

Anhang zum Buch

**„Rekonfiguration komponentenbasierter
Softwaresysteme zur Laufzeit“**

ISBN 978-3-8348-1001-4

Author: Jasminka Matevska

Inhaltsverzeichnis

Anhang	1
A Enterprise JavaBeans Komponentenmodell	3
A.1 EJB Komponentenstruktur	3
A.2 Das EJB Transaktionsmodell	11
B Java EE Deployment API	13
B.1 Java EE Produktanbieter	14
B.2 Deployment-Tool Anbieter	16
B.3 Gemeinsame Klassen	17
C Testergebnisse der Auswertung typischer Rekonfigurationsszenarien	19
C.1 JBoss 4.0.x unter Linux	20
C.2 WebLogic unter Linux	26
D Experimentergebnisse der Bestimmung und Erkennung minimaler Laufzeitabhängigkeiten	27
D.1 Ergebnisse der Bestimmung von Laufzeitabhängigkeiten	27
D.2 Ergebnisse der Wiedererkennung von Laufzeitabhängigkeiten	31
Literaturverzeichnis	35

Abbildungsverzeichnis

A.1	Lebenszyklus einer Entity Bean [LR05]	4
A.2	Lebenszyklus einer Stateless Session Bean [LR05]	6
A.3	Lebenszyklus einer Stateful Session Bean [LR05]	6
A.4	Lebenszyklus einer Message Driven Bean [LR05]	7
A.5	Ablauf eines Zugriffs auf eine Entity oder Session Bean [RAJ04] .	10
B.1	Die drei Hauptrollen der Java EE Deployment Spezifikation [Pak03]	14
D.1	Versuch 1 (1 Benutzer) [Stö07]	28
D.2	Versuch 2 (10 Benutzer) [Stö07]	29
D.3	Versuch 3 (20 Benutzer) [Stö07]	29
D.4	Versuch 1 (1 Benutzer) [Stö07]	29
D.5	Versuch 2 (10 Benutzer) [Stö07]	30
D.6	Versuch 3 (20 Benutzer) [Stö07]	30
D.7	Versuch 1 (1 Benutzer) [Grü08]	32
D.8	Versuch 2 (10 Benutzer) [Grü08]	33
D.9	Versuch 3 (20 Benutzer) [Grü08]	33

Tabellenverzeichnis

C.1	Ergebnisse der Entity Bean Container-Managed Relations Testsuite	20
C.2	Ergebnisse der Entity Bean Testsuite	21
C.3	Ergebnisse der Message Driven Bean Testsuite (Zugriff auf Entity Beans)	22
C.4	Ergebnisse der Message Driven Bean Testsuite (Bibliotheken) . .	22
C.5	Ergebnisse der Stateful Session Bean Testsuite	23
C.6	Ergebnisse der Stateful Session Bean Testsuite mit Transaktionen .	24
C.7	Ergebnisse der Stateless Session Bean Testsuite	25
C.8	Testergebnisse der Stateless Session Bean-Testsuite	26
D.1	Ergebnisse für die Komponente <code>CatalogService</code> [Stö07] . . .	27
D.2	Ergebnisse für die Komponente <code>ItemSqlMapDao</code> [Stö07] . . .	28
D.3	Ergebnisse für die Komponente <code>OrderSqlMapDao</code> [Stö07] . . .	28
D.4	Ergebnisse für die Komponente <code>CatalogService</code> [Grü08] . .	31
D.5	Ergebnisse für die Komponente <code>ItemSqlMapDao</code> [Grü08] . . .	31
D.6	Ergebnisse für die Komponente <code>OrderSqlMapDao</code> [Grü08] . .	32

Anhang

A Enterprise JavaBeans

Komponentenmodell

“The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification“ [Sun06b].

Das **Enterprise JavaBeans Komponentenmodell** realisiert die **Geschäftsschicht** (*Business-Tier*) der mittleren (**Anwendungsschicht**). Enterprise JavaBeans (EJB) sind serverseitige Komponenten, die die Geschäftslogik einer Java EE Anwendung realisieren und in einer transaktionsgesteuerten Laufzeitumgebung ausgeführt werden können. Sie werden auf den EJB-Container deployed und dort können sie ausgeführt werden. Der Aufruf kann lokal oder auch über CORBA bzw. Web Services erfolgen. Sie können über das SOAP/HTTP-Protokoll auch direkt Web-Services anbieten. Das EJB-Komponentenmodell basiert auf der Programmiersprache Java und ist dadurch plattformunabhängig. Die EJB-Architektur basiert auf verteilten Objekten, dem serverseitigen Komponentenmodell und dem Component Transaction Monitor (CTM).

A.1 EJB Komponentenstruktur

Es gibt drei Typen von EJBs: **Entity Beans**, **Session Beans** und **Message Driven Beans**. Der Name, der Typ und die Sichtbarkeit einer Bean werden vorab in den Deskriptoren der Anwendung festgelegt und können sich nicht zur Laufzeit ändern. Jeder Beantyp hat einen fest definierten Lebenszyklus der mögliche Zustände und Zustandsübergänge durch den Aufruf von Methoden bei der Erstellung, Sicherung oder Zerstörung der Beans enthält.

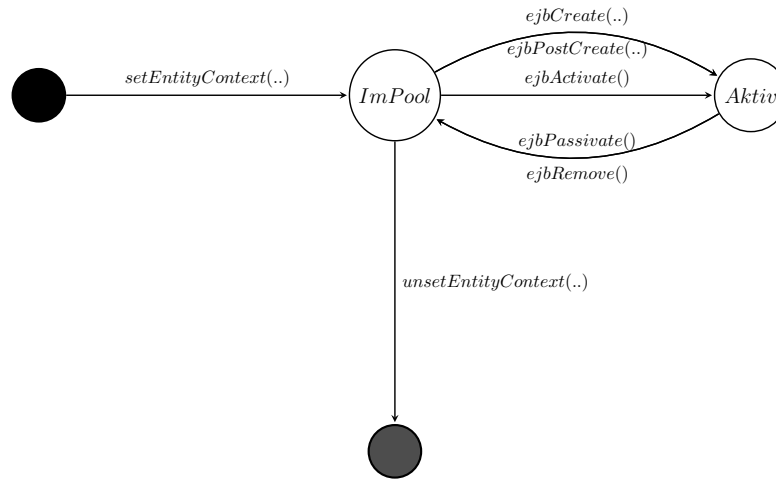


Abbildung A.1: Lebenszyklus einer Entity Bean [LR05]

A.1.1 Entity Beans

Entity Beans modellieren Objekte aus der realen Welt, die mit bestimmten Daten assoziiert werden (z.B. Konto, Kunde, Adresse). Sie repräsentieren persistente Daten, die in der Regel in einer Datenbank gespeichert und verwaltet werden. Jeder Datensatz aus der Datenbank kann durch eine bestimmte Entity Bean abgebildet werden. Die Datensätze sind über einen Primärschlüssel eindeutig identifizierbar. Damit ist es möglich auf Daten in der Datenbank zuzugreifen, bzw. die veränderten Daten abzulegen, ohne einen direkten Datenbankzugriff, z.B. mit der Abfragesprache *SQL*, durchführen zu müssen. Die Schnittstelle zur Kommunikation zwischen einer Entity Bean und dem Container ist der sogenannte Kontext (`javax.ejb.EJBContext`-Klasse). Bei den Entity Beans wird zwischen zwei Typen von Persistenz unterschieden: **Bean-Managed persistence (BMP)** und **Container-Managed Persistence (CMP)**, was nur für den Entwickler, der die Beans implementiert, relevant ist. Bei der Bean-Managed Persistenz muss der Bean-Entwickler selbst die Verwaltung der Datenzugriffe implementieren. Dagegen wird bei der Container-Managed Persistenz diese Aufgabe vom Container erledigt. Er übernimmt die Sicherung der Persistenz der Daten und kann, wenn im Deployment-Deskriptor gefordert, auch das Erstellen der Datenbanktabellen während des Deployments übernehmen.

Der Lebenszyklus einer Entity Bean ist in der Abbildung A.1 dargestellt. Nach dem Starten des EJB-Servers werden mehrere Instanzen der Beans erzeugt und in einen sog. *Beans Pool* abgelegt. Sie sind allerdings passiv und nicht mit einem EJB-Objekt verbunden und repräsentieren somit noch keine Daten aus der Datenbank. Erst mittels Suchmethoden, welche das Interface der Bean bereit stellt oder bei einer Neuerstellung (`ejbCreate`, `ejbPostCreate`, `ejbActivate`), wird eine Bean in den aktiven Zustand überführt und kann verwendet werden. Ein Entfernen oder das kurzfristige Passivieren (`ejbPassivate`) schreibt die Daten der Bean zurück in die Datenbank und gibt den Arbeitsspeicher frei. Beim Erstellen einer neuen EntityBean wird die Methode `ejbCreate` aufgerufen und die Daten in der Datenbank abgelegt. Im Anschluss daran wird durch den Anwendungsserver die `ejbPostCreate`-Methode aufgerufen. Ein Primärschlüssel steht erst nach dem Anlegen der Datenbankzeile zur Verfügung.

