrationsfähigkeit gilt als äußerst problematische und dennoch besonders wichtige Anforderung an moderne IT-Systeme. Der wohl wichtigste Aspekt betrifft die Anbindung an die zum Teil stark heterogenen Systemlandschaften der Unternehmenspartner (Enterprise Application Integration).

- **Konfigurierbarkeit.** Software, die in unterschiedlichen Standorten zum Einsatz kommt, wird möglicherweise mit geänderten Rahmenbedingungen (Sprache, Hardware, etc.) konfrontiert. Darauf muss ein System vorbereitet sein, da ein Kompiliervorgang oftmals einen enormen Kosten- und Zeitaufwand mit sich zieht und ein entsprechendes Risikopotential für neue Fehler enthält.

Mit dem Aufkommen der komponentenbasierten Ansätze entstand auch der Bedarf nach einem robusten, standardisierten System, welches in der Lage sein sollte umfangreiche Geschäftsobjekte und deren prozessorientierten Abläufe zu modellieren bzw. zu persistieren. Komponententechnologien sollten daher eine adäquate Lösungstrategie zur Verfügung stellen, die bei speziell technischen Schwierigkeiten, wie es zum Beispiel bei der Abbildung von relationalen Datenbankmodellen auf objektorientierte Modelle der Fall ist, unterstützend wirkt. Damit verbunden sind auch grundlegende Dienste wie Lebenszyklusmanagement, Zugriffsmanagement, Verteilungs- und Anbindungsdienste, Namens- und Verzeichnisdienste sowie ein passendes Sicherheitsmanagement. Eine genauere Erklärung und Darstellung dieser Aspekte der Komponententechnologien wird in Kapitel 4.2 vorgenommen.

2.2 Defizite bisheriger Modelle

Bis dato gibt es eine Reihe von Technologien die den Anspruch erheben, die bereits genannten Herausforderungen, welche bei der Entwicklung verteilter Systemlandschaften unweigerlich entstehen, ganz oder teilweise zu bewältigen. Bei der Betrachtung fällt jedoch auf, dass insbesondere auf der Ebene von Geschäftsobjekten und Datenbanksystemen derzeit ohne universellen Standard gearbeitet wird. Die alleinige Existenz eines objektorientierten Kommunikationsmechanismus reicht nicht aus, um die komplexen Integrationsprobleme und Nebenbedingungen zufriedenstellend zu lösen. Gemeint sind Standards wie das bereits erwähnte CORBA[9], die Remote Method Invocation (RMI)[10] als Bestandteil von Java und das Distributed Component Object Model (DCOM)[11] seinesgleichen Bestandteil in diversen Microsoft-Produkten. Derartige Modelle für das „Distributed Object Computing“ präsentieren sich jedoch ohne eine umfassende Gesamtarchitektur, obwohl sie vorrangig für den Einsatz im Enterprise-Umfeld entworfen wurden.

Unter Argumentation des im nächsten Kapitel vorgestellten CCM, beschreiben [MarvMerl01] folgende typischen Defizite bisheriger Modelle:[12]

- **Konfiguration und Verteilung.** Die Konfiguration und Wartung von Applikationen wird nur unzureichend unterstützt. Es fehlen sowohl Standards für die Installation und Ausführung als auch normierte Ansätze für die Verteilung von Applikationen oder deren Komponenten.

- **Explizite Programmierung.** Strukturelle Anforderungen, wie Hierarchien oder Topologien müssen explizit programmiert werden. Es gibt nur selten normierte Modelle für die Konzeption und Lösung solcher strukturellen Fragestellungen.

- **Fachobjekte.** Nicht selten werden die „Business Objects“ mit technischen Aspekten vermischt, so dass es öfter zu Kompetenzproblemen bei Design und der Implementierung kommt und diese nicht selten zu Mißverständnissen und damit zu kostspieligen Fehlern führen.

[9] Vgl. früheres CORBA Rev.2.6 in [CORBA01]
[10] Vgl. Remote Method Invocation (RMI) in [RMI01]
[11] Vgl. Distributed Component Object Model in [DCOM97]
[12] Vgl. zum folgenden Abschnitt [MarvMerl01] S.2 ff.

- **Packaging und Deployment.** Komponenten werden nach speziellen Regeln durch die Entwickler „verpackt“ und ausgeliefert. Effektiver lässt sich dieser Prozess gestalten, in dem man ihn nach einer Vorschrift befolgt. Dieser Vorgang gestaltet sich bisweilen stark unterschiedlich. Standards könnten aber dazu beitragen Durchlaufzeiten zu verkürzen und somit Kosten zu sparen.

Zusammenfassend lässt sich also festhalten, dass bisherige Modelle in ihrer Komplexität nicht alle Bereiche abdecken können, d.h. zum Teil auch gar nicht sollen. Klar wird jedoch, dass der Bedarf bzgl. einem umfassenden bzw. vollständigen Ansatz nicht von der Hand zu weisen ist.

3 Das CORBA Component Model

3.1 Begriffsbestimmung

Was ist das CORBA Component Model? Eine Definition von [Raj99] lautet:

„Das CCM ist eine Spezifikation für die Erstellung von serverseitigen, skalierbaren, sprachneutralen, transaktionsorientierten, mehrbenutzerfähigen und sicheren Enterprise Applications.“[13]

Das CCM ist Teil der CORBA 3.0 Spezifikation und erweitert das ursprüngliche CORBA Objektmodell durch neue Features und Services. Dadurch steht ein serverseitiges Modell für die Erstellung von CORBA-basierten Anwendungen zur Verfügung. Im Mittelpunkt der Betrachtungen des CCM steht das Konzept der Komponente. Für diesen Begriff gibt es in der Literatur eine Vielzahl von Definitionen. An dieser Stelle soll kurz auf einige wichtige Definitionen eingegangen werden.

