17. WebSphere Application Server

17.1 z/OS as a Unix System

17.1.1 Unix Betriebssysteme

Welche Server-Betriebssysteme findet man in der Praxis ?

- **Windows, verschiedene Varianten**
- **Unix, verschiedene Varianten**
- **i-Series, OS/400**
- **zSeries Betriebssysteme – z/OS, z/VM, z/VSE, TPF**

Welche wesentlichen Unix Varianten existieren ?

- **HP/UX**
- **SunSolaris**
- **IBM AIX**
- **Siemens Sinix**
- **MacOS (BSD)**
- **Linux, einschließlich zLinux**
- **z/OS Unix System Services**

} **Unix Systeme sind weitestgehend, aber nicht 100 % kompatibel**

Führende Unix Großrechner sind:

- **Integrity Superdome von HP mit Itanium Prozessoren und dem HP-UX Betriebssystem**
- **M9000 bzw. SPARC M5-32 Server von Oracle mit Sparc Prozessoren und dem Solaris Betriebssystem. Ein verwandtes Produkt wird von der Firma Fujitsu unter dem Namen „SPARC Enterprise Server“ vertrieben.**
- **System p von IBM mit PowerPC Prozessoren und dem AIX Betriebssystem**

Neben den proprietären Unix Dialekten ist auf diesen Rechnern auch Linux verfügbar.

Performance Unix versus z/OS

Die I/O Leistung eines Rechners wird gemessen in der Anzahl von I/O Operationen pro Sekunde unter realistischen Betriebsbedingungen. Konkrete Untersuchungen sind nie veröffentlicht worden, aber es wird allgemein angenommen, dass die z/OS I/O Leistung vielleicht um einen Faktor 3 - 10 höher als die I/O Leistung von großen Unix Rechnern wie Superdome und M9000 ist.

17.1.2 Was definiert ein Unix System ?

Unix wurde ursprünglich von Ken Thompson und Dennis Ritchie an den Bell Telephone Laboratories für Rechner der Digital Equipment Corporation (DEC) entwickelt und 1971 erstmalig im praktischen Betrieb eingesetzt. Schon bald entstanden (fast) kompatible Implementierungen für andere Rechner.

Aus Bemühungen um einen einheitlichen Unix Standard entstand die Portable Operating System Interface (POSIX). Dies ist ein gemeinsam von der IEEE und der Open Group für Unix entwickeltes standardisiertes Application Programming Interface, welche die Schnittstelle zwischen Applikation und dem Betriebssystem darstellt.

X/Open ist eine Erweiterung des POSIX Standards, welche ursprünglich von einem Konsortium mehrerer Europäischer Hersteller von Unix Systemen erarbeitet wurde. Die Erweiterung wurde in mehreren X/Open Portability Guides (XPG) veröffentlicht. Die letzten Versionen haben die Bezeichnungen XPG4 und XPG4.2.

Ein Betriebssystem ist ein Unix System, wenn es den Posix (1003.1 and 1003.2) Standard und die X/Open XPG4 und XPG4.2 Standards erfüllt. Linux wurde nie Posix und XPG4.2 zertifiziert, dürfte aber zu 99,9 % Posix und XPG 4.2 kompatibel sein. Ob Linux als Unix System bezeichnet werden sollte, ist umstritten.

Es hat mehrfache Unix Implementierungen für Mainframes gegeben. In der Vergangenheit war Amdahl UTS (Universal Time Sharing System) am erfolgreichsten. Es lief seinerzeit auf etwa 300 Mainframe Rechnern. Die AT&T Portierung von Unix System V lief auf etwa 30 Mainframes. Andere Beispiele von geringerer Bedeutung waren Hitachi HI-OSF/1-M und IBM AIX/ESA.

HP-UX (HP), Solaris (SUN/Oracle), AIX (IBM) und z/OS Unix System Services (IBM) sind heute die wichtigsten Unix Betriebssysteme. Sie sind POSIX und X/Open zertifiziert. Dies garantiert, dass der Quellcode beliebiger Anwendungen mit minimalem Aufwand von einem Unix System auf ein anderes Unix System portiert werden kann, obwohl die Implementierungen sehr unterschiedlich sind.

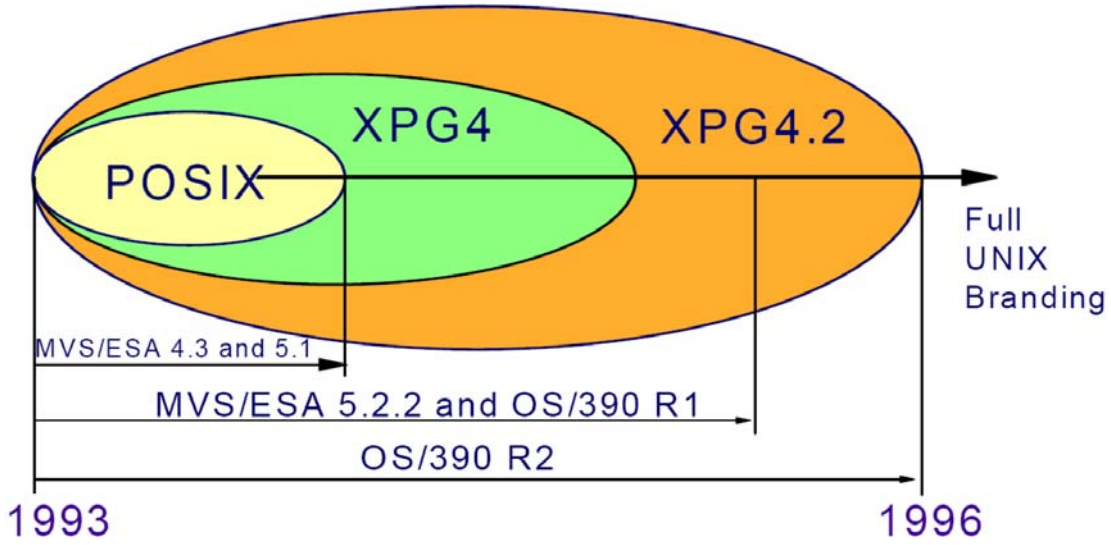
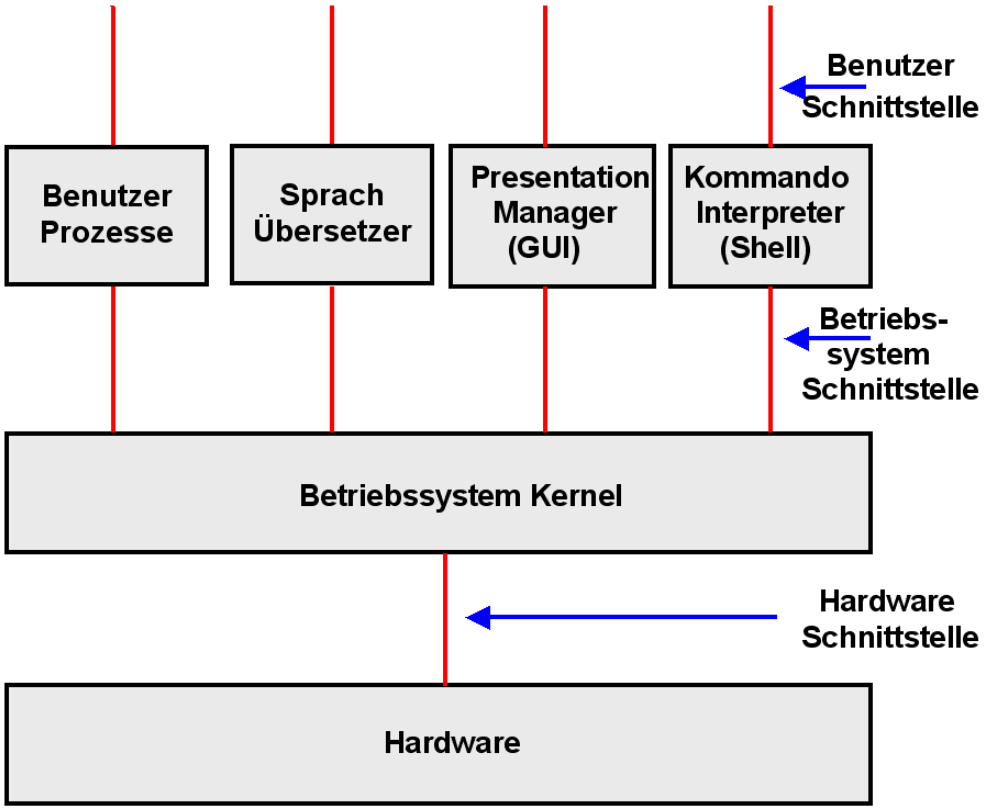


Abb.17.1.1
Posix und XPG4 Modell

Z/OS „Unix System Services“ wurde 1993 erstmalig unter dem Namen Open MVS (OMVS) verfügbar. Es gilt als vollwertiges Unix Betriebssystem.