A.1.2 Session Beans

Session Beans modellieren Interaktionen und verwalten Prozesse und Aktivitäten des Anwendungssystems. Die Session Beans repräsentieren keine Datenbankinhalte und haben keinen Primärschlüssel. Sie können Entity Beans verwenden, um die Funktionalität der Applikationen bzw. die Geschäftslogik der Anwendung zu realisieren. Session Beans sind den aus der *Java-SE* bekannten Klassen und Objekten am ähnlichsten. Es gibt zwei Arten von Session Beans *zustandslose* (*stateless*) und *zustandsbehaftet* (*stateful*). Im Gegensatz zur normalen Objekterstellung (*new*-Operator), werden diese Beans vom Container erstellt und verwaltet. Bei zustandslosen Session Beans wird kein Konversationszustand für die aufgerufenen Methoden verwaltet. Deshalb können sie zwischen den EJB-Objekten ausgetauscht werden. Das bedeutet, wenn die Kommunikation zwischen dem Client und der Bean beendet ist, kann diese Bean einem anderen EJB-Objekt zugewiesen werden. Lesende Aktivitäten, wie z.B. die Kontostandabfragen, könnten gut mit zustandslosen Session Beans realisiert werden. Zustandslose Session Beans eignen sich nicht zum Speichern von Informationen. Weil sie keinen Zustand besitzen, müssen die zu verarbeitenden Daten bei jedem Methodenaufruf als Parameter übergeben oder aus einer Entity Bean bezogen werden. Zur Realisierung von schreibenden Aktionen eignen sich die zustandsbehafteten Session Beans. Eine zustandsbehaftete Session Bean repräsentiert einen Client innerhalb des Anwendungsservers und ist vom Verhalten mit Objekten der objektorientierten Programmierung vergleichbar. So kann z.B. für jeden angemeldeten Benutzer im System eine Session Bean existieren, die die entsprechenden Informationen in ihrem Konversationszustand verwaltet. Wenn eine zustandsbehaftete Bean einmal

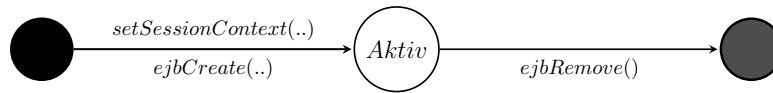


Abbildung A.2: Lebenszyklus einer Stateless Session Bean [LR05]

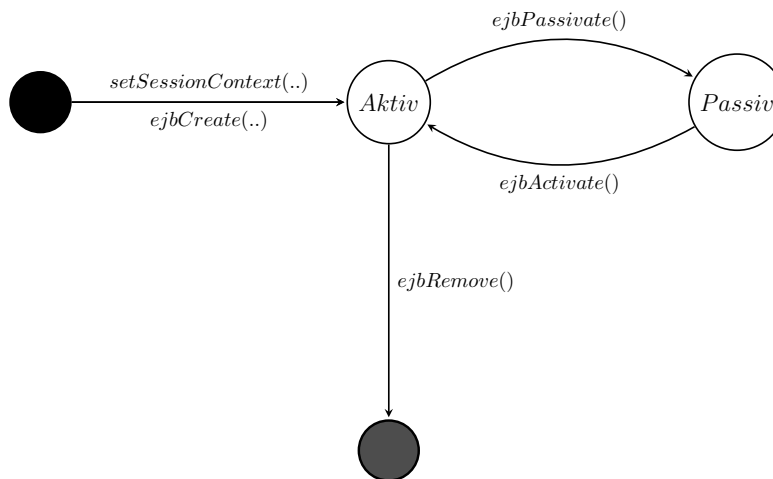


Abbildung A.3: Lebenszyklus einer Stateful Session Bean [LR05]

erzeugt und einem EJB-Objekt zugewiesen wird, bleibt sie für die gesamte Lebensdauer als Referenz zu diesem Objekt. Sie kann nicht zwischen EJB-Objekten ausgetauscht werden. Beim Beenden der Kommunikation wird die Bean automatisch vom Container zerstört.

Die Session Beans haben auch einen Lebenszyklus den der EJB-Container überwacht und steuert. Bei jeder Änderung des Systemzustandes werden entsprechende Methoden der Bean aufgerufen, um diese zu benachrichtigen. Der Lebenszyklus einer zustandslosen Session Bean ist sehr einfach (siehe Abbildung A.2). Die Methode `setSessionContext` setzt den Kontext, in der die Bean sichtbar ist. Wenn eine Bean erstellt wird, wird in der Regel (durch das Caching des Anwendungsservers nicht unbedingt bei jeder Bean-Instanzierung verwendet) Methode `ejbCreate` aufgerufen. Die Methode `ejbRemove` wird dagegen aufgerufen, wenn der EJB-Container die Bean wieder entfernen möchte. Die Methoden der Bean

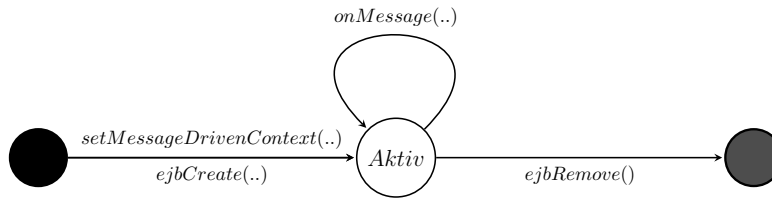


Abbildung A.4: Lebenszyklus einer Message Driven Bean [LR05]

sind im Zustand *Aktiv* verfügbar. Bedingt durch die Eigenschaften einer zustandsbehafteten Session Bean, ist ihr Lebenszyklus komplexer (siehe Abbildung A.3). Bei einer längeren Nichtverwendung einer zustandsbehafteten Session Bean, kann der EJB-Container serialisieren und auf der Festplatte speichern, um Ressourcen des Servers frei zu geben. Dieser Zustand ist der sog. *Passiv-Zustand*, in dem die Bean zwar existiert, aber nicht mehr im Arbeitsspeicher vorliegt. Bei einer erneuten Verwendung der Bean ist von dem Zustandswechsel, abgesehen von der längeren Zugriffszeit, kein Unterschied feststellbar. Der Entwickler kann durch die Methoden `ejbPassivate` und `ejbActivate` auf ein solches Rücksichern der Bean auf Festplatte reagieren und nach Bedarf beispielsweise die Daten zusätzlich in die Datenbank sichern.

A.1.3 Message Driven Beans

Message-Driven Beans werden für den asynchronen Meldungs-austausch zwischen Komponenten verwendet. Im Vergleich zu den Entity Beans und Session Beans, haben die Message-Driven Beans keine Interface-Methoden und bestehen nur aus einer Bean Klasse. Der Lebenszyklus (siehe Abbildung A.4) einer Message Driven Bean ist dem Lebenszyklus einer stateless Session Bean ähnlich. Nach dem Erstellen der Bean kann diese eingehende Nachrichten mit der `onMessage` Methode verarbeiten. Wird die Bean nicht mehr benötigt, wird diese nach Aufruf der `ejbRemove` Methode aus dem Arbeitsspeicher entfernt.

A.1.4 Bestandteile einer EJB-Komponente

Bei der Implementierung einer EJB müssen folgende Schritte durchgeführt werden:

1. Definition des *EJB-Remote-Interface*: Das *Remote-Interface* muss von `javax.ejb.EJBObject` abgeleitet werden, was wiederum von `java.rmi.Remote` abgeleitet ist. Das *Remote-Interface* stellt die öffentlichen Methoden bereit, die ein Client remote aufrufen kann. Diese Schnittstelle wird vom Container implementiert. Dabei muss, im Gegensatz zum lokalen Interface, jede Methode eine `java.rmi.RemoteException` auslösen können. Die Instanzen dieser Klasse sind EJB-Objekte.
2. Definition des *Home-Interface*: Das *Home-Interface* muss von `javax.ejb.EJBHome` abgeleitet werden und stellt Lebenszyklusmethoden zum Erzeugen, Suchen und Zerstören von Beans bereit. Sie können danach, mittels der im *Remote-Interface* spezifizierten Methoden, verwendet werden. Der Container implementiert dieses Interface. Die Instanzen dieser Klasse sind lokale EJB-Objekte.
3. Definition des *LocalHome-Interface*: Das *LocalHome-Interface* wird von `javax.ejb.EJBLocalHome` abgeleitet und spezifiziert Methoden, die lokal von der Bean aufgerufen werden können.
4. Definition des *Local-Interface*: Das *Local-Interface* wird ebenfalls von `javax.ejb.EJBLocalHome` abgeleitet und stellt die Lebenszyklusmethoden, wie die `create`-Methode bereit, die die Bean als *LocalHome-Interface* zurück gibt. Danach können die in *LocalHome-Interface* zur Verfügung gestellten Methoden verwendet werden.
5. Definition der Primärschlüssel Klasse bei einer Entity Bean: Durch einen Primärschlüssel als Objekt wird eine Entity Bean eindeutig anhand des Entity Typs und des Home-Interfaces identifiziert. Es ist möglich, die Primärschlüssel als serialisierbare Objekte zu definieren und somit kann eine bestimmte Bean in der Datenbank wiedergefunden werden.
6. Implementierung der EJB-Klasse: Die Bean Klasse realisiert den Zustand und das Verhalten der Bean. Hier werden Methoden aus dem Remote- bzw. Home-Interface und benötigte zusätzliche Methoden implementiert.
7. Erzeugen der Deployment-Deskriptoren (`ejb-jar.xml`). Deployment-Deskriptoren sind XML-Dateien, die dem EJB Server mitteilen, welche Eigenschaften die Beans haben. Sie beschreiben ihre Struktur und Beziehungen

zu ihrer Umgebung. So wird der Server informiert, welche Transaktionen zur Laufzeit auf jede einzelne Bean Klasse anzuwenden sind. Zu jeder Bean gehört ein entsprechender *Deployment-Deskriptor*, in dem die Sichtbarkeit und der Typ der Bean festgelegt werden. Zusätzlich kann der Deployment-Deskriptor weitere Metadaten enthalten, die der EJB-Container zur Laufzeit umsetzen muss. Abhängig von dem Typ und der Sichtbarkeit, müssen unterschiedliche Interfaces und Klassen mitgeliefert werden, um den Zugriff von anderen Beans zu ermöglichen. Die Deskriptoren werden zusammen mit den benötigten Bean Komponenten in einer *jar-Datei* archiviert. Die EJB-Spezifikation erlaubt es, mehrere EJBs in einer *EJB-Jar Datei* zu archivieren. Der Container behandelt alle zusammen als eine einzige JavaEE-Anwendungskomponente.