[Szyperski98] beschreibt eine Softwarekomponente als eine geordnete Einheit mit definierten Schnittstellen und ausschließlich kontextbezogenen Abhängigkeiten zu anderen Programmteilen. Eine solche Einheit kann unabhängig verteilt werden und ist das Ergebnis von den Dienstleistungen eines Dritten. Nach [Booch87] handelt es sich um wiederverwendbare Softwarekonstrukte welche intern durch eine hohe Kohäsion gekennzeichnet sind und nach außen mit einer weitgehend losen Kopplung

[13] Vgl. [Raj99] für das englische Original

auftreten. Diese Abstraktion ist von nutzen, weil auch [Griffel98] eine Komponente bezeichnet als „ein Stück Software, welches klein genug ist, um es in einem Stück erzeugen und pflegen zu können, groß genug ist, um sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen, sowei mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten“. Komponenten sind noch durch eine Reihe anderweitiger Eigenschaften charakterisierbar. Oft genannt werden Schlagwörter wie Eigenständigkeit, Grobkörnigkeit, „Black Box“-Verhalten, Schnittstellenorientierung oder Konfigurierbarkeit. Für die weiteren Ausführungen, sollte jedoch die bisherige Definition bzw. Einordnung des Begriffs genügen.

Die Softwarekomponente ist die Schlüsseltechnologie im Rahmen des CCM. Die Spezifikation der OMG ist primär eine Architektur zur Definition von Komponenten. Ein Container konstruiert eine Laufzeitumgebung, mit deren Hilfe die, nach speziellen Schema entwickelten, CORBA Components Systemabhängigkeiten wie Programmiersprachen, API-Zugriffe und Datenbanktransaktion verbergen können. Zu diesem Zweck bietet der Container ein Framework an, mit dessen Hilfe verschiedene infrastrukturelle Dienste den CORBA Components zur Verfügung gestellt werden.[14] Mit diesem umfassenden Ansatz der OMG steht eine industriell abgesegnete sowie standardisierte Umgebung zur Verfügung, mit der Entwickler über ein einheitliches System Komponenten implementieren, verteilen, konfigurieren und Teile des Codes automatisch erzeugen können. Dabei wurde ein altes OMG-Prinzip konsequent beibehalten: die Sprachunabhhängigkeit. Aufgrund dieser Vorgabe ist die Spezifikation grundsätzlich straffer und weniger ausschweifend konzipiert, als etwa die Java-basierte EJB-Spezifikation.[15]

3.2 Architekturkonzepte

Grundlage des Systems bildet ein Application Server (Applikationsserver). Darunter versteht man einen Server für Anwendungsprogamme, welche im LAN bzw. WAN zur Verfügung stehen und von vielen Clients benutzt werden können. Im Gegensatz zu einer herkömmlichen Client-Server Beziehung, erweitert der Application Server die Beziehung des Server zu seiner Umwelt. Die daraus resultierende Three-Tier-

[14] Vgl. [Bartlett01]
[15] Vgl. [Bartlett01]

Architecture[16] führt dazu, dass die Benutzerschnittstelle des Client-Programms auf die bereitgestellte Anwendung, d.h. die Arbeitslogik des Servers zugreift und dieser wiederum eine Datenbank benutzt, welche für die Informationssicherung und -bereitstellung verantwortlich ist.[17]

Für eine effiziente serverseitige Abarbeitung der zu erwartenden Vielzahl von Anfragen durch die Clients, steht dem Application Server eine spezialisierte Ausführungsumgebung zur Verfügung. Diese soll multitaskingfähig sein und die zu tätigenden Hardwarezugriffe mit einer optimierten Lastverteilungsstrategie verbinden. Zudem sorgt eine solche Laufzeitumgebung für die Sichtbarkeit der Anwendungen und Dienste, welche von den Clients benutzt werden dürfen.

Im CCM wird diese Umgebung *CCM Container* genannt. Dieser fungiert als Schnittstelle zwischen der Außenwelt[18] und den zur Benutzung freigegebenen Komponenten. Diese „Schutzhülle für Komponenten“ ermöglicht eine ganze Reihe serverseitiger Verwaltunsaufgaben. Ein potentieller CCM-Client greift daher niemals direkt auf eine so zur Verfügung gestellte *CORBA Component*. Jeder Zugriff erfolgt zunächst über eine Art „Wrapper“(Hülle/Mantel) welcher erst dann die containergenerierten Methoden für den eigentlichen Aufruf an die CORBA Component anstößt. Dieses als „Method Invocation“ bezeichnete Zugriffsverfahren soll vorallendigen unkontrollierte direkte Zugriffe vermeiden, welche sonst auch die besten Lastverteilungsstrategien zwecklos machen würden. Der eigentliche Abfangmechanismus des Containers wird auch als „Interception-Mechanismus“ bezeichnet. So fängt der Container, und nicht die Komponente, die Aufrufe von Clients ab und arbeitet die Funktionalitäten ab, in dem er auf die eigentlichen Komponenten die anstehenden Aufgaben deligiert.[19]

Ein Ziel bei der Entwicklung der CCM-Spezifikation war es, möglichst heterogene Anwendungsfälle und Problemstellungen zu berücksichtigen. Aus diesem Grunde

[16] Natürlich kann an dieser Stelle sogar von einer n-tier-Architecture gesprochen werden, wenn die Anwendungslogik es erfordern sollte mehrere Quellen als nur eine simple Datenbank in die Lösung des Problems einzubeziehen. Als wesentlicher Vorteil eines Application Server wird oft die erhöhte Konnektivität angeführt, welche das Problem von Insellösungen durch ein zentrales „alles-verbindendes“ System umgehen soll.

[17] Im Folgenden wird ein serverseitiges Programm als Komponente bezeichnet, da es das Ziel ist Anwendungen möglichst lose gekoppelt den anfragenden Clients zur Verfügung zu stellen.

[18] Im engeren Sinne eigentlich nur zum verwendeten Application Server.

[19] Vgl. [Stahl00] S.2 ff.

entschied man sich, zwei unterschiedliche Arten von Containern einzuführen:[20]

- **Transiente Container.** Abgebildet werden „temporäre“ bzw. „flüchtige“ Container. Anwendungsgebiet sind nicht-persistente (transiente) Komponenten, deren Status nicht gespeichert werden muss und nur für Anfragen oder einfache Anwendungen vorgesehen sind.

