Version: 1.2

Zusammenfassung: mtintel (Martin Tintel)

Quellen: Größtenteils [Wikipedia](#) und paar Sachen [Informatik- Forum](#)

Entity Bean [modellieren](#) die dauerhaften (persistenten) Daten des [Systems](#). Beispiele sind physikalisch vorhandene Dinge wie Benutzer, Informationsstrukturen wie Adressen oder archivierte Vorgangsinformationen wie Rechnungen. Sie repräsentieren z.B. einen Datensatz aus einer Datenbank.

Die [Persistenz](#) kann entweder vom Bean-Entwickler selbst programmiert ("Bean Managed Persistence", *BMP*) oder vom EJB-Container bereitgestellt werden ("Container Managed Persistence", *CMP*). Bei *CMP* wird im *Deployment Descriptor* (siehe unten) unter anderem der Name eines abstrakten Schemas definiert, was üblicherweise dem Namen einer Datenbanktabelle entspricht, in der EJBs einer Klasse abgelegt werden.

WebService: Ab Version 1.4 erlaubt die J2EE-Spezifikation den Aufruf von Stateless Session Beans als [Webservices](#) und beschreibt einen Mechanismus, der die Schnittstelle eines Webservice auf die Schnittstelle einer EJB abbildet.

Session Bean bilden insbesondere Vorgänge ab, die der Nutzer mit dem System durchführt. Sie bedienen sich häufig mehrerer EntityBeans, um die Auswirkungen des Prozesses darzustellen. Man unterscheidet zustandslose ("stateless") und zustandsbehaftete ("stateful") Session-Beans.

Eine **zustandsbehaftete Session Bean** hat ein eigenes Gedächtnis. Sie kann Informationen aus einem Methodenaufruf speichern, damit sie bei einem späteren Aufruf einer anderen (oder der gleichen) Methode wieder zur Verfügung stehen. Die Zustandsbehaftung wird durch die Vergabe einer eindeutigen ID umgesetzt, über diese ID können die stateful Session Beans unterschieden werden.

Im Gegensatz dazu müssen einer **zustandslosen Session Bean** bei jedem Aufruf alle Informationen als Parameter übergeben werden, die für die Abarbeitung dieses Aufrufs benötigt werden. Da eine zustandslose Session Bean keine Informationen speichern kann, ist sie nicht von anderen Session Beans der gleichen Klasse unterscheidbar, sie hat also keine eigene Identität.

Message Driven Bean sind diejenigen Komponenten, die EJB-Systeme für asynchrone Kommunikation zugänglich machen. Hierzu wird der [Java Message Service](#) verwendet. Diese Sorte von Beans wird z. B. häufig für die Kommunikation mit [Legacy-Systemen](#) genutzt.

Servlet: Als Servlets bezeichnet man Java-Klassen, deren Instanzen innerhalb eines Webservers Anfragen von Clients entgegen nehmen und beantworten.

Skeleton: Skeleton (engl. "Skelett", "Gerüst") wird im Bereich Programmierung für eine automatisch generierte Struktur (häufig Quelltext) verwendet, die ein Programmierer oder Benutzer dann ausbauen kann.

Stub: einen lokalen Anknüpfungspunkt für Software. Wort stub für „Stubben“, „Stummel“, „Stumpf“ oder Stutzen bezeichnet in der Softwareentwicklung einen Programmcode, der anstelle eines anderen Programmcodes steht. Dabei ist der andere Programmcode entweder noch nicht entwickelt, zu kompliziert (z.B. während eines Tests) oder er ist auf einem anderen Rechner oder in einem anderen Speicherbereich.

In den letzten beiden Beispielen ist der Stub der lokale Anknüpfungspunkt um ansonsten nur über komplexe Protokolle erreichbare Softwarekomponenten einfach anzusprechen und die Komplexität zu verbergen. Insbesondere kommen Stubs zur Anwendung bei der Entwicklung verteilter Systeme: Funktionalität eines entfernten, nur über ein Netzwerk erreichbaren, Softwaresystems wird auf einem anderen System in Form einer „Stub-Komponente“ zur Verfügung gestellt.

Diese Stubkomponente kann von der Software des anderen Systems angesprochen werden, so als wäre die Funktionalität des entfernten Systems lokal vorhanden. Statt aber diese gewünschte Funktionen tatsächlich zu implementieren, übersetzt die Stub-Komponente die Anfragen in Netzwerkaufrufe, kommuniziert mit dem fernen System und delegiert die Aktion an dieses. Für das nutzende System bleibt dieser Kommunikationsvorgang verborgen, die Stubkomponente verhält sich wie andere lokale Softwarekomponenten auch.

Das **inkrementelle Vorgehensmodell** beschreibt einen [Prozess](#) der [kontinuierlichen](#) Verbesserung, welcher häufig in kleinen oder sogar kleinsten Schritten vollzogen wird. Viele [freie Projekte](#) und [Wikis](#) arbeiten nach diesem Prinzip. Bei der inkrementellen Verbesserung steht der Endzustand in der Regel nicht fest. Das Projekt wächst organisch.