Eine detaillierte Beschreibung zu den Deployment-Deskriptoren und Interfaces befindet sich in [LR05] und [BG02].

A.1.5 Zugriff auf eine EJB 2.1

Obwohl eine Enterprise Java Bean aus mehreren Teilen besteht, sind davon nur die Interfaces und die Primärschlüssel-Klasse für einen Client sichtbar. Ein Client kann nur über die vom Anwendungsserver angebotene Infrastruktur auf eine Enterprise Java Beans zugreifen. Ein direkter Zugriff ist aufgrund der Konzeption in Schichtenarchitektur nicht möglich. Der Anwendungsserver regelt vor und nach dem Aufruf einer Bean, z.B. die Transaktionssteuerung oder führt Sicherheitsabfragen durch. Der Ablauf der Kommunikation beim Zugriff ist bei Entity und Session Beans der selbe (siehe Abbildung A.5). Als erstes muss der Client eine Referenz auf ein *EJB-Objekt* erhalten. Hierzu erfragt er eine Referenz auf ein *EJBHome-Objekt* im Namen und Verzeichnisdienst von Java (JNDI) [Sun09]. Mit der Referenz auf das *EJBHome-Objekt* kann der Client eine Referenz auf das *EJB-Objekt* erzeugen oder *Home-Methoden* ausführen. Dabei hängt der Ablauf des Zugriffs davon ab, ob der Client lokal bzw. auf der selben Virtuellen Maschine läuft und einen direkten Zugriff auf Objekte hat oder nicht. Falls der Client lokal ist, kann er über die lokalen Schnittstellen auf die Beans zugreifen. Falls der Client keinen direkten Zugriff auf die Objekte hat, muss er über *Remote-Procedure-Calls* basierend auf der Java Technik *Remote Method Invocation (RMI)*[Sun06a] auf die *Remote-Interfaces* der Beans zugreifen.

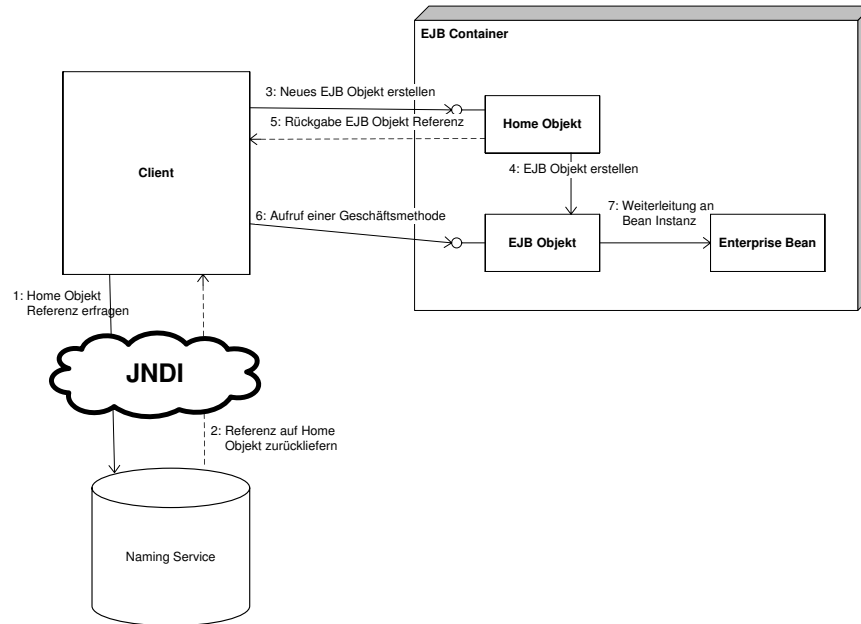


Abbildung A.5: Ablauf eines Zugriffs auf eine Entity oder Session Bean [RAJ04]

A.1.6 Änderungen in der EJB 3.0

Aufgrund der Notwendigkeit, (1) diverse Schnittstellen zu einer Bean zu erstellen, (2) sämtliche Eigenschaften in einem Deployment-Descriptor festzulegen und (3) Methoden zur Zustandswechselverwaltung, wie `ejbActivate` und `ejbPassivate` auch bei einer Nichtbeachtung zu implementieren, war die Entwicklung mit Enterprise Java Beans der Version 2.1 verhältnismäßig aufwendig. Eine Erleichterung bei der Entwicklung brachten Tools, wie z.B. *XDoclet* [XT09]. *XDoclet* nutzt die Beschreibungen in den JavaDoc-Kommentaren der Bean und der Methoden, um die entsprechenden Interfaces und die Deployment-Deskriptoren zu erzeugen.

Mit der *Java Standard Edition* in der Version 5.0 [Sun04] wurden sog. *Annotationen* eingeführt. Annotationen sind Anmerkungen im Quellcode, die mit einem `@` beginnen und direkt über dem Kopf der Klasse bzw. Methode geschrieben werden. Die Umgebung wertet diese Metadaten zur Laufzeit aus und setzt die angegebenen Spezifikationen zur Implementierung um. Damit kann der Entwickler auf die

Erstellung und Pflege von Interfaces verzichten. Sie werden für alle benötigten Methoden vom Entwickler als Annotationen angegeben und werden zur Laufzeit automatisch angelegt. Welche Annotationen für welche Methoden angegeben werden müssen, ist genau festgelegt (siehe z.B. [LR05]). Für den Zugriff auf eine EJB kann jedoch nicht auf die Interfaces verzichtet werden. Der erfolgt genauso wie bei EJB 2.1 (siehe Abbildung A.5 im Abschnitt A.1.5).

Diese Erleichterung stellt allerdings eine große Änderung dar und führt zu Kompatibilitätsproblemen. Eine Vermischung der beiden Versionen in einer Anwendung ist gar nicht möglich, weil der Anwendungsserver vor der Installation der Anwendung anhand der Deployment-Deskriptoren feststellen muss, um welche Version der Beans es sich handelt, um die Installation mit der entsprechenden Implementierung ausführen zu können.

A.2 Das EJB Transaktionsmodell

Ein wichtiges Merkmal der Enterprise JavaBeans Architektur ist die Unterstützung verteilter Transaktionen. Es soll möglich sein, eine Anwendung zu erstellen, die automatisch Daten in verteilten Datenbanken aktualisieren kann. Dabei können unterschiedliche EJB-Server benutzt werden. Für die Transaktionsverwaltung in Java EE ist die Java Transaction API [CM02] verfügbar. Das EJB-Modell beinhaltet ein flaches Transaktionsmodell und unterstützt somit keine geschachtelten Transaktionen. Dabei hat der Bean Entwickler die Möglichkeit die Transaktionsverwaltung vom Container erledigen zu lassen oder eine Implementierung derer selbst vorzunehmen (ähnlich wie bei der CMP bzw. BMP).

Der Container, der EJBs mit *container-managed-transactions* enthält, verwaltet die Transaktionsgrenzen für jede Methode der EJB-Komponenten separat. Es gibt dabei folgende sechs unterschiedlichen Strategien für die der Container konfiguriert werden kann:

- **Not supported:** Wenn die Methode aufgerufen wird, darf keine Transaktion aktiv sein. Der Container wird vor jedem Aufruf der Methode alle aktiven Transaktionen suspendieren.
- **Supports:** Für die Methode ist es nicht relevant ob eine Transaktion aktiv ist. Falls eine Transaktion aktiv ist, wird diese Methode in den aktiven Transaktionskontext ausgeführt.
- **Required:** Die Methode verlangt eine Transaktion. Der Container wird eine neue Transaktion starten, falls keine aktiv ist und die Methode aufrufen. Die Transaktion wird bestätigt sobald die Methode beendet ist (*commit*).

- **Requires new:** Die Methode verlangt einen neuen Transaktionskontext. Der Container wird vor jedem Aufruf der Methode alle aktiven Transaktionen suspendieren und einen neuen Transaktionskontext starten. Sobald die Methode beendet ist, wird dieser bestätigt (*commit*).
- **Mandatory:** Die Methode braucht einen aktiven Transaktionskontext, der mit dem Aufruf zusammen kommt. Falls zum Zeitpunkt des Aufrufs kein Kontext aktiv ist, scheitert der Container mit einem Ausnahmefehler.
- **Never:** Die Methode schlägt fehl, falls zum Zeitpunkt des Aufrufs ein Transaktionskontext aktiv ist.