- **Persistente Container.** Ermöglicht wird eine sichere Umgebung für Komponenten, welche den eigenen Status abspeichern und so bei späteren Anfragen über deren eigenen Lebenszyklus hinweg wieder auf die Daten zugreifen. Das CCM unterscheidet auch zwischen *Session Containers* und *Entity Containers*. Der Unterschied wird bei genauerer Betrachtung der verschiedenen, zur Verfügung gestellten Komponenten deutlich, die im Folgenden beschrieben werden.

3.3 Komponentenarten des CCM

Wie bereits anhand der CCM Container angedeutet, gibt es auch zwischen den Komponenten, den so genannten *CORBA Components*, Unterschiede. Zur Spezifikation der Komponenten kommt die Hochsprache *Component IDL*[21] zum Einsatz. Dabei handelt es sich um eine Weiterentwicklung der Schnittstellenbeschreibungssprache Interface Definition Language (IDL).

Die Beschreibung der Schnittstellen erfolgt nach einem speziellem Schema. Spezifiziert werden müssen sämtliche Schnittstellen, welche Dienste anbieten bzw. deren Nutzung durch andere Komponenten vorgesehen ist. Dabei vertraut das CCM Paradigma dem bewährten sprachneutralen Ansatz der IDL. Dieser eignet sich sowohl zur Generierung des Codes, welcher für die Integration zwischen Container und Komponente benötigt wird, als auch dem Teil, welcher für die Kommunikation zwischen Client und Server verantwortlich ist.

In Abbildung 3.1 wird schematisch eine CORBA Component skizziert. Dabei werden neue, CCM-spezifische Begriffe eingeführt, die im Folgenden detailiert werden:[22]

[20] Vgl. [CCM02] Kapitel 4.2.2.1, S.141
[21] Vgl. [CCM02] Kapitel 2, S.71
[22] Vgl. zum folgenden Abschnitt [Raj99]

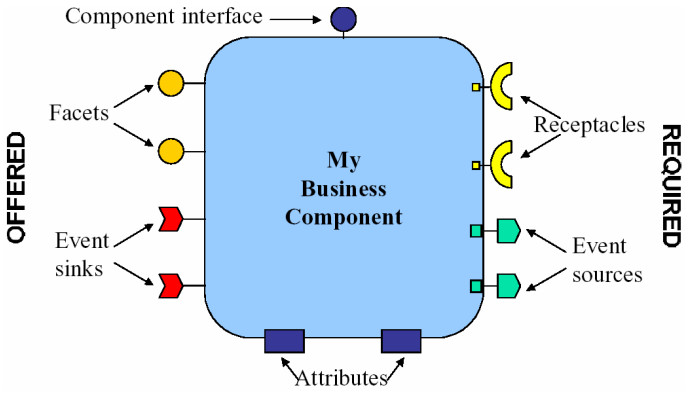

Abbildung 3.1: *Skizze einer CORBA Component [CCM02]*

- **Facets.** Schnittstellen, welche zum Export von Funktionalitäten vorgesehen sind, bezeichnet man als *Facets*. Diese Applikationsschnittstellen definieren explizit Dienste (Provides, Supports) welche zur Benutzung von Klienten oder alternative Komponenten vorgesehen sind.

- **Receptacles.** Wenn CORBA Components innerhalb einer Container-Domäne gekoppelt werden sollen, setzt man Receptacles ein. Dabei handelt es sich um interne Verknüpfungspunkte bzw. Eingänge, mittels denen Komponenten Zugriffspfade zur Verfügung stehen.

- **Event Sinks & Event Sources.** Oft kommt es vor, dass auf bestimmte Ereignisse passende Aktionen ausgelößt werden müssen. CORBA Components können sowohl auf Ereignisse reagieren (Event Sinks) aber auch welche erzeugen (Event Sources). Diese Prinzip ähnelt dem „Observer"-Design Pattern, welches oft in GUI-basierten Anwendungen eingesetzt wird. Event Sinks bzw. Sources ermöglichen auch ein Exception-Handling[23] für Komponenten.

- **Attributes.** Attribute bilden Aspekte, Eigenschaften oder Konfigurationen von Komponenten ab. Im objektorientierten Sinne handelt es sich dabei um Klassen-Member, deren Zugriff über Getter- und Settermethoden (bzw. Schnittstellen) erfolgt. Wie auch bei der Ereignisbehandlung können illegale Zugriffe

[23] Exceptions sind Ausnahmen, welche im Verlauf einer Abarbeitung von Befehlen auftreten können. Diese Form der Fehlerbehandlung hat sich seit dem Siegeszug der objektorientieren Sprachen etabliert und stellt ein flexibles System dar, wie man angemessen auf potentielle Fehler reagieren kann.

oder Operationen auf Attribute, Exceptions zur Folge haben und damit eine Fehlerbehandlung erzwingen.

Klienten (*CCM Clients*), welche CORBA Components nun für deren Operationen benutzen wollen, suchen zunächst die passende Komponente über den Namens- und Suchdienst *CosNaming*, welcher durch die CCM Laufzeitumgebung zur Verfügung gestellt wird. Anschließend erzeugt bzw. referenziert der Klient die gewünschte Komponente und erhält einen Zugriff über das *Container Home Interface*.