Wie ist es möglich, dass z/OS gleichzeitig ein Unix Betriebssystem sein kann ?

17.1.3 Schichtenmodell der Rechnerarchitektur



Der z/OS Kernel (Supervisor, Überwacher) wird auch als „Basic Control Program“ oder BCP bezeichnet.

Abb.17.1.2
Schichtenmodell der Rechner Architektur

Betriebssystem Kernels haben grundsätzlich immer eine sehr ähnliche Struktur. Sie bestehen aus den in Abb. 17.1.3 dargestellten Komponenten.

Diese Folien sind teilweise eine Wiederholung von Band 1, Kapitel 2, Verarbeitungs-Grundlagen.

17.1.4 Struktur des Supervisors

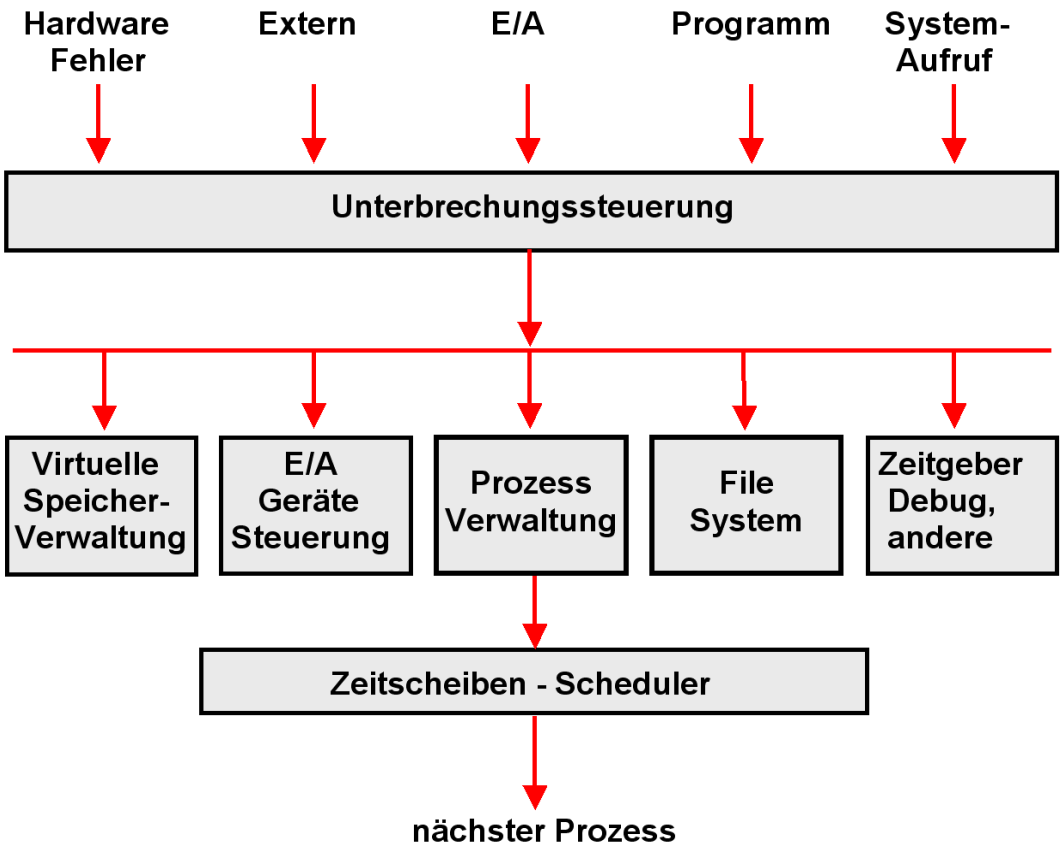


Abb.17.1.3
Struktur des Überwachers

Der Aufruf des Überwachers (Supervisor, Kernel) erfolgt grundsätzlich über Unterbrechungen.

Je nach Art der Unterbrechung werden unterschiedliche Komponenten des Überwachers aufgerufen.

Benutzerprozesse nehmen Dienste des Überwachers über eine architekturierte Schnittstelle, den Systemaufruf (System Call, Supervisor Call, SVC) in Anspruch. Diese Begriffe haben alle die gleiche Bedeutung.

Der Scheduler (Zeitscheibensteuerung) sucht den nächsten auszuführenden Prozess aus.

17.1.5 Supervisor Calls

z/OS SVC's (Supervisor Calls) sind das Äquivalent zu den Unix System Calls.

Supervisor Calls im weiteren Sinne sind Library Routinen, die eine Dienstleistung des z/OS Kernels in Anspruch nehmen. Andere Bezeichnung: System Calls.

Supervisor Calls im engeren Sinne sind Maschinenbefehle, die mittels eines Übergangs vom User Mode (Problem Status) zum Kernel Mode (Supervisor Status) einen Interrupt Handler des Kernels aufrufen. Äquivalente Maschinenbefehle in der X86 (Pentium) Architektur sind int 0x80 Call Gate, und SYSENTER. Für die IA-64 Architektur wird der EPC (Enter Privileged Mode) Maschinenbefehl benutzt.

Ein SVC Maschinenbefehl enthält einen 8 Bit Identifier, welcher die Art des Supervisor Calls identifiziert.

Beispiele sind:

GETMAIN	**SVC 10**	**Anforderung von Virtual Storage**
OPEN	**SVC 19**	**Öffnen eines Data Sets**
EXCP	**SVC 0**	**Lesen oder Schreiben von Daten**
WAIT	**SVC 19**	**Warten auf ein Ereignis, z.B. Abschluss einer Lese Operation**

Unix System Services sind eine Erweiterung des z/OS Kernels (BCP) um 1100 Unix System Calls, die als z/OS SVCs implementiert sind.

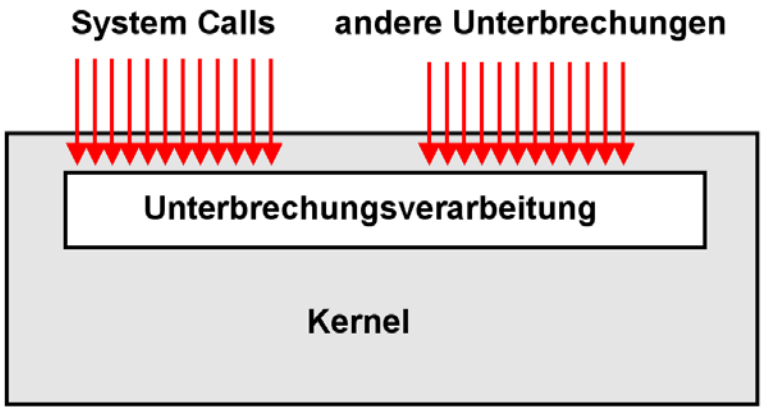


Abb.17.1.4
System Calls werden vom Überwacher wie Unterbrechungen behandelt

Die Kernel (Überwacher) aller Betriebssysteme haben de facto identische Funktionen, z. B.

- **Unterbrechungsverarbeitung**
- **Prozessmanagement**
- **Scheduling/Dispatching**
- **Ein/Ausgabe Steuerung**
- **Virtuelle Speicherverwaltung**
- **Dateimanagement**

Ein Unix Kernel unterscheidet sich von einem Windows Kernel durch die Syntax und Semantik der unterstützten System Calls, seine Shells sowie durch die unterstützen Dateisysteme.

Linux, Solaris, HP-UX und AIX haben unterschiedliche Kernel, aber (nahezu) identische System Calls..

Der Kernel eines Betriebssystems wird fast ausschließlich über Unterbrechungen oder über System Calls aufgerufen.

17.1.6 Unix System Services

Die Unix System Services (USS) des z/OS Betriebssystems sind eine Erweiterung des z/OS Kernels um 1100 Unix System Calls, zwei Unix Shells und zwei Unix Datei-Systeme. Sie erfüllen den POSIX Standard.

Damit wird aus z/OS ein Unix Betriebssystem. Dies ermöglicht eine einfache Portierung von Unix Anwendungen nach z/OS. Ein typisches Beispiel ist das SAP R/3 System, welches ursprünglich für das Unix Betriebs system entwickelt wurde, aber auch unter z/OS Unix System Services lauffähig ist.