B Java EE Deployment API

Die Java EE Spezifikation definiert nicht den Deployment-Prozess. Das Deployment der Anwendungen auf einen Server wird herstellerspezifisch und folglich unterschiedlich für unterschiedliche Anwendungsserver definiert und durchgeführt. Als Definition des Deployment-Prozesses bietet Sun Microsystems die J2EE Deployment API Spezifikation (JSR-88) [Sea03] an. Diese Spezifikation setzt eine rollenbasierte Entwicklung als Basis zur Definition des Deployment-Prozesses und legt drei Hauptrollen fest (siehe Abbildung B.1):

1. J2EE (Java EE) Produkthanbieter
2. Deployment-Tool Anbieter
3. Deployer bzw. Benutzer

Das Deployment von Komponenten wird über Komponenten-Meta-Daten koordiniert und konfiguriert. Diese Meta-Daten beschreiben sowohl die Schnittstellen der Komponenten als auch eine genau definierte Menge an nicht-funktionalen Eigenschaften, wie Transaktionsgrenzen, Abhängigkeiten zwischen Komponenten und Ressourcen, Persistenzverwaltung, physikalische Verteilung, aber auch Interoperabilitäts- und Sicherheitsaspekte. Die Deployment API definiert im Grunde ein Protokoll für den Austausch von diesen Meta-Daten zwischen dem Java EE Produkthanbieter und dem Deployment-Tool Anbieter. Sie beinhaltet die Definition der rollenspezifischen Schnittstellen und eine dritte Menge an Klassen (*utility classes*), die seitens der Spezifikation (in diesem Fall von Sun Microsystems) implementiert werden und zur Verbindung der beiden anderen Teile dient. Dazu wird folgende Paketstruktur vorgegeben:

- `javax.enterprise.deploy.spi` beinhaltet die Interfaces, die vom Java EE Produkthanbieter implementiert werden sollen.
- `javax.enterprise.deploy.model` beinhaltet die Interfaces, die vom Deployment-Tool Anbieter zur Verfügung gestellt werden.
- `javax.enterprise.deploy.shared` beinhaltet die gemeinsamen Klassen, die von der spezifizierenden Seite (Sun Microsystems) implementiert werden.

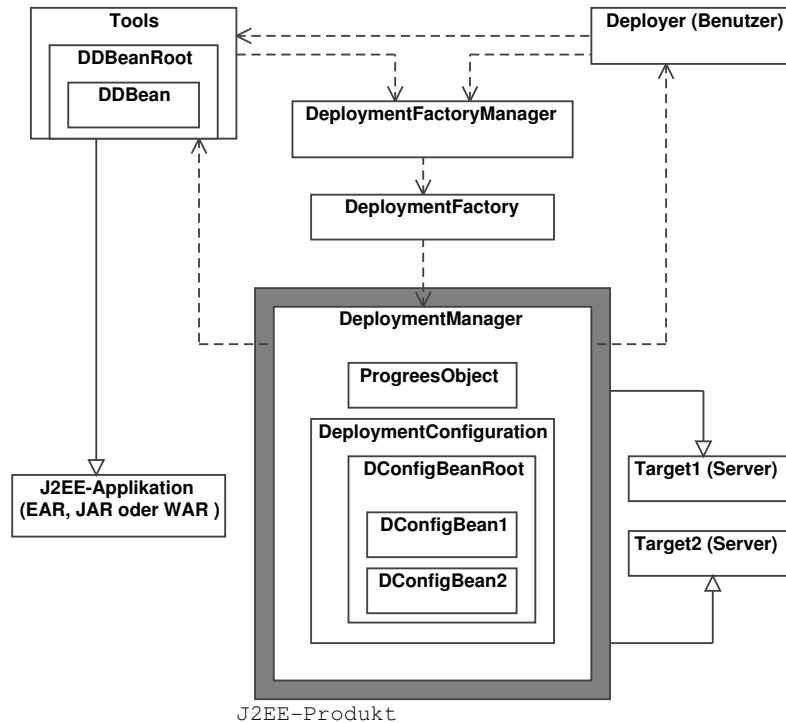


Abbildung B.1: Die drei Hauptrollen der Java EE Deployment Spezifikation [Pak03]

Der Deployer hat die Aufgabe, die Java EE Anwendungen auf ein spezielles Java EE Produkt mit Hilfe des Deployment-Tools zu konfigurieren, anzupassen, die Anwendung auf den Server zu verteilen und deren Ausführung zu starten.

B.1 Java EE Produktanbieter

Java EE Produktanbieter sind Hersteller von Java-EE-spezifischen Produkten wie Anwendungsserver, Web-Server oder Datenbanken. Damit ein Deployment der Java EE Anwendungen auf diese Produkte durchgeführt werden kann, sollten diese Produkte mit den Deployment-Tools der Toolanbieter kommunizieren können. Die

Java EE Produktanbieter sind verantwortlich für die Implementierung der Interfaces im `javax.enterprise.deploy.spi` Paket.

B.1.1 Der Deployment Manager

Der Java EE Produktanbieter muss mindestens einen *DeploymentManager* mit Funktionalität zum Starten, Stoppen, Verteilen und Entfernen von Java EE Modulen zur Verfügung stellen. Der *DeploymentManager* wird über einen sog. *uniform resource identifier (URI)*, einer Instanz des Java EE Produkts, die er managen soll, eindeutig zugeordnet. Instanzen eines *DeploymentManagers* werden über eine *DeploymentFactory* erzeugt. Das Deployment-Tool erzeugt wiederum Instanzen von der *DeploymentFactory* und registriert diese mit dem *DeploymentFactoryManager*. Dabei gibt es den folgenden Ablauf bei der Zuordnung eines *DeploymentManagers*: Der Deployer, der das Deployment-Tool benutzt, stellt eine URI und Zugangsdaten zum Server, der administriert werden soll. Das Deployment-Tool versucht dann einen *DeploymentManager* durch den *DeploymentFactoryManager* zu finden. Der *DeploymentFactoryManager* durchsucht die Liste aller registrierten Instanzen der *DeploymentFactory*, um eine zu finden, die die vorgegebene URI managen kann. Falls gefunden, erzeugt diese eine Instanz des entsprechenden *DeploymentManagers*. Der *DeploymentManager* bietet unter anderem folgende Interfaces an:

- `createConfiguration` gibt ein Objekt zurück, das die serverspezifische Laufzeitkonfiguration der J2EE-Anwendung Module auswertet und generiert.
- `getAvailableModuls` gibt eine Liste aller J2EE-Module zurück, die auf dem abgefragten Server vorhanden sind.
- `getRunningModuls` gibt eine Liste aller J2EE-Module (TargetModulIDs) zurück, die auf dem abgefragten Server laufen.
- `getNonRunningModuls` gibt eine Liste aller J2EE-Module zurück, die auf dem abgefragten Server zwar deployed sind, aber zur Zeit nicht laufen.
- `distribute` schickt das komplette Modul, die Konfigurationsdaten und den zusätzlich erzeugten Code an den Server.
- `start` startet die Anwendung und macht sie zugänglich für die Clients.
- `stop` macht eine laufende Anwendung unzugänglich für die Clients. Diese Aktion ist nur für Wurzelmodule gültig. Sie bewirkt, dass sowohl das Wurzelmodul als auch alle seine Kindmodule gestoppt werden.

- `undeploy` entfernt eine Anwendung vom Server. Auch diese Aktion ist nur für Wurzelmodule gültig. Dabei werden zuerst alle Kindmodule und das Wurzelmodul gestoppt und anschließend entfernt.
- `redeploy(optional)` ersetzt eine Applikation bzw. ein Modul durch eine aktuellere Version. Der Vorgang muss für den Client transparent bleiben.

B.1.2 Deployment-Konfigurationsdateien

Der Java EE Produktanbieter muss auch Interfaces zu verschiedenen Deployment-Konfigurationskomponenten, wie z.B. `DeploymentConfiguration`, `DConfigBean`, `DConfigBeanRoot`, `Target` und `TargetModuleID`, anbieten.

- `DeploymentConfiguration` ist die oberste Komponente in der Konfigurationsstruktur des Deployments. Es stellt eine Art Container für alle Java EE produktspezifischen Konfigurationsobjekten dar.
- `DConfigBean` ist eine JavaBeans Komponente, die einen ganzen oder einen Teil eines Deployment-Deskriptors repräsentiert. Sie dient zum Vermitteln produktspezifischer Deployment-Konfiguration an das Deployment-Tool.
- `DConfigBeanRoot` ist die Wurzel-`DConfigBean` für einen Deployment-Deskriptor.
- `Target` stellt eine Verknüpfung zwischen einen Server oder Gruppe von Servern und die Ziellokation zum Deployen und Ausführen eines Java EE Moduls, was entsprechend vorbereitet wurde.
- `TargetModuleID` ist ein Bezeichner, der eindeutig einem deployed Anwendungsmodul zugeordnet wird. Jede `TargetModuleID` repräsentiert ein einzelnes Modul das auf einem bestimmten `Target` deployed wurde.