Neben den hier erwähnten Elementen verfügen sämtliche Komponenten über gemeinsame Eigenschaften (nicht in der Abbildung). Diese Äquivalenzschnittstelle stellt sicher, dass Komponenten einen gemeinsamen Grundsatz an Operationen mitbringen. Nur so kann die Gleichbehandlung unterschiedlichster Komponenten durch den CCM Container gewährleistet werden. Welche Komponentenarten dabei auftreten können, soll in den folgenden Unterkapiteln dargestellt werden.[24]

3.3.1 Service Components

Service Components sind einem bestimmten Client fest zugeordnet. Daher hat nach der Instanzierung nur *ein* Client Zugriff auf diese Komponente und kann nicht gleichzeitig für einen anderen Client Anfragen nachkommen. Die Komponente wird nach Beendigung der Aufgabe wieder dereferenziert. Die Service Component besitzt keinen Zustand, sie ist transient. Ihr Lebenszyklus ist beschränkt auf die Abarbeitung einer konkreten Funktionsanfrage (single method call). In objektorientierten Sprachen entspricht dies einem statischen Funktionsaufruf ohne unmittelbare Auswirkung auf die eigentliche Klasse.

3.3.2 Session Components

Eine Session Component ist ebenso wie die Service Component nur einem Client zugeordnet. Auch sie ist transient und kann mangels Identifikator niemals gleichzeitig mehreren Clients zur Verfügung stehen. Der entscheidende Unterschied dieser Komponente liegt in der Instanzierung und Benutzung. Eine Session Component wird durch einen Client angelegt (create), nach Bedarf benutzt (multiple method call), im Zustand verändert und durch den selben Client wieder freigegeben (destroy). Der

[24] Vgl. zu den folgenden Unterkapiteln [Raj99] und [Stahl00]

Zustand kann jedoch nach einem Neustart oder in einer späteren Sitzung (Session) nicht wieder hergestellt werden.

3.3.3 Process Components

Diese Komponentenart hat immer einen Status und ist persistent, d.h. die Zustände werden dauerhaft festgehalten. Die Umsetzung dieser Eigenschaft gelingt mit Hilfe eines Schlüssels, mit dessen Hilfe diese Komponente eindeutig identifiziert und von mehr als nur einem Clienten angefordert werden kann. Dieser Schlüssel unterscheidet sich von einem Primärschlüssel, da eine Process Component keine Entität oder einen Datensatz symbolisiert.

Ein gleichzeitige Nutzung durch mehrere Clients wird dennoch ermöglicht und erlaubt außerdem multiple Funktionsaufrufe.

3.3.4 Entity Components

Die wohl aufwendigste Komponente ist die Entity Component. Zum besseren Verständnis empfiehlt sich im Rahmen des CCM eine weitere Unterteilung des Persistenzbegriffs. Tatsächlich gibt es zwei Arten wie man Komponenten, d.h. deren Status, abspeichert. Den Vorgang zum persistieren kann die Komponente selbst übernehmen. Zu diesem Zweck programmiert man die Komponente direkt mit den dazu nötigen Anweisungen (hard-coded) und implementiert die erforderliche Äquivalenzschnittstelle. Diese Form der Persistenz heißt *Component-managed persistence*. Diese Aufgabe kann jedoch auch der CCM Container selber übernehmen. In diesem Fall spricht man von *Container-managed persistence*.

Hier wird die entscheidende Stärke der Komponententechnologien erst wirklich deutlich. Während die transienten Komponenten und die Process Components immer nur einen Ausschnitt der Fachlogik repräsentieren, können Entity Components, wie der Name schon andeutet, Entitäten im Sinne der Konzeptmodellierug bei Datenmodellen unmittelbar abbilden. Wie auch im Entity Relationship Model (ERM) können unter Benutzung der durch die Komponenten zur Verfügung gestellten Techniken(Facets, Receptacles), komplexe Beziehung unmittelbar modelliert werden. Mit Hilfe von Primärschlüsseln steht der Benutzung durch mehrere Clients nichts mehr im Wege. Der Container übernimmt selbstständig die dabei auftretenden Probleme hinsichtlich Transaktionenverarbeitung, O/R-Mapping und Konfliktverwaltung.

3.4 Einordnung in die Gesamtarchitektur

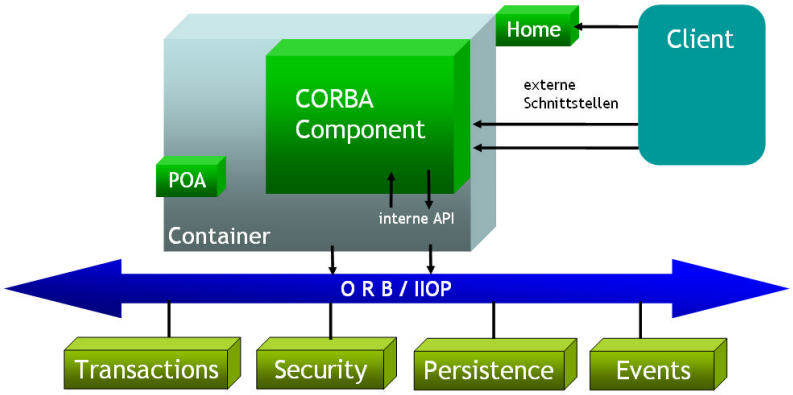

Abbildung 3.2: *CCM Architektur im Überblick*

Aus verwaltungstechnischer Sicht, verfügt die Gesamtarchitektur neben den bereits erwähnten Konstruktionsmerkmalen, noch über folgende Funktionsbereiche:

- **Object Request Broker (ORB).** Die Kommunikationsstruktur der Architektur basiert auf dem ORB. Dieses System stammt bereits aus früheren Ansätzen von CORBA und symbolisiert eine Art Datenbus. Der ORB stellt einen Mechanismus bereit, mit dem es möglich ist beliebige CORBA Objekte (ferner auch CORBA Components) anzufordern.[25] Ist ein solches Objekt einmal lokalisiert, steht es dem Aufrufer uneingeschränkt zur Verfügung. Mit Hilfe des ORB macht es nunmehr keinen Unterschied wo und wie dieses Objekt angefordert wurde. Auch die Kommunikation über das Internet-Protokoll wird mit Hilfe der Erweiterung Internet Inter-ORB Protocol (IIOP)[26] ermöglicht.[27]

- **Portable Object Adapter (POA).** Innerhalb der POA werden CORBA Objekte bzw. deren abstrakte Konzepte während ihrer individuellen Lebenszyklen zu Servants, d.h. zu konkreten Objekten in einem Adressraum. Die Verwaltung obliegt unmittelbar einer nach standardisierten Richtlinien[28] entwickelten POA Implementierung. Im Zuge der bereits erwähnten Interception-

[25] Vgl. [CORBA02] Kapitel 2.1, S. 48
[26] Vgl. [CORBA02] Kapitel 12.1.4, S.459
[27] Aber auch Protokolle unabhängig der typischen Internet-Protokolle werden berücksichtigt. Die CORBA Spezifikation legt sich in dieser Angelegenheit nicht fest und formuliert mit dem General Inter-ORB Protocol (GIOP) einen allgemeinen Ansatz.
[28] Vgl. [CCM02] Kapitel 4.2.4, S.142 ff.