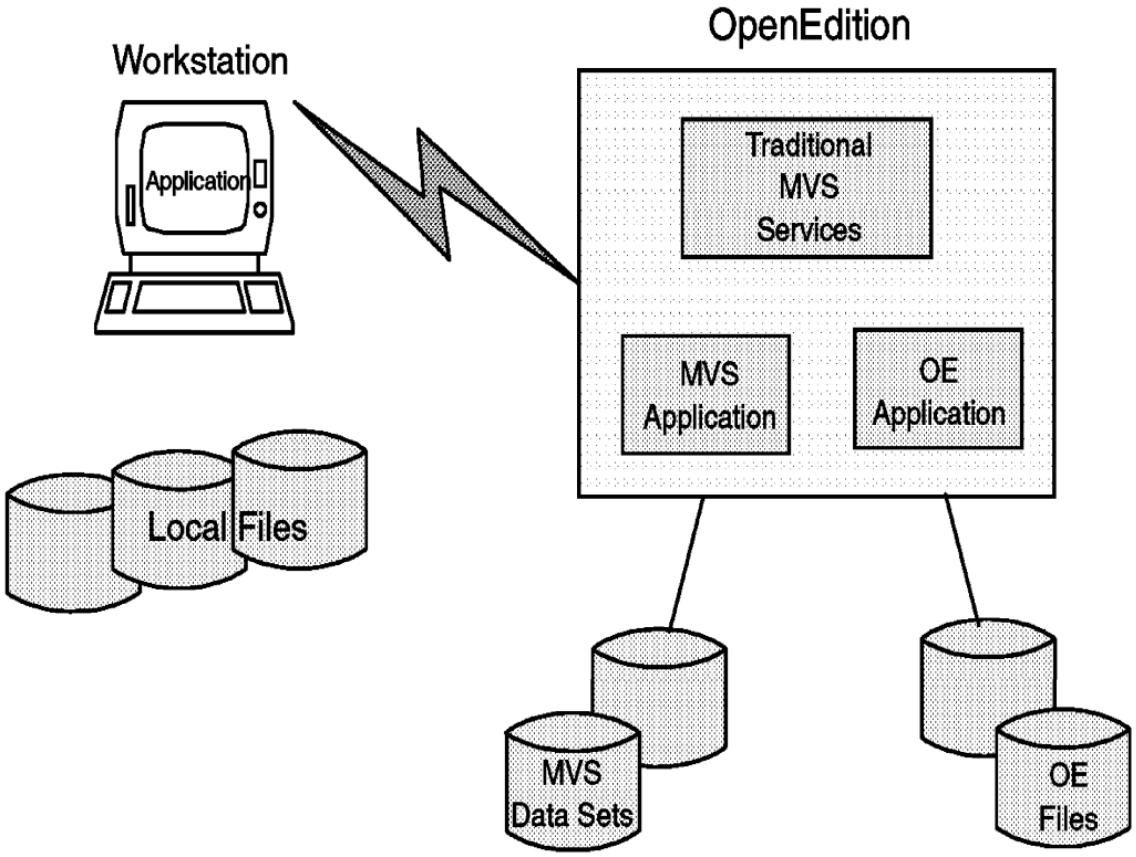


Abb.17.1.5
Normale z/OS Programme können auch auf Unix System Service Files zugreifen

z/OS verhält sich entweder wie ein traditionelles z/OS (MVS) System oder wie ein Unix System.

z/OS Unix System Services wurde früher als Open Edition (OE) oder Open MVS bezeichnet.

17.1.7 z/OS UNIX System Services Komponenten

Unix System Services besteht aus diesen Komponenten:

z/OS UNIX Shell und Utilities

Eine interaktive Interface zu z/OS UNIX Services, welche Commands von interaktiven Benutzern und von Programmen interpretiert.

Die Shell und Utilities Komponente kann mit den TSO-Funktion unter z/OS verglichen werden.

Hierarchical File System

Neben den vorhandenen z/OS Dateisystemen (z.B. VSAM, PDS) wurde ein zusätzliches Unix kompatibles hierarchischen Filesystem eingerichtet. Jedes z/OS Programm kann auf das hierarchische Filesystem zugreifen.

z/OS UNIX debugger (dbx)

Die z/OS UNIX-Debugger ist ein Werkzeug, das Anwendungsprogrammierer verwenden, um interaktiv ein C-Programm zu debuggen. Der dbx-Debugger ist nicht Teil des POSIX-Standards. Es basiert auf dem Unix dbx Debugger, der auch in vielen UNIX-Umgebungen vorhanden ist.

C/C++ Compiler and C run-time Libraries

Die C/C++ Compiler und C-Laufzeitbibliotheken werden benötigt, um ausführbare Dateien zu erstellen, die der Kernel verstehen und verwalten kann. Die C/C++ Compiler und Language Environment (LE) Feature-Bibliothek wurde verändert und erweitert, um Unterstützung für die POSIX-und XPG4 C Funktionsaufrufe zu enthalten. Das LE Produkt stellt eine gemeinsame Laufzeitumgebung und sprachspezifische run-time Services für kompilierte Programme zur Verfügung.

Um einen Shell-Befehl oder Dienstprogramm, oder ein vom Benutzer bereitgestelltes Anwendungsprogramm in C oder C++ auszuführen, benötigen Sie die C/C++ Runtime Library, die mit LE zur Verfügung gestellt wird.

Beim z/OS Systemstart (IPL) werden UNIX System Services Kernel-Dienste automatisch gestartet. Der Kernel bietet z/OS USS als Service für Anforderungen von Unix Programmen und von der USS Shell an. Der Kernel verwaltet ein Unix hierarchisches Filesystem (HFS), die Kommunikationseinrichtungen und die UNIX System Services (USS) Prozesse. Das hierarchische Filesystem wird als Teil des Kernels betrachtet, und als Komponente des DFSMS (Data Facility Storage Management Subsystem) installiert. Siehe Band 1, Abschnitt 3.3.9.

Auch nicht-Unix Programme können auf das hierarchische File System zugreifen. So kann z.B. ein CICS Programm im Zusammenhang mit CICS Web Support auf HFS zugreifen.

Um diese APIs zu unterstützen, muss das z/OS-System einige System-Dienste in seinem Kernel enthalten, z. B. das hierarchische Filesystem-und Kommunikationsdienstleistungen.

Daemons sind Programme, die typischerweise gestartet werden, wenn das Betriebssystem initialisiert wird. Sie bleiben aktiv, um Standard Services auszuführen. z/OS UNIX (USS) enthält Daemons, die auf Anforderung Anwendungen oder eine User Shell Sitzung starten.

17.1.8 Unix System Services Struktur

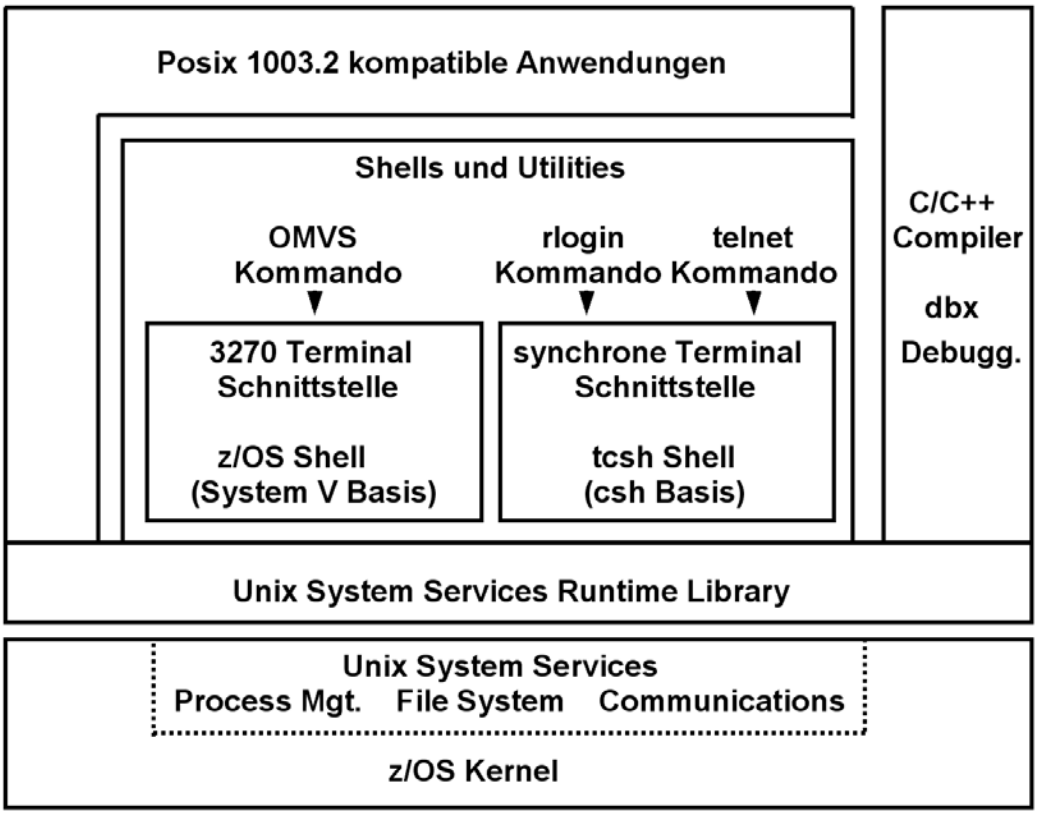


Abb.17.1.6
Komponenten der Unix System Services Erweiterung

Die Unix System Services Kernel Funktion läuft in einem eigenen virtuellen Adressenraum, der als Teil des IPL (Boot) Vorgangs hochgefahren wird. Er wird wie jeder Unix Kernel über eine API aufgerufen, die aus C/C++ Function Calls besteht.