B.2 Deployment-Tool Anbieter

Der Deployment-Tool Anbieter liefert Implementierungen für die Schnittstellen aus dem Paket `javax.enterprise.deploy.model`. Sie bieten Methoden zum Navigieren durch Standard-Deployment-Deskriptoren und Behandlung von Modul Binary Code. Dabei spielen die Interfaces `DeployableObject` und `J2EEApplicationObject` eine wichtige Rolle. Ein `DeployableObject` ist eine abstrakte

Repräsentation eines J2EE Moduls (JAR, WAR, RAR, und EAR). Diese Schnittstelle ermöglicht den Zugang zu den Klassen- und Deskriptor-Dateien dieses Moduls. Ein `J2EEApplicationObject` repräsentiert ein EAR-Archiv und stellt somit einen Spezialfall eines `DeployableObject` dar, weil es eine zusätzliche Unterstützung von eingebetteten Modulen enthält. Die Deskriptoren können entweder als XML-Datenstrom oder über die `DDBeanRoot` und `DDBean` Schnittstellen, die auch vom Tool angeboten werden, eingelesen werden. Die `DDBeanRoot` stellt dabei die Wurzel in dem Deployment-Deskriptor. Jede `DDBean` stellt ein Fragment vom Deployment-Deskriptor dar und kann mittels *XPath* [W3C09] nach „Kindern“ durchsucht werden. Ein Objekt vom Typ `DDBean` stellt eine Art nicht modifizierbarer Zeiger auf der XML-Struktur. Die XML-Struktur wird durch einen sog. `XpathListener` beobachtet.

B.3 Gemeinsame Klassen

Dieser Teil enthält Klassen aus dem Paket `javax.enterprise.deploy.shared` implementiert durch Sun Microsystems, die zur Verbindung und Koordination der beiden anderen Teile dienen. Sie definieren verschiedene Standardbezeichner durch Aufzählungstypen. Die wichtigsten Klassen sind `ActionType`, `CommandType`, `StateType` und `ModuleType`.

Die `ActionType` definiert eine Aufzählung (*enumeration*), die zur Identifikation der Aktionen *cancel*, *stop* und *execute* dient. Diese können auf den *Deployment-Manager* als Befehl, definiert durch `CommandType`, ausgeführt werden. Dabei sind die Befehle beschrieben im Abschnitt B.1 verfügbar.

Die, in der `StateType` festgelegten, Werte bestimmen den Befehlszustand eines `DeploymentManagers`. Folgende Zustände sind dabei identifizierbar:

- *Completed*: Die Aktion wurde ordnungsgemäß beendet.
- *Failed*: Die Aktion ist fehlgeschlagen.
- *Released*: Der `DeploymentManager` läuft im sog. *disconnected* Modus (nicht verbunden mit einer Instanz des Java EE Produkts).
- *Running*: Die Aktion wird ordnungsgemäß durchgeführt.

Durch die Typen festgelegt in der `ModuleType`-Klasse werden die möglichen J2EE Modultypen definiert:

- *CAR*: Client Application Archive

- *EAR*: Enterprise Application Archive
- *EJB*: Enterprise Java Bean Archive
- *RAR*: Connector Archive
- *WAR*: Web Application Archive

C Testergebnisse der Auswertung typischer Rekonfigurationsszenarien

In diesem Anhang werden Testergebnisse, entnommen der Diplomarbeit von Stefan Hildebrandt [Hil05], aufgelistet. Als primäre Testplattform wurde ein Linux System [Lin09] mit Kernel 2.6.11, eingeschalteten Nativ Posix Threads [IEE09] und dem Blackdown JDK-1.4.2.01 auf einem Rechner mit 2 CPUs und 1GB RAM eingesetzt. Als Vergleichsplattform wurde ein Windows System mit Windows XP SP2, dem Sun JDK 1.4.2.07, einer CPU und 512MB RAM benutzt. Es wurde der JBoss Anwendungsserver [JBo09] in den Versionen 4.0.1 und 3.2.7 getestet. Zur Speicherung der Daten der Entity Beans wurde die mitgelieferte Datenbank HypersonicDB benutzt. Die Tests wurden ausschließlich mit Call-By-Reference durchgeführt, Call-By-Value im Container wurde abgeschaltet. Als Vergleichs-server diente der Anwendungsserver Bea WebLogic in der Version 8.1sp4 [Ora09].

C.1 JBoss 4.0.x unter Linux

Ergebnisse der Entity Bean Container-Managed-Relations Testsuite

Client	Remoteclient		Treiber	
	RemoteHome	Handle	RemoteHome	LocalHome
Vom Client benutztes Interface der Session Bean				
TestCMRAddDataBefore-AndAfterRedeploymentCommitTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁷
TestCMRAddDataBefore-AndAfterRedeploymentRollbackTx	✓ ⁴	✓ ⁴	✓ ⁴	✗ ⁷
TestCMRAddDataBefore-AndAfterRedeploymentEndTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁷
TestCMRAddDataWhileRedeploymentCommitTx	○ ^{2,5}	○ ^{2,5}	○ ^{2,5}	✗ ⁷
TestCMRAddDataWhileRedeploymentRollbackTx	○ ^{1,2,6}	○ ^{1,2,6}	○ ^{1,2,6}	✗ ⁷
TestCMRAddDataWhileRedeploymentOpenEndTx	○ ^{2,5,6}	○ ^{2,5,6}	○ ^{2,5,6}	✗ ⁷
TestCMRCreateEntityAndAddDataBeforeAndAfter-RedeploymentCommitTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁷
TestCMRCreateEntityAndAddDataBeforeAndAfter-RedeploymentRollbackTx	✓ ⁴	✓ ⁴	✓ ⁴	✗ ⁷
TestCMRCreateEntityAndAddDataBeforeAndAfter-RedeploymentOpenEndTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁷

¹ Zugriff vor und nach der Rekonfiguration ist möglich.

² Es ist kein Zugriff während der Rekonfiguration möglich.

³ Die vor der Rekonfiguration in der Transaktion gesetzten Daten gehen verloren.

⁴ Zufällig richtig, da nicht auffällt, dass Daten verlorengegangen sind.

⁵ Transaktion wurde fürs Rollback markiert, so ist noch ein Zugriff auf die Bean-Referenz möglich, aber die Methode wird nicht ausgeführt, sondern eine Ausnahme geworfen.

⁶ Transaktion wurde zurückgesetzt und alle Daten entfernt.

⁷ Die Referenz ist nach der Rekonfiguration nicht mehr benutzbar.

Tabelle C.1: Ergebnisse der Entity Bean Container-Managed Relations Testsuite

Ergebnisse der Entity Bean Testsuite

Client	Remoteclient		Treiber	
	RemoteHome	Handle	RemoteHome	LocalHome
Vom Client benutztes Interface der Session Bean				
TestEBCreateEntityInTx- BeforeAndAfterRedeployCommitTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁹
TestEBCreateEntityInTx- BeforeAndAfterRedeployRollbackTx	✓ ⁴	✓ ⁴	✓ ⁴	✗ ⁹
TestEBCreateEntityInTx- BeforeAndAfterRedeployOpenEndTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁹
TestEBGetDataWhileRedeployment	○ ^{1,2}	○ ^{1,2}	○ ^{1,2}	✗ ⁹
TestEBSetDataWhileRedeployment	○ ^{1,2,5}	○ ^{1,2,5}	○ ^{1,2,5}	✗ ⁹
TestEBSetDataWhileRedeploymentCommitTx	○ ^{1,2,6}	○ ^{1,2,6}	○ ^{1,2,6}	✗ ⁹
TestEBSetDataWhileRedeploymentRollbackTx	○ ^{1,2,7}	○ ^{1,2,7}	○ ^{1,2,7}	✗ ⁹
TestEBSetDataWhileRedeploymentOpenEndTx	○ ^{1,2,6,8}	○ ^{1,2,6,8}	○ ^{1,2,6}	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRedeploymentCommitTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRedeploymentRollbackTx	✓ ⁴	✓ ⁴	✓ ⁴	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRedeploymentOpenEndTx	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRDAddNoDataAfterRDCCommitTx	✓	✓	✓	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRDAddNoDataAfterRDRRollbackTx	✓	✓	✓	✗ ⁹
TestEBStartTx- BeforeAndEndAfterRDAddNoDataAfterRDOpenEndTx	✓	✓	✓	✗ ⁹
TestEBUseBeanBeforeAndAfterRedeployment	✓	✓	✓	✗ ⁹

¹ Zugriff vor und nach der Rekonfiguration ist möglich.

² Es ist kein Zugriff während der Rekonfiguration möglich.

³ Die vor der Rekonfiguration in der Transaktion gesetzten Daten gehen verloren.

⁴ Zufällig richtig, da das „Lost Update“ keinen Einfluss auf das Ergebnis hat.

⁵ Die Daten, die während der Rekonfiguration hinzugefügt werden sollten, fehlen.

⁶ Transaktion wurde fürs Rollback markiert, so ist noch ein Zugriff auf die Bean-Referenz möglich, aber die Methode wird nicht ausgeführt, sondern eine Ausnahme geworfen.

⁷ Die Transaktion wurde zurückgesetzt und alle Daten entfernt.

⁸ Die Transaktion wurde zurückgesetzt, aber nicht immer alle Daten entfernt.

⁹ Die Referenz ist nach der Rekonfiguration nicht mehr benutzbar.