Strategie, kümmert sich diese um die Aktivierung bzw. Passivierung von Komponenten. Das Ziel ist also die Gestaltung und Verwaltung eines „Lebensraumes“ für CORBA Components.

- **CORBA Services.** Mit den CORBA Services steht dem CCM ein reichhaltiger Satz an standardisierten CORBA-Diensten zur Verfügung. Dabei handelt es sich um Dienste, die mit bewährten Methoden Lösungsstrategien für diverse Problemstellungen zur Verfügung stellen. Zu nennen sind unter anderem der Transaktionsdienst (CORBA Transactions), das Sicherheitsmodell (CORBA Security), der Persistenzansatz (CORBA Persistence) und die Ereignisbehandlung (CORBA Events).

Eine grafische Einordnung der bisher genannten Konzepte findet sich in Abbildung 3.2.

Zum Abschluß empfiehlt sich ein Gesamtüberblick über den CCM-Teil der CORBA 3.0 Spezifikation. Das rund 500 Seiten umfassende Dokument[29] gliedert sich in fünf Modelle und einem Metamodell[30]. Diese sollen überblicksartig in der Tabelle 3.1 zusammengetragen werden.

	Beschreibung
Abstract Model	Interface - Design, Component Interface Definition Language (CIDL), Port-Definition (Multi-Interfaces)
Programming Model	Component Implementation Framework (CIF), (Non)Functional Class Design, Container Zugriffsfunktionen
Packaging Model	Definition der Packaging-Prozesse, Deskriptoren-Spezifikation durch die XML/DTD Grammatik Open Software Description (OSD)
Deployment Model	Prozessdefinition rund um das Deployment: installieren, einbetten, integrieren, pflegen, ...

[29] Vgl. CORBA Components - full specification [CCM02]

[30] Das verwendete Metamodell wird mit Hilfe des OMG-Standards Meta Object Facility (MOF) in [MOF00] beschrieben. Dabei handelt es sich um eine Meta-Metamodell Architektur, welche mit Hilfe der bekannten grafischen Konzepte aus der UML beliebige Sachverhalte abstrahieren kann.

Execution Model	Definition der Ausführungsumgebung, technische Aspekte der Container (Persistenz, Performance, ..), Lebenszyklusmanagement
Meta Model	MOF Metamodelle der verwendeten Konzepte, abstrahierte Darstellung der CCM Compoments (Facets, Receptacles, Sinks, ...)

Tabelle 3.1: *Übersicht der CCM-Modelle*

4 CORBA Components & Enterprise Java Beans

4.1 Überblick zur EJB Spezifikation

Gegenstand der folgenden Darstellungen ist die Enterprise JavaBeans Spezifikation in der Version 2.0[31]. Im Mittelpunkt steht die Spezifikation und nicht die konkrete Umsetzung durch Softwareprodukte wie IBM Websphere oder SunOne. Als Bestandteil der Java2 Enterprise Edition (J2EE) von Sun Microsystems stellt EJB den zentralen Baustein für die Entwicklung von verteilten Applikationen dar. Ein „Bean“ ist im Wortlaut von Sun eine Komponente. Oftmals wird der Begriff JavaBeans verwechselt mit Enterprise JavaBeans, obwohl es fundamentale Unterschiede gibt. Während JavaBeans visuelle Komponenten zur Manipulation durch Builder-Tools sind, versteht man unter Enterprise JavaBeans den eigentlichen Komponentenansatz im serverbasierten Unternehmensumfeld. Wie durch den Namen ersichtlich, basiert die gesamte Spezifikation auf der Programmiersprache Java. Daher wird auch intensiv auf die umfassenden Freiheitsgrade der Plattformunabhängigkeit bzw. virtuellen Maschine zurückgegriffen. EJB setzt dabei voll auf die Ideen von modernen komponentenbasierten Modellen und bietet auch eine breite Unterstützung bei ähnlichen bzw. verwandten Servermodellen wie zum Beispiel den Messaging-Diensten oder dem Transaction Monitoring. Besonders im Bereich der B2B-Application Integration steht mit der J2EE in Verbindung mit der EJB Spezifikation eine mächtiges und mittlerweile in weiten Kreisen etabliertes „Machwerk“ zur Verfügung.

Speziell konzipiert für den unternehmensweiten Einsatz und die Steuerung von kritischen Geschäftsprozessen, präsentiert sich der Ansatz heute mit mehr als nur

[31] Vgl. Enterprise JavaBeans 2.0 in [EJB01]

einer aufwändigen technischen Spezifikation. Enthalten sind mittlerweile auch Empfehlungen über ein Vorgehensmodell und eine dazu passende Rollenverteilung, für die im Entwicklungsprozess involvierten Parteien.