Das Byte-orientierte hierarchische File System arbeitet wie jedes Unix File System. Es wird in z/OS linear VSAM Data Sets abgebildet. Alle Files sind dem Unix System Services Kernel zugeordnet, und alle Ein-/Ausgabe Operationen bewirken Calls in den Kernel.

Häufig werden auf dem gleichen Mainframe Rechner Unix System Services und zLinux parallel betrieben.

17.1.9 TSO und UNIX System Services

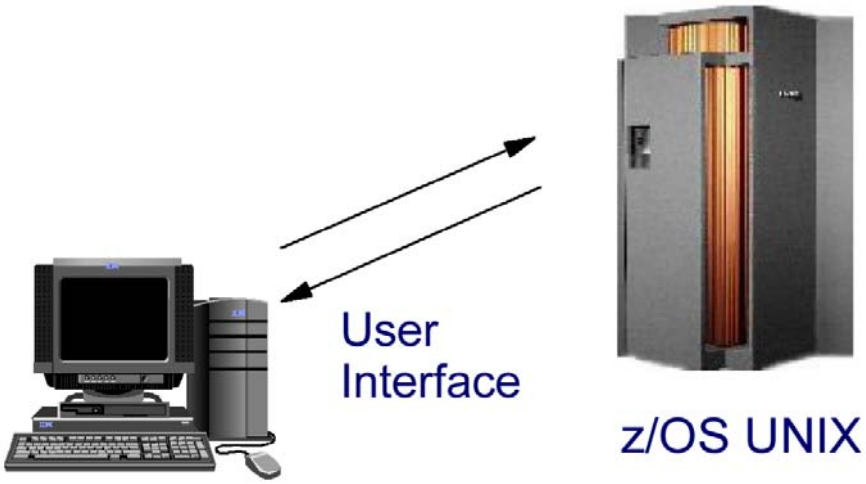


Abb.17.1.7
Zwei unterschiedliche Unix System Service Shells

Es existieren zwei alternative Möglichkeiten für den Zugriff auf Unix System Services (zwei unterschiedliche Shells).

Eine Möglichkeit ist die Benutzung der z/OS Shell mittels eines 3270 Terminals und TSO. Die z/OS Shell gleicht der Unix System V Shell mit einigen zusätzlichen Eigenschaften der Korn Shell. Sie benutzt den ISPF Editor. Ein Benutzer startet eine TSO Session und gibt das OMVS Kommando ein. Dies ist die bevorzugte Methode für TSO Experten.

Die andere Möglichkeit ist die tcsh Shell. Diese ist kompatibel mit der csh Shell, der Berkley Unix C Shell. Sie wird über rlogin oder telnet (z.B. mittels putty) aufgerufen und verwendet den vi Editor. Dies ist die bevorzugte Methode für Unix Experten.

17.1.10

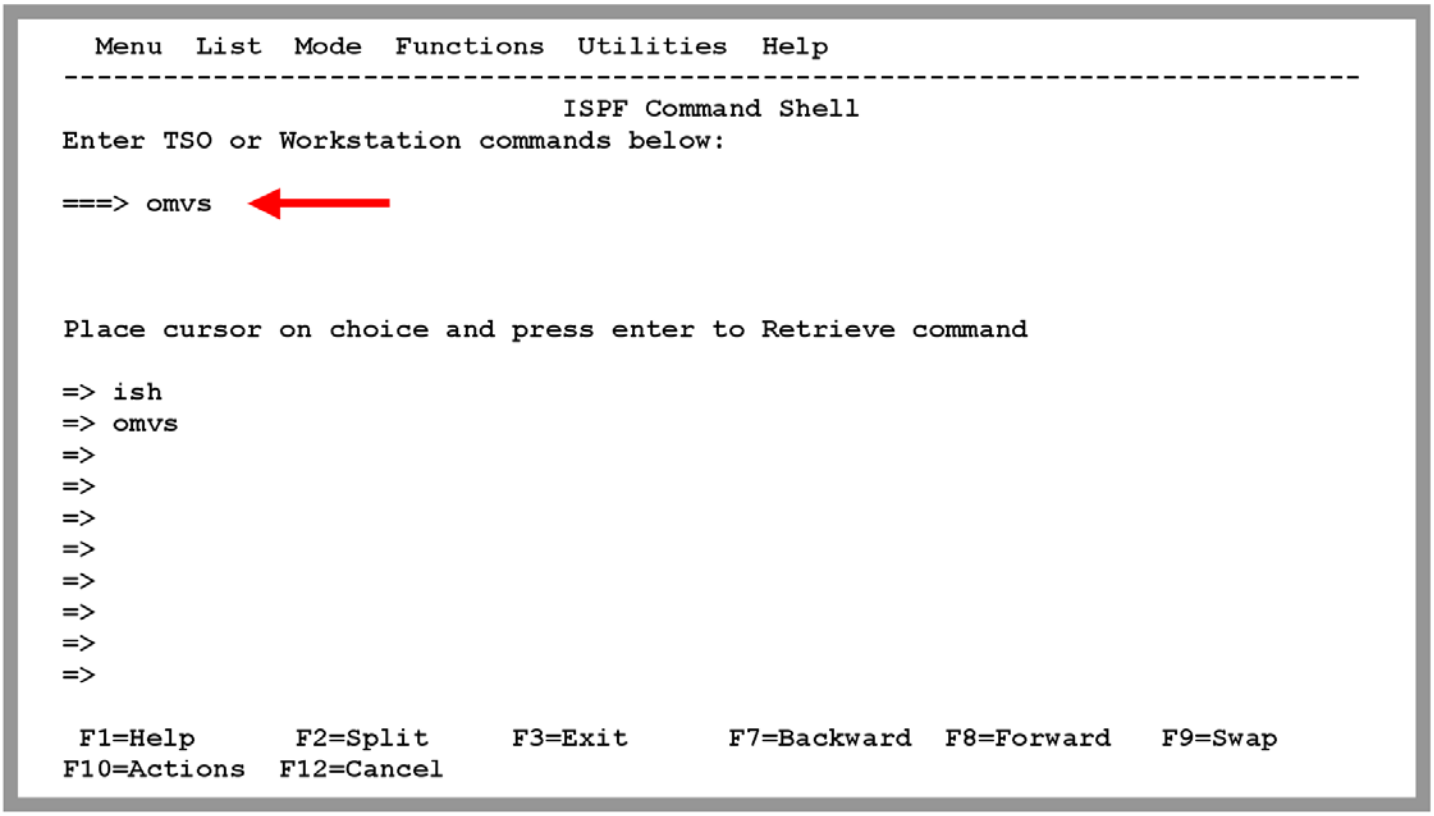


Abb.17.1.8
Das OMVS Kommando ruft die TSO USS Shell auf

Die Unix System Services Shell wird vom ISPF Command Shell Screen durch die Eingabe des TSO Kommandos “OMVS” gestartet.