Tabelle C.2: Ergebnisse der Entity Bean Testsuite

Ergebnisse der Message Driven Bean Testsuite

Von Message Driven Bean benutztes Interface der Session Bean	RemoteHome						LocalHome			
	Exception			Nur Zurücksetzen			Exception		Nur Zurücksetzen	
Verhalten bei Fehlern	Keine	Home-Cache	Narrow-Cache	Keine	Home-Cache	Narrow-Cache	Keine	Home-Cache	Keine	Home-Cache
TestFillQueue AndRedeployEntityBean	✓	✓	X ²	✓	✓	X ²	✓	✓	X ¹	X ¹
TestFillQueue AndRedeployMDB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TestFillQueue WithLongTests AndRedeployEntityBean	✓	✓	X ²	✓	✓	X ²	✓	X ³	X ¹	X ¹
TestFillQueue WithLongTests AndRedeployMDB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

¹ Da die Referenz nicht erneuert wird und nach dem Redeployment ungültig ist, treten zu viele Fehler auf, so dass die Nachricht in der DLQ landet.

² Die Referenz im Cache aus vorherigem Deployment sorgt für eine `ClassCastException`

³ Die genaue Ursache lässt sich nicht bestimmen, aber scheinbar ist die Last zu hoch.

Tabelle C.3: Ergebnisse der Message Driven Bean Testsuite (Zugriff auf Entity Beans)

Version der Bibliothek	Externes JAR	Kompatibel	Inkompatibel
Empfangen	X ¹	✓	X ^{2,3}
Senden	X ¹	✓	X ^{2,3}

¹ Die Message Driven Bean benutzt weiterhin die alte Version der Bibliothek.

² Rekonfiguration sollte zurückgewiesen werden, wird aber ausgeführt.

³ Nach der Rekonfiguration treten Serialisierungsprobleme bei der Übertragung des Value-Objektes auf.

Tabelle C.4: Ergebnisse der Message Driven Bean Testsuite (Bibliotheken)

Ergebnisse der Stateful Session Bean Testsuite

Client	Remoteclient		Treiber	
	RemoteHome	Handle	RemoteHome	LocalHome
Vom Client benutztes Interface der Session Bean				
TestHoldString-RunNoTestsWhileRedeploy	X ¹	X ¹	X ¹	X ¹
TestHoldString-RunTestsWhileRedeploy	X ¹	X ¹	X ¹	X ¹
TestHoldTestVO-RunNoTestsWhileRedeploy	X ¹	X ¹	X ¹	X ¹
TestHoldTestVO-RunNoTestsWhileRedeployDifferentLibVersions	X ¹	○ ²	X ¹	X ¹
TestHoldTestVO-RunNoTestsWhileRedeployIncompatibleLibVersions	X ¹	X ¹	X ¹	X ¹
TestHoldTestVO-RunTestsWhileRedeploy	X ¹	X ¹	X ¹	X ¹
TestHoldTestVO-RunTestsWhileRedeployDifferentLibVersions	X ¹	○ ²	X ¹	X ¹
TestHoldTestVO-RunTestsWhileRedeployIncompatibleLibVersions	X ¹	X ¹	X ¹	X ¹

¹ Die Session konnte nicht wiederhergestellt werden.

² Zugriff vor und nach der Rekonfiguration funktioniert. Version des Modells ist falsch.

Tabelle C.5: Ergebnisse der Stateful Session Bean Testsuite

Ergebnisse der Stateful Session Bean Testsuite mit Transaktionen

Client	Remoteclient				Treiber			
	RemoteHome		Handle		RemoteHome		LocalHome	
Vom Client benutztes Interface der Session Bean	RemoteHome	LocalHome	RemoteHome	LocalHome	RemoteHome	LocalHome	RemoteHome	LocalHome
Das von der Session Bean benutzte Interface der Entity Bean	RemoteHome	LocalHome	RemoteHome	LocalHome	RemoteHome	LocalHome	RemoteHome	LocalHome
TestSetDataWhile-RedeploymentCommitTx	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{1,3}	X ^{1,3}
TestSetDataWhile-RedeploymentRollbackTx	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{1,4}	X ^{1,4}
TestSetDataWhile-RedeploymentOpenEndTx	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{1,4}	X ^{1,4}
TestStartTxBeforeAndEnd-AfterRedeploymentCommitTx	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ^{2,3}	X ¹	X ¹
TestStartTxBeforeAndEnd-AfterRedeploymentRollbackTx	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ¹	X ¹
TestStartTxBeforeAndEnd-AfterRedeploymentOpenEndTx	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ^{2,5}	X ¹	X ¹

¹ Die Referenz auf die Bean ist defekt.

² Die Session konnte nicht wiederhergestellt werden.

³ Transaktion soll fürs Rollback markiert werden, was nicht funktioniert, da der Zugriff auf das Transaktionsobjekt nicht mehr funktioniert.

⁴ Die Transaktion kann nicht zurückgesetzt werden: `NullPointerException`.

⁵ Die Transaktion kann nicht zurückgesetzt werden, da die Session nicht wiederhergestellt werden konnte.

Tabelle C.6: Ergebnisse der Stateful Session Bean Testsuite mit Transaktionen

Ergebnisse der Stateless Session Bean Testsuite

Client Benutztes Interface der Session Bean	Remoteclient				Treiber					
	RemoteHome				LocalHome					
Wiederverwendung der Referenz	Keine	Home-Cache	Bean-Cache	Remoteclient mit Handle	Keine	Home-Cache	Bean-Cache	Keine	Home-Cache	Bean-Cache
TestGetString	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	X ⁴	X ⁴
TestGetString- ExpandedInterface	○ ¹	○ ¹	○ ¹	○ ¹	X ⁵	X ⁵	X ⁵	X ²	X ⁴	X ⁴
TestGetString- OtherBeanClassName	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	X ⁴	X ⁴
TestGetString- ReducedInterface	X ^{1,7}	X ^{1,7}	X ^{1,7}	X ^{1,7}	X ⁵	X ⁵	X ⁵	X ⁴	X ⁴	X ⁴
TestGetString- ReducedInterface- RemoveRequiredMethods	X ^{7,8}	X ^{7,9}	X ^{7,9}	X ^{7,8}	X ⁵	X ⁵	X ⁵	X ⁴	X ⁴	X ⁴
TestGetString- ReducedInterface- RemoveRequiredMethods- InBeanToo	X ^{7,8}	X ^{7,9}	X ^{7,9}	X ^{7,8}	X ⁵	X ⁵	X ⁵	X ⁴	X ⁴	X ⁴
TestGetTestVO	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	○ ¹	X ⁴	X ⁴
TestGetTestVO- ChangeLibOnly	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	○ ^{1,3}	X ⁴	X ⁴
TestGetTestVO- DifferentLibs	○ ¹	○ ¹	○ ¹	○ ¹	X ⁵	X ⁵	X ⁵	X ²	X ⁴	X ⁴
TestGetTestVO- DifferentLibVersions	○ ¹	○ ¹	○ ¹	○ ¹	X ⁵	X ⁵	X ⁵	X ²	X ⁴	X ⁴
TestGetTestVO- IncompatibleLibVersions	X ⁶	X ⁶	X ⁶	X ⁶	X ⁵	X ⁵	X ⁵	X ²	X ⁴	X ⁴
TestGetTestVO- ExpandedInterface	○ ¹	○ ¹	○ ¹	○ ¹	X ⁵	X ⁵	X ⁵	X ²	X ⁴	X ⁴

¹ Zugriff vor und nach der Rekonfiguration ist möglich. Während der Rekonfiguration kommt es zu fehlerhaften Anfragen.

² Kein Zugriff mehr nach der Rekonfiguration: `ClassCastException`.

³ Bibliotheksversion wurde nicht getauscht.

⁴ Es tritt eine `NullPointerException` im Stub auf.

⁵ Probleme mit dem „packaging“.

⁶ Es tritt eine `InvalidClassException` durch unterschiedliche SerialIDs der Klassen auf.

⁷ Das Redeployment sollte zurückgewiesen werden, es wird aber ausgeführt.

⁸ Kein Zugriff mehr nach der Rekonfiguration, da die Methode nicht aufgerufen werden kann.

⁹ Kein Zugriff mehr nach der Rekonfiguration. `IllegalStateException`

Tabelle C.7: Ergebnisse der Stateless Session Bean Testsuite

C.2 WebLogic unter Linux

Ergebnisse der Stateless Session Bean Testsuite

Client	Remoteclient				Treiber							
Benutztes Interface der Session Bean	RemoteHome									LocalHome		
	Keine	Home-Cache	Bean-Cache	Remoteclient mit Handle	Keine	Home-Cache	Bean-Cache	Keine	Home-Cache	Bean-Cache		
Wiederverwendung der Referenz	Keine	Home-Cache	Bean-Cache	Remoteclient mit Handle	Keine	Home-Cache	Bean-Cache	Keine	Home-Cache	Bean-Cache		
TestGetString	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		
TestGetString-ExpandedInterface	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		
TestGetString-OtherBeanClassName	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		
TestGetString-ReducedInterface	X ^{1,11}	X ^{2,12}	X ^{2,12}	X ¹³	X ^{2,3}	X ^{2,3}	X ^{2,7}	-,5	-,5	-,5		
TestGetString-ReducedInterface-RemoveRequiredMethods	X ^{2,10}	X ^{2,12}	X ^{2,12}	X ¹³	X ^{2,3}	X ^{2,3}	X ^{2,8}	-,5	-,5	-,5		
TestGetString-ReducedInterface-RemoveRequiredMethods-InBeanToo	X ^{2,10}	X ^{2,12}	X ^{2,12}	X ¹³	X ^{2,3}	X ^{2,3}	X ^{2,8}	-,5	-,5	-,5		
TestGetTestVO	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		
TestGetTestVO-ChangeLibOnly	-,5	-,5	-,5	-,5	-,5	-,5	-,5	-,5	-,5	-,5		
TestGetTestVO-DifferentLibs	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		
TestGetTestVO-DifferentLibVersions	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	○ ^{1,9}	-,5	-,5	-,5		
TestGetTestVO-IncompatibleLibVersions	X ⁴	X ^{2,12}	X ^{2,12}	X ¹³	X ⁴	X ⁴	X ^{2,6}	-,5	-,5	-,5		
TestGetTestVO-ExpandedInterface	✓	X ^{2,12}	X ^{2,12}	X ¹³	○ ¹	○ ¹	X ^{2,6}	-,5	-,5	-,5		

¹ Der Zugriff vor und nach der Rekonfiguration ist möglich, während der Rekonfiguration kommt es zu fehlerhaften Anfragen.