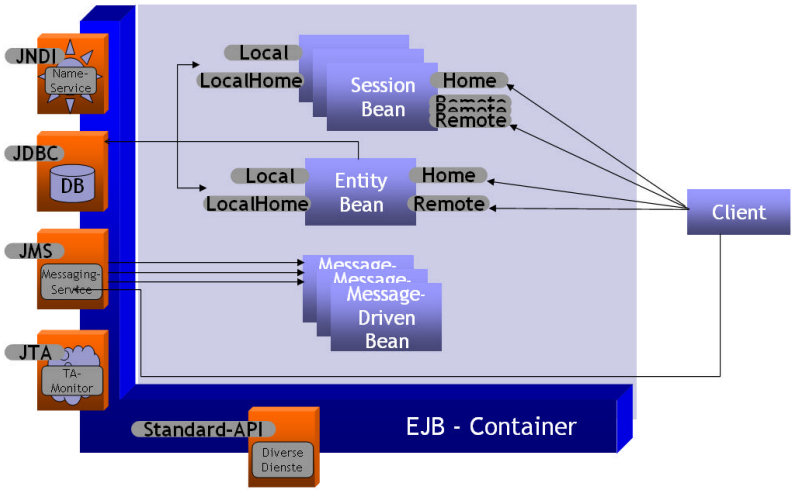

Abbildung 4.1: *EJB Architektur im Überblick*

Auch in dieser Komponententechnologie steht der Container als Standard- Programmierschnittstelle für Server-Anwendungen im Mittelpunkt. Für eine effektive Bearbeitung der anstehenden Aufgaben, steht innerhalb der Java Platform eine breite Palette an Services zu Verfügung. Die Wichtigsten sind in der Abbildung 4.1 skizziert und sollen kurz angesprochen werden.[32]

- **Standard API.** Seit Veröffentlichung von Java wurde das Java-Framework konsequent weiterentwickelt. Heute verfügt ein gängiges Java Development Kit (JDK) über zahlreiche Teilframeworks mit Tausenden von Klassen. Auf all diese fertigen Programmiermodelle kann der EJB-Developer fast uneingeschränkt zugreifen.[33]

- **JNDI Namenservice.** Der Namens- und Verzeichnisdienst definiert einen Platz zum Hinterlegen und Wiederfinden von Komponenten. Das Java Naming and Directory Interface bietet zudem die Möglichkeit auf bestimmte Ressourcen (wie zum Beispiel Datenbanken) zuzugreifen.

[32] Vgl. zum folgenden Abschnitt [DenPet02] S.31 ff.

[33] Bei der Implemtierung von Enterprise JavaBeans stehen dem Entwickler nicht alle APIs zur Verfügung. Insbesondere die Java Swing Klassen bzw. GUI-basierte Klassen dürfen nicht benutzt werden, die auch in einer Serverdomäne kaum sinnvoll sind.

- **JDBC Datenbankschnittstelle.** Die Java DataBase Connectivity bietet dem Programmierer ein plattformunabhängiges Standardzugriffsverfahren auf die unterschiedlichsten Datensenken bzw. Datenbanken.

- **JMS Messaging Service.** Die JMS API bietet eine Technologie für die asynchrone Kommunikation zwischen Komponenten mittels „Messages“ in einer verteilten Umgebung. Dafür gibt es eine Reihe von Verteiler-Paradigma wie zum Beispiel Publishing/Subscribing Verfahren oder Point-to-Point, die auch eine parallele Abarbeitung von Anforderungen ermöglichen.

- **JTA TA-Monitor.** Die JTA API implementiert einen Transaktionsmonitor für EJB-Container, welcher dafür sorgt, dass die einzelnen Aktionen einer -wie auch immer gearteten- Transaktion erfolgreich ausgeführt werden. Sollte eine Aktion fehlschlagen, werden auch alle bisher getätigten Aktionen der Transaktion rückgängig gemacht.

Basiskomponente bildet der Server. Da die Spezifikation jedoch keine konkreten Anforderungen an den Server stellt, gibt es auch nicht einen „einzig richtigen Server“. Ein wichtige Anforderung ist jedoch die Bereitstellung einer Laufzeitumgebung für Container. Diese wiederum stellen eine Laufzeitumgebung für Komponenten bereit. Der Server soll die genannten Dienste der J2EE Platform zur Verfügung stellen. Dem Container fällt die Aufgabe zu, die im Standard formulierten Schnittstellen zu implementieren und die verschiedenen Java-Dienste den Komponenten zugänglich zu machen. Darunter fallen die typischen Aufgaben wie z.B. Thread- und Prozessmanagement, Lastverteilung, Sicherheitsmanagement, Namens- und Verzeichnisdienste oder das Pooling von Ressourcen.

Bei diesem Ansatz steht die Integration innerhalb einer Application Server Umgebung klar im Vordergrund. Bezeichnend ist die konzeptionelle Ausrichtung, die EJB Laufzeitumgebung gewissermaßen als Bestandteil eines noch größeren Systems anzusehen.

Die Ähnlichkeit[34] der verwendeten Konzepte zum CCM sind teilweise so frappierend, dass im Folgenden der Schwerpunkt auf den Unterschieden liegen soll. Dazu ist es nicht nötig, die Konzepte von EJB im Detail zu verstehen. Für eine Gegenüberstellung soll im nächsten Abschnitt eine Sammlung von Klassifikationsansätzen er-

[34] Es wurden z.T. sogar gleiche Design-Patterns (Observer) und Metamodelle benutzt. Dies unterstreicht die Ähnlichkeit zwischen EJB und CCM.

arbeitet werden. Damit wird es möglich die wichtigsten Konzepte zu nennen und Unterschiede aufzudecken.

4.2 Klassifikationsansätze

In Tabelle 4.1 werden zum einen Klassifikationsansätze genannt und zum anderen kurz beschrieben. Die Auflistung soll im nächsten Kapitel benutzt werden, um die beiden Ansätze (CCM und Enterprise JavaBeans) gegenüberzustellen.