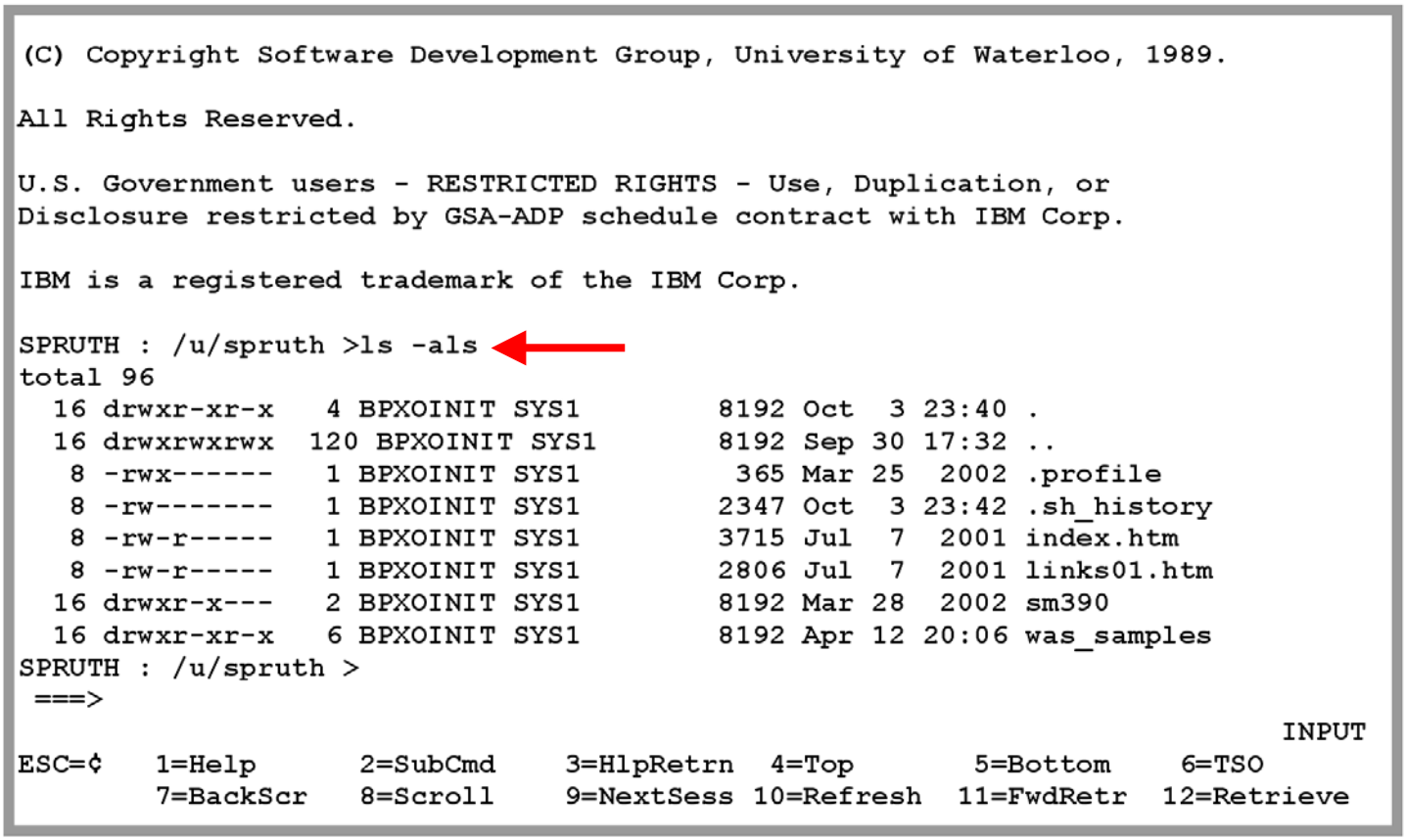


Abb.17.1.9
Die TSO USS Shell benutzt alle üblichen Unix Befehle

In dem gezeigten z/OS Shell Beispiel hat der Benutzer /u/spruth den Unix Befehl `ls –als` eingegeben

17.1.11 Logical File System

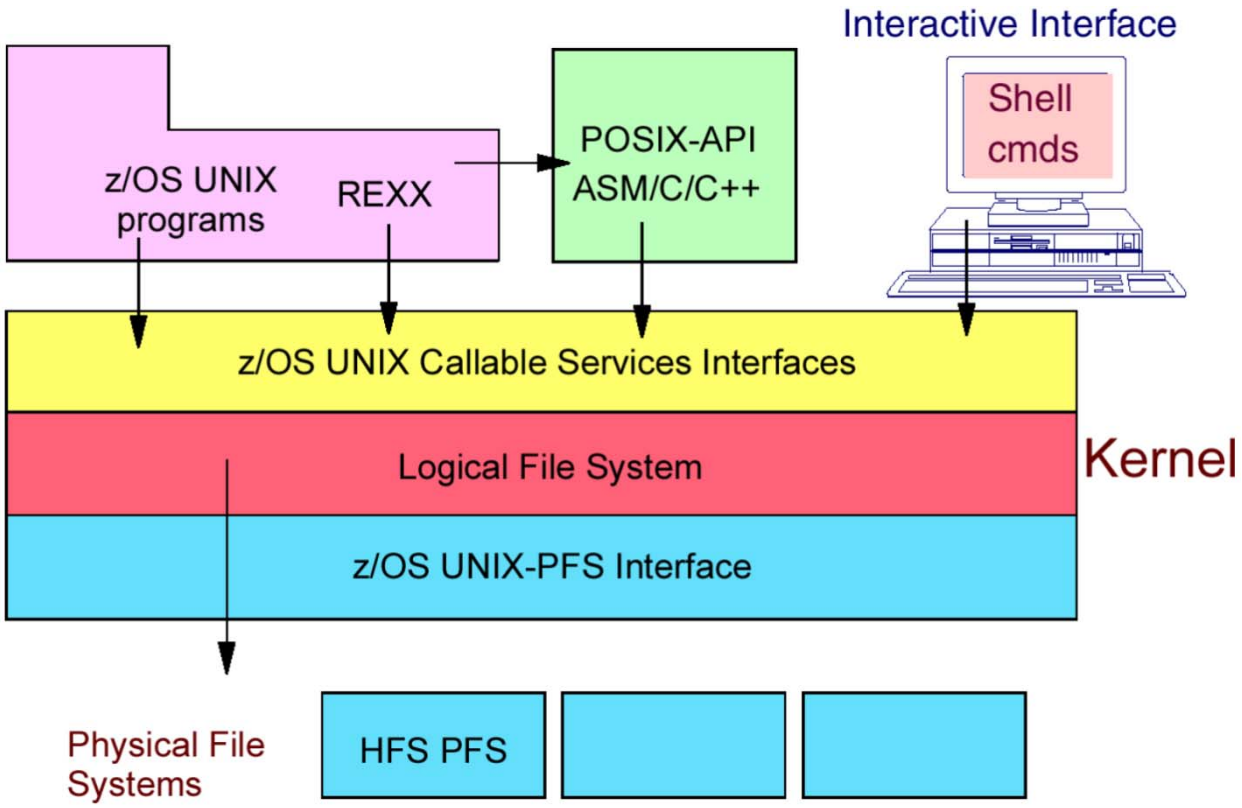


Abb.17.1.10
Hinter dem logischen File System verbergen sich mehrere physische file Systeme

Mit beiden Shells kann ein Standard Unix hierarchisches Filesystem benutzt werden. z/OS verfügt über ein logisches hierarchisches File System, welches auf eins von mehreren verfügbaren physischen File Systemen abgebildet wird. So wie Linux über mehrere physische File Systems verfügt (ext2, ext3, ext4, ReiserFS), existieren auch für z/OS USS mehrere physische File Systeme.

REXX ist eine populäre z/OS und USS Script Sprache.

Die USS Kernel Funktionen sind ein Bestandteil des z/OS Supervisors.

17.1.12 Benutzung von z/OS UNIX System Services

Ein Benutzer kann mit z/OS UNIX über die folgenden Schnittstellen interagieren:

- **Das Application Programming Interface (API) besteht aus Aufrufen (Calls), die von C/C++ Programmen verwendet werden können, um auf z/OS UNIX zuzugreifen. Diese C Aufrufe sind in dem POSIX 1003.1 Standard definiert.**

  Die aufrufbaren Dienstleistungen können direkt vom Assembler-Programmen verwendet werden, um auf z/OS UNIX zuzugreifen, z. B. um auf Dateien in dem hierarchischen Filesystem zuzugreifen. Diese Möglichkeit erlaubt es anderen Hochsprachen und Assembler, z/OS UNIX zu verwenden. Die API-Schnittstelle bietet die Möglichkeit, XPG4.2 Programme auf z/OS auszuführen. Ein Programm, das dem XPG4.2 Standard entspricht, kann auf einem System entwickelt werden, und dann auf ein anderes System portiert werden, dort kompiliert und gelinked werden, und dann auf diesem ausgeführt werden. Ein solches Programm wird als portable bezeichnet.

- **Die interaktive Schnittstelle wird als z/OS UNIX-Shell bezeichnet. Die Shell ist ein Kommando-Interpreter, der Befehle akzeptiert, die in dem POSIX 1003.2 Standard definiert sind. Shell-Befehle können in eine Reihenfolge gebracht werden, in einer Textdatei als ein Shell-Skript gespeichert werden, und dann ausgeführt werden. Das Shell-Skript arbeitet ähnlich wie z/OS CLISTs oder REXX EXECs.**

  TSO REXX enthält Erweiterungen, um den Zugang zu den z/OS UNIX abrufbaren Dienstleistungen zu ermöglichen. Eine REXX EXEC mit UNIX System Services kann von TSO/E ausgeführt werden, als z/OS Batch Job oder in der Shell.