² Es ist kein Zugriff mehr nach der Rekonfiguration möglich.

³ Die Methode wird scheinbar über die Position der Methode im Interface identifiziert. Dabei kann die Methode nicht mehr gefunden werden.

⁴ Es tritt eine `InvalidClassException` durch unterschiedliche SerialIDs der Klassen auf.

⁵ Hierfür ist kein Deployment möglich.

⁶ Die Referenz ist ungültig geworden: ***** ASSERTION FAILED *****.

⁷ Greift scheinbar auf falsche Methode zu. (Exception im TestVO, welches in diesem Test nicht benutzt wird)

⁸ Der erwartete Stub ist nicht vorhanden: expected:<StatelessSessionBeanTests1> but was:<null>.

⁹ Das es funktioniert könnte nicht geklärt werden.

¹⁰ Die Referenz auf die Methode ist ungültig geworden: CORBA_BAD_OPERATION 0 No.

¹¹ Zugriff funktioniert, das Redeployment sollte aber zurückgewiesen werden.

¹² Die CORBA Referenz auf das Objekt ist ungültig geworden: CORBA_OBJECT_NOT_EXIST.

¹³ Findet die Klassen zum Serialisieren des Stubs nicht über den URL-Classloader.

Tabelle C.8: Testergebnisse der Stateless Session Bean-Testsuite

D Experimentergebnisse der Bestimmung und Erkennung minimaler Laufzeitabhängigkeiten

In diesem Anhang befindet sich die tabellarische Auflistung der Evaluationsergebnisse aus den individuellen Projekten von Lena Stöver [Stö07] und Eike Grüssing [Grü08] zu der Bestimmung und Erkennung von minimalen Laufzeitabhängigkeiten. Zusätzlich wird eine Auswahl der auswertenden Diagrammen aufgelistet. Diese Auswahl beinhaltet die wesentlichen Ergebnisse zu dem Rekonfigurationsauftrag für die Komponente `com.ibatis.jpjpetstore.service.CatalogService`, die eine häufig benutzte Komponente im Testsystem JPetStore darstellt [iBa09]. Die Ergebnisse zu den anderen getesteten Komponenten wurden in den individuellen Projekten von Lena Stöver und Eike Grüssing entsprechend dargestellt und ausgewertet.

D.1 Ergebnisse der Bestimmung von Laufzeitabhängigkeiten

Versuch	1	2	3
Anzahl angefangener Traces	168	1846	3650
Anzahl Intervalle mit min. Gewicht	61	1024	485
Anzahl min. Gewichte vs. Anzahl Traces	0,363	0,555	0,1328
Anzahl bei Gewicht = 0 vs. Anzahl Traces	0,363	0,472	0,121
Mittlere Dauer bei Gewicht = 0 [μs]	2542,246	1931,83	1738,15
Mittlere Dauer insgesamt [μs]	2261,21	2949,41	2542,25

Tabelle D.1: Ergebnisse für die Komponente `CatalogService` [Stö07]

Versuch	1	2	3
Anzahl angefangener Traces	168	1846	3650
Anzahl Intervalle mit min. Gewicht	106	1087	1096
Anzahl min. Gewichte vs. Anzahl Traces	0,63	1,076	0,3
Anzahl bei Gewicht = 0 vs. Anzahl Traces	0,63	0,958	0,27
Mittlere Dauer bei Gewicht = 0 [μs]	3735,49	3910,54	3676,17
Mittlere Dauer insgesamt [μs]	3735,49	4801,72	4174,33

Tabelle D.2: Ergebnisse für die Komponente ItemSqlMapDao [Stö07]

Versuch	1	2	3
Anzahl angefangener Traces	168	1846	3650
Anzahl Intervalle mit min. Gewicht	162	3552	1805
Anzahl min. Gewichte vs. Anzahl Traces	0,96	1,924	0,49
Anzahl bei Gewicht = 0 vs. Anzahl Traces	0,96	1,90	0,49
Mittlere Dauer bei Gewicht = 0 [μs]	4368,4	5587,74	4923,41
Mittlere Dauer insgesamt [μs]	4368,4	5697,74	4957,21

Tabelle D.3: Ergebnisse für die Komponente OrderSqlMapDao [Stö07]

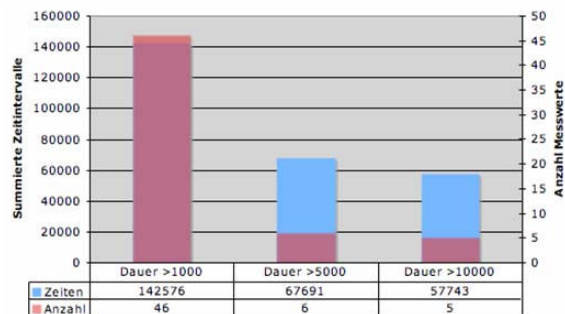


Abbildung D.1: Versuch 1 (1 Benutzer) [Stö07]



Abbildung D.2: Versuch 2 (10 Benutzer) [Stö07]



Abbildung D.3: Versuch 3 (20 Benutzer) [Stö07]

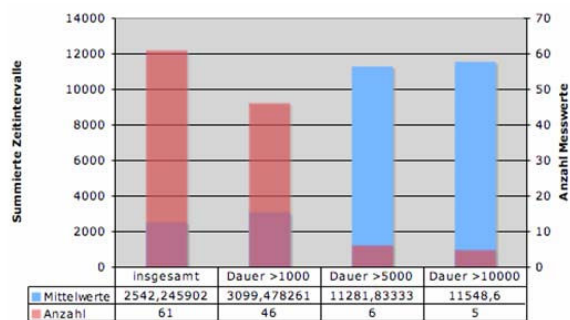


Abbildung D.4: Versuch 1 (1 Benutzer) [Stö07]

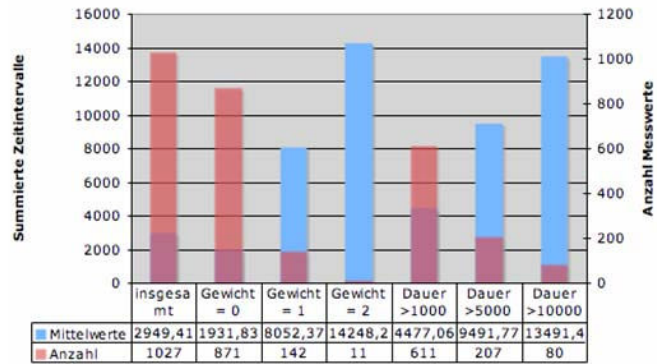


Abbildung D.5: Versuch 2 (10 Benutzer) [Stö07]

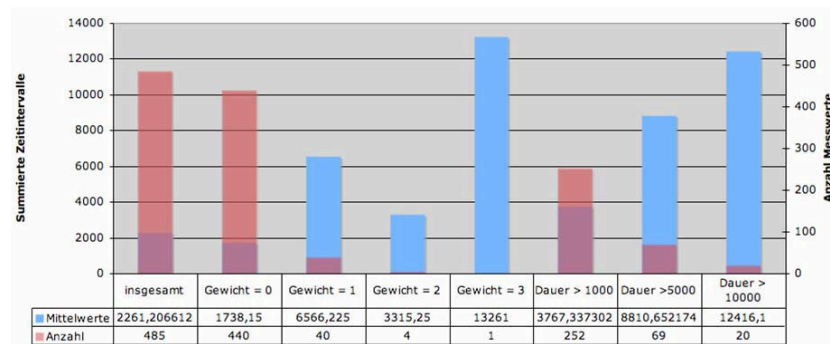


Abbildung D.6: Versuch 3 (20 Benutzer) [Stö07]

D.2 Ergebnisse der Wiedererkennung von Laufzeitabhängigkeiten

Anzahl Benutzer	1	10	20
Anzahl min. Abhängigkeit	192	2258	2559
Summierte Dauer der min. Abhängigkeit (s)	50,2	48,3	45,1
Durchschn. Dauer der min. Abhängigkeit (ms)	296	24	19
kürzeste min. Abhängigkeit (ms)	0.23	0.20	0.20
längste min. Abhängigkeit (ms)	4282	441	294
< 1 ms	81	976	1104
>= 1 ms und < 5ms	0	114	168
>= 5 ms und < 10 ms	0	161	221
>= 10 ms und <50 ms	5	602	721
>= 50 ms und < 1 s	99	405	345
>= 1 s	7	0	0

Tabelle D.4: Ergebnisse für die Komponente `CatalogService` [Grü08]