	Beschreibung
Architekturkonzept	Grundlegendes Architekturparadigma und zentrale Ansätze der jeweiligen Komponententechnologie....
Schnittstellen pro Komponente	Die Anzahl von Schnittstellen, welche den einzelnen Komponenten zur Verfügung stehen sollen. ...
Komponentenarten	Betrachtet wird die Anzahl unterschiedlicher Komponentenarten bzw. Komponentenkonzepte....
Metadatenmanagment für Komponenten	Die Frage danach, wie Metadaten für Verwaltung und Anbindung der Komponenten hinterlegt werden.
Life Cycle Management	Wie und wann werden Komponenten aktiviert, passiviert und ausgelagert? Welche grundsätzlichen Ansätze werden durch die Technologie verfolgt?
Middleware	Die technische Infrastruktur zur Übertragung der Daten, Objekte und Komponenten im LAN/WAN.
Persistenz	Welche Ansätze für die dauerhafte Speicherung von Daten stellt das Framework standardmäßig zur Verfügung?
Namens- und Verzeichnisdienst	Konzept und Umsetzung der Technik, die das Auffinden, Wiederverwenden und Verteilen von Ressourcen ermöglicht.
Integrations-möglichkeiten	Welche Anbindungsmöglichkeiten und Erweiterungen sieht die jeweilige Spezifikation vor? Gibt es Grenzen in der Skalierbarkeit?
Sicherheits-management	Wie sind Sicherheitsfragen bei komponentenbasierten Ansätzen berücksichtigt? Wie wird auf die neuen Herausforderungen bei derart verteilten Systemen reagiert?

Installation	Am Ende eines Entwicklungsprozesses steht die Installation. Wie kann also die Installation bzw. das Deployment von Komponenten durch die Spezifikation unterstützt werden?

Tabelle 4.1: *Klassifikationsansätze*

4.3 Gegenüberstellung beider Technologien

In Tabelle 4.2 werden beide Ansätze nach vorgestellten Schema gegenübergestellt. Dafür wird zunächst das Basiskonzept (sofern vorhanden) der jeweiligen Spezifikation genannt und anschließend kurz charakterisiert.[35]

Architekturkonzept

EJB Spec. V2.0: Container&Interception	Der Zugriff auf Komponenten erfolgt nach dem bereits vorgestellten Interception-Mechanismus. Ein Container erlaubt niemals den direkten Zugriff auf die Komponenten.
CORBA3 (CCM): Container&Interception	Dieser Sachverhalt ist im Wesentlichen identisch. Eine nicht unbedeutende Rolle spielt der POA, welcher Teil des Interception-Mechanismus im CCM ist.

Schnittstellen pro Komponente

EJB Spec. V2.0: Home&Remote Interface	EJB-Komponenten unterstützen standardmäßig nur eine Schnittstelle. Dies ist mit dem Sprachkonzept(„Interface“) von Java begründet.
CORBA3 (CCM): Facets&Receptacles	Eine CORBA Component kann nach Definition mehrere Schnittstellen besitzen.

Komponentenarten

EJB Spec. V2.0: Beans	Session-Bean (nicht persistent), Message-Driven Bean (nicht persistent, aber Nachrichten persistent nach Terminierung), Entity-Bean (persistent auch nach Terminierung)

[35] Vgl. zur folgenden Tabelle [Stahl00], [DenPet02]

CORBA3 (CCM): CORBA Components	Service Component, Session Component, Process Component, Entity Component

Metadatenmanagement

EJB Spec. V2.0: Package Deskriptoren	Beans werden durch XML kodierte Deskriptoren ausgeliefert. Die EJB-Spezifikation gibt klare Richtlinien, wie das Format auszusehen hat. (Appendix B: Deployment descriptor)
CORBA3 (CCM): Package Deskriptoren	Auch im CCM werden die Deskriptoren über ein offenes XML-Format, welches die Spezifikation definiert, beschrieben.

Life Cycle Management

EJB Spec. V2.0: Container	Der Lebenszyklus als auch die Verwaltung der Komponenten bzw. der Adressräume unterliegt den Containern.
CORBA3 (CCM): Portable Object Adapter	Die Aktivierung und Passivierung erledigt sowohl der Container als auch der POA. Der POA merkt sich die Abbildung von CORBA-Objekten auf „Servants“ in einer Objekttabelle.

Middleware

EJB Spec. V2.1: RMI & JMS	Mit Hilfe Remote Method Invocation oder dem Messaging Dienst JMS kommunizieren die Komponenten untereinander. Es spielt sowohl für den Client als auch für die Komponenten keine Rolle, in welchem Adressraum sich die Gegenstelle befindet.
CORBA3 (CCM):CORBA IIOP & CORBA Messaging	Die CCM ist Bestandteil der CORBA Spezifikation. Damit steht dem CCM die volle Funktionalität(auch eine asynchrone Kommunikationsform) des bewährten Objektmodells zur Verfügung.

Namens- und Verzeichnisdienst

EJB Spec. V2.0: Java Naming & Directory Interface	Der EJB Container stellt Beans Informationen über Zugriff und Nutzung des JNDI-Dienstes zur Verfügung.

CORBA3 (CCM): CosNaming	Das Prinzip ist im CCM mittels des CosNaming-Dienstes identisch.
	Persistenz
EJB Spec. V2.0: JDBC und JNDI	Beans haben die Möglichkeit über den Namensdienst (JNDI) auf Datenbankressourcen (JDBC) zu zugreifen. Im Falle der automatischen Persistenz ist es daher für die Komponente unerheblich, wo die Daten abgelegt werden.
CORBA3 (CCM): Persistent State Service	Das abstrakte Verhalten zur Speicherung von Objekt-Zuständen erfolgt durch Storage-Objekte. Für den Benutzer soll nicht sichtbar sein, welche Art von Datenhaltung sich hinter dem Mechanismus verbirgt (information hiding).
	Integrationsmöglichkeiten
EJB Spec. V2.0: RMI-over-IIOP	Die Spezifikation erlaubt mit Hilfe des RMI-over-IIOP Mechanismus explizit die Anbindung an CCM-konforme Komponenten.
CORBA3 (CCM): CORBA Components view for EJBs	Eine spezielle Sicht erlaubt es, EJB Komponenten in das Systemumfeld einzubinden und wie CORBA Components zu behandeln.
	Sicherheitsmanagement
EJB Spec. V2.0: Java Security Package & Java Cryptography Extension	Obwohl Sicherheitsaspekte erst nachträglich in die Spezifikation eingebaut wurden, enthält EJB mit Hilfe der JSP/JCE API eine umfangreiche Auswahl an Maßnahmen.
CORBA3 (CCM): CORBA Security Levels	Die OMG hat mit dieser Technik mehrere Aspekte (Identifikation, Authentifikation, Administration,...) in der Spezifikation berücksichtigt.