Die **Iteration** (von lateinisch *iterare*, „wiederholen“) ist ein Begriff aus der [numerischen Mathematik](#) und bezeichnet eine Methode, sich der Lösung eines Rechenproblems schrittweise, aber zielgerichtet anzunähern. Sie besteht in der wiederholten Anwendung desselben Rechenverfahrens. Meistens *iteriert* man mit [Rückkopplung](#): Die Ergebnisse eines

[Iterationsschritte](#) werden als Ausgangswerte des jeweils nächsten Schrittes genommen – bis das Ergebnis (beziehungsweise Veränderung einer Bestandsgröße) zufrieden stellt. Ein Beispiel dafür ist das [Newton-Verfahren](#). Manchmal setzt man den nächsten Schritt aus den Ergebnissen der vorherigen zwei Schritte (oder vor noch mehr Schritten) an, zum Beispiel bei der [Regula Falsi](#). Neben der mathematischen iterativen Problemlösung wird in der [Informatik](#) auch von Iteration gesprochen, wenn ein Zugriff iterativ, das heißt schrittweise, beziehungsweise wiederholt, auf Datenstrukturen erfolgt. (z. B.: FOR-Schleife)

der ganze prozess: inkrementell (es kommt immer etwas dazu) (incrementare kommt aus dem Lateinischen und heißt "Der Zuwachs")
die "schleife": iterativ (jedes mal werden die selben schritte durchgeführt) ("iteratio" heißt "Die Wiederholung")

Capability Maturity Model® (kurz CMM®) ist ein [Prozessmodell](#) zur Beurteilung der Qualität ("Reife") des [Softwareprozesses](#) ([Softwareentwicklung](#), Wartung, Konfiguration etc.) von Organisationen sowie zur Bestimmung der Maßnahmen zur Verbesserung desselben.

CMM wurde Ende 2003 durch [Capability Maturity Model Integration®](#) (kurz CMMI®) ersetzt um dem Wildwuchs diverser CM-Modelle (jede Entwicklungs-Disziplin entwickelte ein eigenes Modell) entgegenzuwirken und ein neues, modulares und vor allem vereinheitlichtes Modell zu erstellen.

Neben den Fähigkeitsgraden (*capability levels*) eines einzelnen Prozessgebiets definiert **CMMI Reifegrade** (*maturity levels*). Ein Reifegrad umfasst eine Menge von Prozessgebieten, die zu einem bestimmten Fähigkeitsgrad umgesetzt sein müssen. Jeder Reifegrad ist ein Entwicklungsplateau in der Prozessverbesserung der Organisation. Die Reifegrade sind:

- 1 – Initial: keine Anforderungen, diesen Reifegrad hat jede Organisation automatisch
- 2 – Managed: die Projekte werden gemanagt durchgeführt und ein ähnliches Projekt kann erfolgreich wiederholt werden
- 3 – Defined: die Projekte werden nach einem angepassten Standard-Prozess durchgeführt, und es gibt eine kontinuierliche Prozessverbesserung
- 4 - Quantitatively Managed: es wird eine statistische Prozesskontrolle durchgeführt
- 5 – Optimizing: die Prozesse werden mit den Daten aus der statistischen Prozesskontrolle verbessert

ISO 9000: Eine **Qualitätsmanagementnorm** beschreibt, welchen Anforderungen das Management eines Unternehmens genügen muss, um einem bestimmten Standard bei der Umsetzung des [Qualitätsmanagements](#) zu entsprechen, und kann sowohl informativ für die Umsetzung innerhalb eines Unternehmens oder auch zum Nachweis bestimmter Standards gegenüber Dritten dienen.

In [computer science](#), a [transaction](#) is a group of logical operations that must all succeed or fail as a group. Systems dedicated to supporting such operations are known as **transaction processing** systems.

As an example, consider a typical banking operation, moving \$500 from your savings account to your checking account. This seems like a single operation to the user but in fact consists of two: debiting the savings account by \$500 and crediting the checking account by \$500. If the debit operation succeeded and the credit did not, the \$500 would disappear. This might still sound simplistic and easy to avoid, but consider that each of these two main operations may consist of several sub-operations, each of which may or may not fail; therefore we'll use these two "main" operations (debiting and crediting) as a generic alias for any number of elementary operations in a generic transaction.

Transaction processing systems allow these two operations to be "grouped" into a single transaction so these sorts of consistency problems cannot occur. They do this by making copies of the data in question and then running the operations on the copied data. When both commands have successfully completed, the changed data are written back to the system in a single operation. If either operation failed, the copied data are simply discarded, and an error is reported. This desirable property is called [atomicity](#) and is one of the desirable [ACID](#) transaction properties.