Anzahl Benutzer	1	10	20
Anzahl min. Abhängigkeit	97	922	1110
Summierte Dauer der min. Abhängigkeit (s)	50,1	49,8	49,6
Durchschn. Dauer der min. Abhängigkeit (ms)	555	60	50
kürzeste min. Abhängigkeit (ms)	0.59	0.12	0.27
längste min. Abhängigkeit (ms)	2632	889	467
< 1 ms	12	100	110
>= 1 ms und < 5ms	0	29	54
>= 5 ms und < 10 ms	0	82	93
>= 10 ms und <50 ms	4	315	460
>= 50 ms und < 1 s	68	396	393
>= 1 s	13	0	0

Tabelle D.5: Ergebnisse für die Komponente `ItemSqlMapDao` [Grü08]

Anzahl Benutzer	1	10	20
Anzahl min. Abhängigkeit	6	36	52
Summierte Dauer der min. Abhängigkeit (s)	44,7	45,1	44,2
Durchschn. Dauer der min. Abhängigkeit (ms)	7400	1200	999
Kürzeste min. Abhängigkeit (ms)	4400	23	7
Längste min. Abhängigkeit (ms)	15800	4500	3600
< 1 ms	0	0	0
>= 1 ms und < 5ms	0	0	0
>= 5 ms und < 10 ms	0	0	1
>= 10 ms und <50 ms	0	2	1
>= 50 ms und < 1 s	0	16	34
>= 1 s	6	18	16

Tabelle D.6: Ergebnisse für die Komponente OrderSqlMapDao [Grü08]

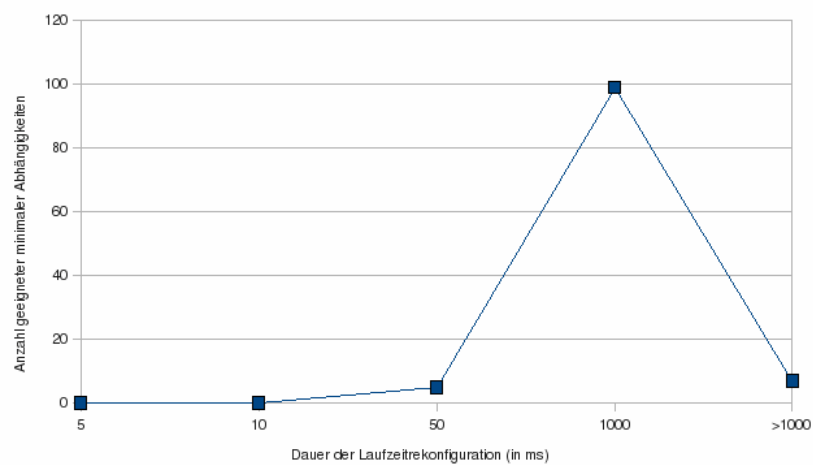


Abbildung D.7: Versuch 1 (1 Benutzer) [Grü08]

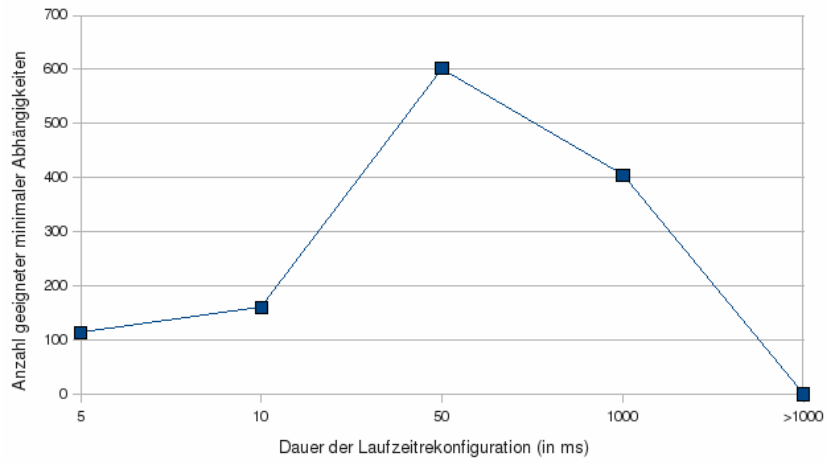


Abbildung D.8: Versuch 2 (10 Benutzer) [Grü08]

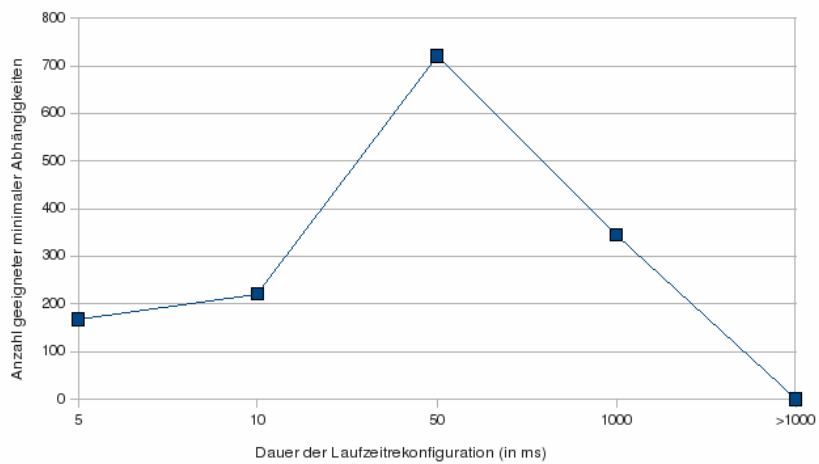


Abbildung D.9: Versuch 3 (20 Benutzer) [Grü08]

Literaturverzeichnis

- [BG02] BACKSCHAT, Martin ; GARDON, Otto: *Enterprise JavaBeans: Grundlagen - Konzepte - Praxis. EJB 2.0 / 2.1*. 1. Spektrum Akademischer Verlag, 2002
- [CM02] CHEUNG, Susan ; MATENA, Vlada ; SUN MICROSYSTEMS (Hrsg.): *Java Transaction API (JTA)*. <http://java.sun.com/products/jta>: Sun Microsystems, 2002
- [Grü08] GRÜSSING, Eike C.: *Erkennung von zur Laufzeit-Rekonfiguration geeigneten Nutzungsszenarien*. 2008. – Individuelles Projekt betreut durch Jasminka Matevska und Wilhelm Hasselbring, Software Engineering Group, Universität Oldenburg
- [Hil05] HILDEBRANDT, Stefan: *Evaluation typischer Rekonfigurationsszenarien an J2EE-basierten Anwendungen*. 2005. – Diplomarbeit betreut durch Jasminka Matevska und Wilhelm Hasselbring, Software Engineering Group, Universität Oldenburg
- [iBa09] IBATIS: *JPetStore*. <http://www.jwebhosting.net/jpetstore/>, 2009
- [IEE09] IEEE STANDARDS ASSOCIATION: *POSIX: Portable Operating System Interface*. <http://standards.ieee.org/regauth/posix/>, 2009
- [JBo09] JBOSS GROUP: *JBoss Application Plattform*. <http://www.jboss.com/products/platforms/application/>, 2009
- [Lin09] LINUX.ORG: *The Linux Home Page*. <http://www.linux.org/>, 2009
- [LR05] LANGNER, Torsten ; REIBERG, Daniel: *J2EE und JBoss*. Hanser Fachbuchverlag, 2005
- [Ora09] ORACLE: *Oracle BEA WebLogic*. <http://de.bea.com/products/weblogic/>, 2009
- [Pak03] PAKDAMAN, Mahboubeh: *Dynamische Rekonfiguration von Enterprise JavaBeans Softwaresystemen*. 2003. – Diplomarbeit betreut durch Jasminka Matevska und Wilhelm Hasselbring, Software Engineering Group, Universität Oldenburg
- [RAJ04] ROMAN, Ed ; AMBLER, Scott ; JEWELL, Tyler: *Struts-Tutorial - Ein Tutorial für Java-Entwickler*. John Wiley & Sons, Inc - New York, 2004
- [Sea03] SEARLS, Rebecca ; SUN MICROSYSTEMS (Hrsg.): *J2EE Deployment API Specification [JSR-88]*. <http://java.sun.com/j2ee/tools/deployment/>: Sun Microsystems, 2003
- [Stö07] STÖVER, Lena: *Evaluation dienstbezogener Abhängigkeiten in komponentenbasierten Systemen*. 2007. – Individuelles Projekt betreut durch Jasminka Matevska und Wilhelm Hasselbring, Software Engineering Group, Universität Oldenburg

- [Sun04] SUN MICROSYSTEMS: *Java Platform Standard Edition 5.0*. <http://java.sun.com/j2se/1.5.0/docs/index.html>, 2004
- [Sun06a] SUN MICROSYSTEMS: *Java Remote Method Invocation Specification*. <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>, 2006
- [Sun06b] SUN MICROSYSTEMS: *JSR 220: Enterprise JavaBeans Specification, Version 3.0*. <http://java.sun.com/products/ejb/>, 2006
- [Sun09] SUN MICROSYSTEMS: *Java Platform, Enterprise Edition (Java EE)*. <http://java.sun.com/javaee/>, 2009
- [W3C09] W3C RECOMMENDATION: *XML XPath Language (XPath) 2.0*. <http://www.w3.org/TR/xpath20/>, 2009
- [XT09] XDOCLET TEAM, Sourceforge.net: *XDoclet: Attribute-Oriented Programming*. <http://xdoclet.sourceforge.net/xdoclet/>, 2009