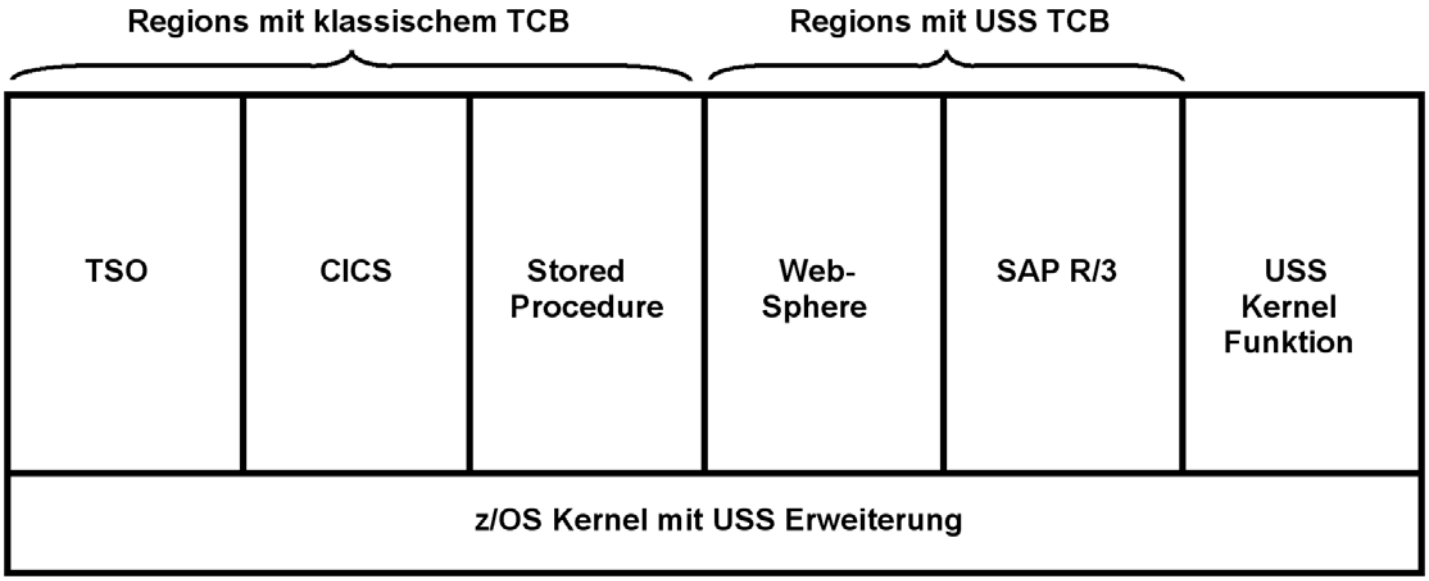


Abb.17.1.11
USS Regions nutzen den USS TCB

Regions, welche Unix System Services einsetzen benutzen einen modifizierten (erweiterten) TCB. Programme, die unter dem klassischen TCB laufen, können auf bestimmte USS Funktionen zugreifen, z.B. das USS hierarchische File System.

Die Unix System Services Kernel Funktion läuft in einem eigenen virtuellen Adressenraum, der als Teil des IPL (Boot) Vorgangs hochgefahren wird.

17.1.13 Unix System Services versus zLinux

Für die heutigen Mainframes sind heute zwei Unix Implementierung von Bedeutung:

- **zLinux**
- **z/OS Unix System Services (USS)**

Sie sind vor allem von Interesse, wenn existierende Unix/Linux Anwendungen von einer distributed Platform auf den Mainframe portiert werden sollen.

Warum zwei Alternativen, und was ist der Unterschied ?

zLinux ist ein regulärer Port von Suse oder Red Hat Linux auf System z Hardware. Es läuft entweder in einer eigenen LPAR, oder als virtuelle Maschine unter z/VM. Beide Alternativen sind auf Mainframe Servern relativ weit verbreitet und werden fast immer nicht an Stelle von z/OS, sondern parallel zu z/OS benutzt, siehe Abb. 12.3.8 .

Auf der Betriebssystem Ebene ist der zLinux Kernel performanter als der z/OS Kernel. Es fehlen aber alle z/OS spezifischen Funktionen, beispielsweise Unterstützung für Sysplex, Coupling Facility, Work Load Manager, RACF, FICON, Protection Keys, Krypto und vieles mehr. Je nachdem ob diese Funktionen gebraucht werden, laufen Unix Programme entweder unter zLinux oder USS. Häufig wird in einer Mainframe Installation beides genutzt.

Im Gegensatz zu zLinux ist Unix System Services (USS) kein eigenes Betriebssystem, sondern ein integrierter Bestandteil von z/OS. Es ist damit in der Lage, alle z/OS Funktionen zu nutzen. Es ermöglicht eine relativ problemlose Portierung von existierenden Unix Anwendungen auf einen Mainframe Server, ohne dabei auf existierende z/OS Eigenschaften verzichten zu müssen.

Zwei der wichtigsten Software Pakete, die unter z/OS Unix System Services laufen (oder auch unter zLinux), sind:

- **SAP/R3**
- **WebSphere**

17.1.14 Microsoft Windows Services for UNIX

Microsoft Windows Services for UNIX (SFU) ist ein Software-Paket von Microsoft, welches ähnlich wie z/OS Unix System Services arbeitet. Es handelt sich hierbei um ein Unix-Subsystem (und weitere Komponenten einer Unix-Umgebung) nach dem POSIX-Standard, welches unter den Windows Betriebssystemen verfügbar ist. Dieses Subsystem wird als Interix, SFU oder SUA bezeichnet. Siehe
http://de.wikipedia.org/wiki/Microsoft_Windows_Services_for_UNIX

Microsoft Windows Services for UNIX setzt als Implementierung eines User-Mode-Subsystems unmittelbar auf dem Windows-Kernel auf. Die aktuelle Version trägt die Nummer 3.5. Als Veröffentlichungsdatum wird der 21. September 2006 angegeben.

Microsoft Windows Services for UNIX kann auf den folgenden Windows-Varianten installiert werden:

- **Windows XP Professional,**
- **Windows Server 2003,**
- **Windows Server 2008,**
- **Windows 7 Enterprise und Ultimate.**

Microsoft Windows Services for UNIX hat als Bestandteil von Windows eine in etwa vergleichbare Funktionalität wie Unix System Services als Bestandteil von z/OS. Im Gegensatz zu z/OS Unix System Services wird es aber weniger häufig eingesetzt.

17.2 Web Archiv

17.2.1 Container

Containers sind Software Constructe innerhalb eines Web Application Servers. Sie dienen als eine Ausführungsumgebung. In einem Web Application Server existieren 2 Arten von Containern:

- **EJB Container, in denen EJBs laufen**
- **Servlet Container, in denen Servlets und EJBs ablaufen.**

Angenommen das Erstellen einer neuen Anwendung für einen Web Application Server:

Beide Container Arten stellen Services für die Software Elemente zur Verfügung, die in ihnen laufen. Beispiele für solche Service Funktionen sind z.B. Sicherheit und Namensdienste. Der Entwickler müsste derartige Funktionen für seine Anwendung selbst entwickeln, wenn der Container sie nicht zur Verfügung stellen würde.

Servlet Container werden häufig auch als Web Container bezeichnet. Sie enthalten statische Web Seiten (zusätzlich zu Servlets und JSPs), die spezifisch für diese Anwendung entwickelt wurden und damit Bestandteil der Anwendung sind.

EJBs, die eventuell teilweise bereits existieren, müssen für eine neue Anwendung und Installation in einem EJB Container konfiguriert werden. Dies geschieht mit Hilfe eines „Deployment Descriptors“.

17.2.2 Deployment Descriptor

Der Deployment Descriptor legt die Laufzeit Parameter einer EJB fest. Er wird in XML codiert und beim Aufruf der EJB ausgewertet. Parameter können statisch zur Assembly Zeit oder dynamisch zur Laufzeit festgelegt werden.

Der Deployment Descriptor wird typischerweise durch den EJB Entwickler angelegt, kann aber durch einen Administrator mit Hilfe eines Tools abgeändert werden.

Der Deployment Deskriptor ist eine Datei, die eine oder mehrere EJBs beschreibt, spezifisch deren Zusammenwirken und die Art, wie der EJB Container sie zur Laufzeit behandeln soll. Er enthält hauptsächlich deklarative Informationen, welche nicht im Bean – Code zu finden sind. Dies sind vor allem Informationen über die Struktur der Bean und ihrer Abhängigkeiten zu anderen Beans oder Ressourcen wie z. B. einer Datenbankverbindung.