Standard transaction processing [middleware](#), notably [IBM's Information Management System](#), emerged in the 1960s and was often closely coupled to particular [database management systems](#). Client-server computing embraced similar principles in the 1980s with mixed success. But, in more recent years, the distributed client-server model has become considerably more difficult to maintain. As the number of transactions grew in response to various online services (especially the [Web](#)), a single distributed database was not a practical solution. In addition, most online systems consist of a whole suite of programs operating together, as opposed to a strict client-server model where the single server could handle the transaction processing. Today a number of transaction processing systems are available that work at the inter-program level and which scale to large systems, including [mainframes](#).

Remote Procedure Call, oder kurz **RPC**, ist ein [Netzwerkprotokoll](#) auf der fünften, teilweise auch sechsten Schicht des [ISO/OSI-Modells](#). Mit Hilfe von RPC können über ein Netzwerk Funktionsaufrufe auf entfernten Rechnern durchgeführt werden.

RPC wurde ursprünglich durch [Sun Microsystems](#) für [NFS](#) entwickelt. Der genaue Aufbau von RPC wird in den [RFCs](#) 1057 und 1831 beschrieben. Die Idee hinter RPC beruht auf dem [Client-Server-Modell](#), es sollte die gemeinsame Nutzung von Programmfunktionen über Rechengrenzen ermöglichen. Ein RPC-Aufruf läuft fast immer [synchron](#) ab, das heißt, dass der aufrufende [Client](#) mit der Ausführung des weiteren Programmcodes wartet, bis er eine Antwort der Prozedur vom [Server](#) erhalten hat.

Es lassen sich weiterhin drei inkompatible Versionen von RPC unterscheiden, das [ONC RPC](#), das vielfach auch als [Sun RPC](#) bezeichnet wird, ist die am weitesten verbreitete RPC Variante. [ONC RPC](#) steht hierbei für [Open Network Computing](#) Remote Procedure Call, für diese RPC Variante findet sich unter anderem auch eine Implementierung in [Linux](#).

Die zweite nicht ganz so weit verbreitete RPC Variante ist das [Distributed Computing Environment](#) Remote Procedure Call oder kurz DCE RPC. Die letzte Variante schließlich ist ein gescheiterter Versuch der [ISO](#) ein standardisiertes ISO RPC zu etablieren, von ISO RPC gibt es bis heute jedoch kaum eine Implementierung.

Die wichtigste Komponente auf der Serverseite ist der [Portmapper-Daemon](#), der auf dem [UDP](#)- und [TCP-Port](#) 111 lauscht. Der Portmapper übernimmt die Koordination der durch den Client gewünschten Funktionsaufrufe. Jedes [Programm](#), das auf dem Server RPC-Dienste zur Verfügung stellen will, muss daher dem Portmapper bekannt sein.

Neben [NFS](#) (Network File System) basiert unter anderem noch [NIS](#) (Network Information Service) in weiten Teilen auf RPC-Aufrufen.

Ein weiterer RPC-Ableger ist das so genannte [XML-RPC](#), welches die versendeten Daten in ein [XML](#)-Dokument kapselt und über eine [HTTP](#)-Verbindung überträgt.

Nachrichtenorientierte Middleware bzw. **Message Oriented Middleware (MOM)** bezeichnet [Middleware](#), die auf der asynchronen Kommunikation, also der Übertragung von Nachrichten ([Messages](#)) beruht. Das Format für die Nachrichten ist nicht festgelegt, in der Praxis hat sich jedoch [XML](#) als beliebtes Format etabliert.

MOM unterstützt drei verschiedene Kommunikationsprotokolle

[Message Passing](#) (Direkte Kommunikation zwischen Applikationen)

[Message Queueing](#) (Indirekte Kommunikation über eine Nachrichtenqueue)

[Publish & Subscribe](#) (Erzeuger (Publisher) stellt Konsument (Subscriber) Messages zur Verfügung)

Ein sogenannter **Object Request Broker** ermöglicht die Kommunikation von Objekten innerhalb eines verteilten Systems, wie z.B. das [Internet](#), unabhängig vom jeweiligen Betriebssystem des Zielrechners und der verwendeten Programmiersprache. Die Aufgaben des ORB bestehen unter anderem darin, Daten zu senden bzw. zu empfangen. Über eine Schnittstelle ist es Servern und Clients überdies möglich, wechselseitig Informationen abzurufen und Aktionen auf den jeweiligen Rechnern zu starten. Diese Schnittstelle wird beispielsweise von europäischen [Polizeibehörden](#) für den Datenabgleich benutzt.

Die Object Management Group (OMG) versucht unter dem Namen [CORBA](#) eine Standardisierung des ORB zu erreichen.

Middleware bezeichnet in der [Informatik](#) anwendungsunabhängige Technologien, die Dienstleistungen zur Vermittlung zwischen Anwendungen anbieten, so dass die Komplexität der zugrundeliegenden Applikationen und Infrastruktur verborgen wird (nach W. Ruh u.a.:

[Enterprise Application Integration](#), Wiley, 2001). Man kann Middleware auch als eine Verteilungsplattform, d.h. als ein [Protokoll](#) (oder Protokollbündel) auf einer höheren [Schicht](#) als der der gewöhnlichen Rechnerkommunikation auffassen.