Tabelle 4.2: *Gegenüberstellung beider Technologien*

5 Fazit

Im Rahmen dieser Arbeit sollte das CORBA Component Model als ein umfassendes Konzept für die Realisierung von Enterprise-Anwendungen vorgestellt werden. CCM konnte sich als ein universeller und umfassender Ansatz für die moderne komponentenbasierte Softwarentwicklung präsentieren. Es steht ein komplexer und vollständiger Standard mit sehr hohem Potential zur Verfügung. Dabei steht die Stärke der CCM Spezifikation stets im Vordergrund: absolute Sprachneutralität.

Im Vergleich zur EJB Spezifikation werden starke Ähnlichkeiten beider Architekturkonzepte deutlich. Der Erfolg von Enterprise JavaBeans begründet sich durch die breite Akzeptanz der Programmiersprache Java. Der hohe Verbreitungsgrad von J2EE, die bewährten sowie vertrauten Sprachkonstrukte und die gelungene Abbildung eines vollständigen Entwicklungsprozesses erklärt die Beliebtheit von EJB besonders im Bereich von Webservices.

Dabei orientiert sich die EJB Architektur derart stark an CCM und umgekehrt, dass [Stahl00] sogar von „zwei unterschiedlichen Seiten der *selben Medaille*“ spricht. Dies ist auch nicht verwunderlich, weil bei der Konzeption der Entwürfe zum Teil die gleichen Parteien beteiligt waren.

Bleibt festzuhalten, dass man CCM gewissermaßen als Obermenge zu EJB bezeichnen kann. Die verwendeten Konzepte sind letztlich die Gleichen, nur werden diese abstrahiert, sprachneutral umformuliert und neu verpackt. Ebenfalls interessant dürften konkrete Produkte zum CCM sein. Vorschläge zeigen, wie zum Beispiel von [MarvMerl01] mit OpenCCM[36] formuliert und implementiert, dass auch ein sprachneutrales Konzept spätestens im konkreten Einsatz an seine Grenzen stößt und praktische sowie produktivitätsfördernde Ansätze in der industriellen Praxis stets am meisten Anklang finden werden.

[36] Vgl. [MarvMerl01]

Literaturverzeichnis

[1] [Bartlett01] D. Bartlett: */CORBA Component Model (CCM) : Introducing next-generation CORBA/* International Business Machines, Armonk, NY 2001, http://www-106.ibm.com/developerworks/library/co-cjct6/index.html; Abruf 2003-01-14

[2] [Booch87] G. Booch: */Software Engineering with Ada/*, Benjamin-Cummings Publishing Company, San Francisco,CA 1986-1987

[3] [CCM02] Object Management Group: */CORBA Components Full Specification/* Framingham 2002, ftp://ftp.omg.org/pub/docs/formal/02-06-65.pdf; Abruf 2003-01-14

[4] [CORBA01] Object Management Group: */The Common Object Request Broker: Architecture and Specification, Revision 2.6/* Framingham 2001, ftp://ftp.omg.org/pub/docs/formal/01-12-01.pdf; Abruf 2003-01-14

[5] [CORBA02] Object Management Group: */The Common Object Request Broker: Architecture and Specification, Revision 3.0.1/* Framingham 2002, ftp://ftp.omg.org/pub/docs/formal/02-11-01.pdf; Abruf 2003-01-14

[6] [DCOM97] Microsoft Corporation: */The Distributed Object Model (DCOM)/* Redmond,WA 1997, http://www.microsoft.com/com/tech/DCOM.asp; Abruf 2003-01-14

[7] [DenPet02] S. Denninger, I. Peters: */Enterprise JavaBeans 2.0/*, Addison-Wesley Verlag, München 2002

[8] [EJB01] Sun Microsystems: */Enterprise Java Beans 2.0 Public Release/*, Santa Clara 2001, http://www.javasoft.com/ejb/spec.html; Abruf 2003-01-14

[9] [Griffel98] F. Griffel: */Componentware - Konzepte und Techniken eines Softwareparadigmas/*, dpunkt-Verlag 1998

[10] [MarvMerl01] R. Marvie, P. Merle: */CORBA Component Model: Discussion and Use with OpenCCM/* Laboratoire d'Informatique Fondamentale de Lille 2001, http://corbaweb.lifl.fr/OpenCCM/docs; Abruf 2003-01-14

[11] [Merle02] P. Merle: */CORBA Component Model Tutorial/* OMG Meeting Yokohama, 2002, OMG TC Document ccm/2002-04-01, http://www.omg.org; Abruf 2003-01-14

[12] [MOF00] Object Management Group: */OMG Meta Object Facility Specification, Version 1.3/* Framingham 2000, ftp://ftp.omg.org/pub/docs/ad/99-07-03.pdf; Abruf 2002-10-05

[13] [Raj99] G. S. Raj: */The CORBA Component Model (CCM)/* Oak Creek,WI 1999 http://my.execpc.com/ gopalan/corba/ccm.html; Abruf 2003-01-14

[14] [Szyperski98] C. Szyperski: */Component Software/* Addison-Wesley, Longman 1998

[15] [RMI01] Sun Microsystems: */Java Remote Method Invocation (RMI)/*, Santa Clara 2001, http://java.sun.com/products/jdk/rmi/; Abruf 2003-01-14

[16] [Stahl00] M. Stahl: */Im Reich der Mitte - Die Komponententechnologien COM+, EJB und „CORBA Components“/*, In: OBJEKTspektrum, Nr. 3, 2000, S.26 ff., SIGS-Datacom GmbH, Troisdorf 2000

Erklärung über Hilfsmittel

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ilmenau, 1. Februar 2003, Thomas Havemeister