Der Deployment Descriptor wird zusammen mit dem EJB Code in einer EJB *.jar File verpackt

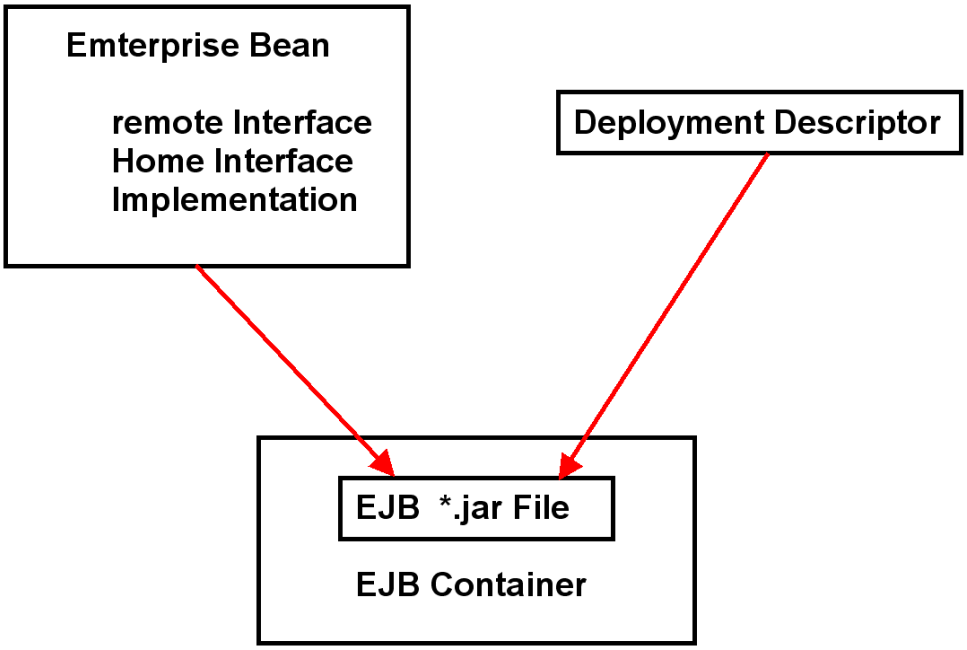


Abb. 17.2.1
Elemente einer *.jar File

Außerdem können im Deployment Deskriptor Umgebungsvariablen gesetzt werden, die von der Bean ausgelesen werden können und somit ihr Verhalten beeinflussen. Dies führt zu einer höheren Flexibilität, da dieselbe Bean in verschiedenen Umgebungen eingesetzt werden kann und nur der Deployment Deskriptor angepasst werden muss.

Bei der Installation einer CICS Anwendung (in einer beliebigen Sprache) benutzen wir das „Define“ Kommando zur Definition der Eigenschaften der neuen Anwendung. Für EJBs hat der Deployment Descriptor eine vergleichbare Funktion. Es besteht allerdings ein wesentlicher Unterschied: Die Parameter des Define Kommandos werden bei der Installation der CICS Anwendung fest eingebunden. Der Deployment Descriptor kann beim Aufruf einer EJB ausgewertet und abgeändert werden, was die Flexibilität (auf Kosten der Performance) erhöht.

17.2.3 Installation einer neuen Anwendung auf einem Server

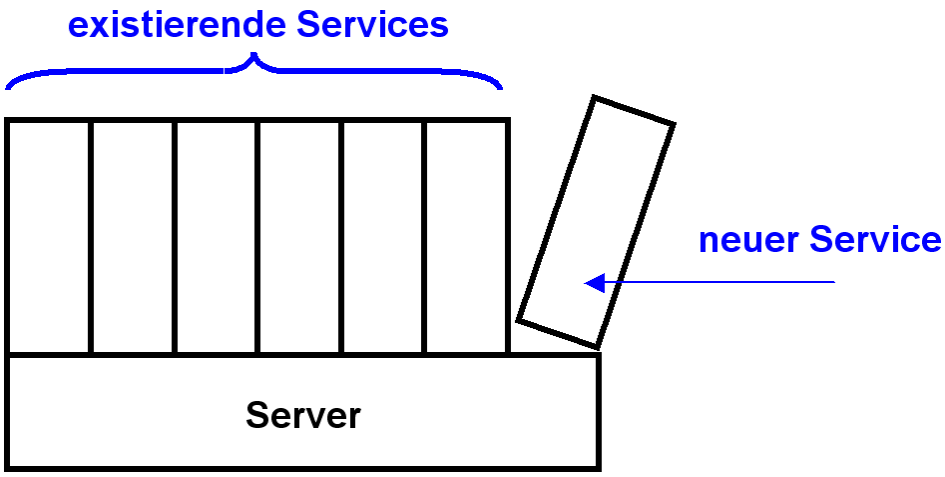


Abb. 17.2.2
Die Installation einer neuen Anwendung braucht mehrere Teilschritte

Wenn wir eine neue Anwendung entwickeln, und auf einem Server zum Laufen bringen wollen, gehen wir durch die folgenden Schritte:

1. **Wir entwickeln die Anwendung in irgend einer Programmiersprache auf irgendeiner Entwicklungsumgebung, z.B. TSO, JDK, Eclipse, andere ...**
2. **Wir testen die neue Anwendung aus (normalerweise auf einem getrennten Testsystem).**
3. **Wir installieren die neue Anwendung für die Benutzung durch zahlreiche Klienten auf einem Produktionssystem (einem Server), z.B. einem CICS Transaktionsserver oder dem EJB Container eines Web Application Servers.**

Für den letzten Schritt gehen wir durch die folgenden Teilschritte:

a. **Wir definieren dem Server gegenüber, dass es jetzt einen neuen Service (unsere neue Anwendung) gibt. Dies geschieht, damit beim Aufruf des neuen Services der Server weis, wo er den entsprechenden Code findet, unter welchen Umständen der Zugriff erfolgt, was der Server beim Aufruf beachten muss, welche Ressourcen er bereitstellen muss, und vieles mehr. Dies erfolgt bei einer CICS Anwendung mittels mehrerer „define“ Kommandos, bei einer EJB Anwendung mittels des „Deployment Descriptors“ .**

b. **Wir installieren den Code der neuen Anwendung auf dem Server. Im Zusammenhang mit dem WebSphere Application Server wird dieser Schritt als „Deployment“ bezeichnet. Bei einer CICS Anwendung benutzen wir das „install“ Kommando.**

17.2.4 Vergleich CICS – Enterprise Java Beans

	CICS	EJB
Definition	Define Command	Deployment Descriptor
Installation	Install Command	Deploy Command
Package	Group	JAR, WAR, EAR

Abb. 17.2.3
Dieser Vergleich soll beim Verständnis der Zusammenhänge helfen

Unter CICS wird ein neues Anwendungsprogramm, ein Mapset usw. mit Hilfe des Define Kommandos gegenüber dem CICS Transaktionsserver definiert. Bei einer EJB übernimmt der Deployment Descriptor die gleiche Funktion.

Für die Installation einer neuen Anwendung werden das Business Logik Programm, der Mapset und die TRID zu einer „Group“ zusammengefasst und das „Install“ Kommando ausgeführt. Bei einer JEE Anwendung werden die Komponenten zu einer Java Archive (*.jar) File zusammengefasst und mittels des „Deploy“ Kommandos installiert.

Die CICS Group entspricht hierbei grob dem Java Archive.

17.2.5 ejb – jar – Datei

Files mit einer .jar extension (JAR files), sind im Prinzip einfach eine Gruppe von Files

Eine ejb – jar – Datei ist das standardmäßige Verpackungsformat für Enterprise JavaBeans. Dabei handelt es sich um eine normale Java – ARchivdatei (JAR), die mit Hilfe des Hilfsprogramms jar erzeugt werden kann. Sie enthält aber spezielle Dateien, die alle jene Informationen zur Verfügung stellen, die ein EJB – Container zur Inbetriebnahme der in der JAR – Datei enthaltenen Beans benötigt.

In einer ejb – jar – Datei gibt es zwei Arten von Inhalten:

- **Die Klassendateien aller EJBs einschließlich ihrer Interfaces sowie der Bean – Implementationen (der Bean Code). Darüber hinaus können auch containergenerierte Klassen enthalten sein**
- **die Deployment Descriptor Dateien. Als Minimum muss eine standardmäßige ejb – jar.xml – Datei enthalten sein.**

Die kompilierten Java – Klassen, aus denen die EJBs bestehen, befinden sich in package-spezifischen Verzeichnissen innerhalb der ejb – jar – Datei, genau wie es bei normalen JAR – Dateien der Fall ist. Der Deployment Deskriptor, mit dem die EJBs beschrieben und konfiguriert werden, muss in der ejb – jar – Archivdatei unter META-INF/ejb-jar.xml zu finden sein.

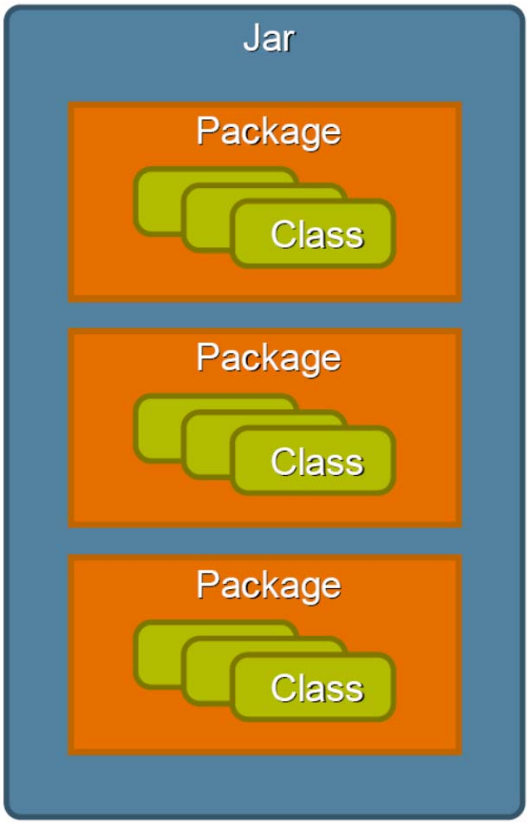


Abb. 17.2.4
Eine Anwendung kann aus mehreren Packages bestehen.

Enterprise Applications sind eine Kollektion von JAR Files. Sie

- **enthalten Packages**
- **enthalten Klassen**
- **encapsulate Daten und Logik**

Verschiedene, funktional zusammengehörende Klassen werden in Packages mit einem klaren Interface (`public` Klassen und Methoden) integriert.

Zur Laufzeit (runtime) ist eine JAR File nichts anderes als eine Sammlung von Klassen auf einem Classpath.

Die JAR Spezifikation definiert ein Class-Path Attribute, welches den Classpath extended.

17.2.6 Web Application

Eine "Web-Applikation" besteht aus einem oder mehreren Servlets, weiteren Java-Klassen, die als Hilfsklassen zur Unterstützung der Servlets dienen, statische Dateien wie HTML-Seiten und GIF/JPG-Bilder, sowie JSPs (Java Server Pages), die eine dynamische Ausgabe formatieren. All diese Elemente sind wichtig für die Ausführung. Zusammen bilden sie eine "Web-Application".

In diesem Zusammenhang wird ein Servlet Container häufig auch als „Web Container“ bezeichnet.

Der gesamte Web-Interaktion Prozess beginnt mit einer einzigen Seite im Browser. Der Benutzer klickt auf eine Schaltfläche oder einen Link auf der Seite. Dies verursacht eine Anfrage (Request) an die Web-Anwendung im Web-Container des Web-Application-Servers, zum Beispiel unter Verwendung von HTML-Forms. Die Anfrage wird von der Web-Anwendung verarbeitet. Je nach den Umständen kann (oder kann nicht) eine EJB oder eine andere Business Logik Komponente aufgerufen werden. Eine neue Seite wird zurück an den Browser gesendet. Diese enthält die Ergebnisse der Anfrage und präsentiert Schaltflächen oder Links für die nächste Anfrage. So besteht die Web-Anwendung aus einem Satz von Verarbeitungsschritten oder Interaktionen. Jede von ihnen erhält eine Anfrage, die von einer Seite erzeugt wird. Eine Antwort wird in Form von einer Seite erstellt, die als Eingabe für eine nachfolgende Interaktion dienen kann.

Es ist wichtig, ein "Web-Applikation" von einer Enterprise Java Bean (EJB) zu unterscheiden. Nach der JEE-Spezifikation sind dies verschiedene Dinge. Sie werden in verschiedenen Arten von Containern ausgeführt, beispielsweise innerhalb einem WebSphere für z/OS Anwendungsserver (WAS).

Servlets, die als Teil einer Web-Anwendung ausgeführt werden, können auf EJBs zugreifen. Sie verhalten sich wie ein "Ersatz" Anwendungs-Client für eine EJB. Das Servlet steuert die http-Sitzung mit dem Browser, das Format der Eingabeparameter und das Format der Ausgabe, die an den Browser des echten Klienten zurückgegeben wird. Die EJB konzentriert sich auf die Geschäftslogik der Anwendung und greift auf Unternehmensressourcen und Daten zu.

17.2.7 Statische HTML Seiten

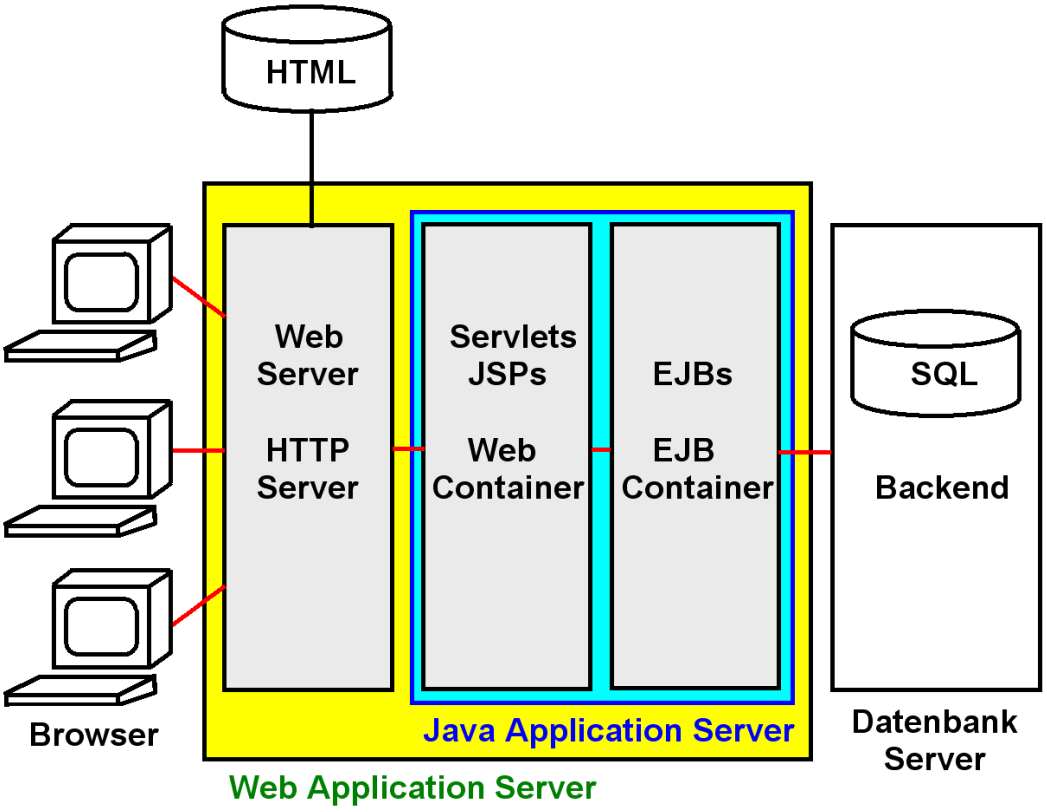


Abb. 17.2.5
Die wichtigsten Komponenten eines Web Application Servers

Eine neue Anwendung besteht in der Regel aus 2 Teilen, der Business Logik und der Präsentationslogik.

Die Businesslogik wird als eine Menge von EJBs erstellt, und in der Form eines *.jar Verzeichnisses für die Installation in dem Web Application Server bereit gestellt.

Wir brauchen etwas vergleichbares für die Präsentations-Logik.

In einem Unternehmen sind die meisten statischen Web Seiten nur im Zusammenhang mit einer spezifischen Anwendung relevant.

Die Präsentationslogik besteht typischerweise wiederum aus 2 Teilen:

a. **einer Reihe von Servlets und Java Server Pages, und**
b. **einer Reihe von statischen HTML Seiten.**

Beispiele solcher statischen HTML Seiten sind z.B. ein Welcome Screen oder ein Verabschiedungsscreen („Thank you for visiting our order status page“).

Es ist sinnvoll, diese statischen HTML Seiten, zusammen mit den Servlets und Java Server Pages der Präsentatinslogik in ein gemeinsames „Web Archive“, einer WAR File, zu verpacken.

17.2.8 Web Archive (WAR) und Enterprise Application Resource (EAR)

Eine WAR – Datei (Web ARchiv) dient standardmäßig dazu, Web – Anwendungen zu verpacken. In einer WAR – Datei gibt es drei Arten von Inhalten:

- **Die Klassendateien aller Servlets und JSPs, die Bestandteil der Webanwendung sind. Diese sind unter dem Verzeichnis WEB-INF/classes in einer package – spezifischen Verzeichnisstruktur zu finden.**

- **Die statischen HTML – Seiten, welche für das Ausführen der Webanwendung nötig sind. Es ist aber auch möglich, JSP – Seiten, als dynamische Generierung von Response – Seiten zu verwenden. Diese Dokumente befinden sich innerhalb der WAR – Datei in einer Verzeichnisstruktur. Zum Beispiel könnten alle Bilder im Verzeichnis *images* abgelegt sein, während die *index.html* im context – root des Archivs liegt.**

- **Die Deployment Deskriptor – Dateien. Der Deployment Deskriptor eines Web – Archives ist (wie auch schon der ejb – jar – Deskriptor), eine XML – Datei (siehe Abschnitt 20.1), welche eine Beschreibung der Anwendung, die Namen und die Parameter der Servlets zur Laufzeit, sowie sessionrelevante Einstellungen enthält. Als Minimum muss eine standardmäßige web.xml – Datei enthalten sein, welche im Verzeichnis WEB-INF zu finden ist. Wie bei den Java Archiven ist es auch hier möglich, das noch weitere Deskriptoren, je nach Art des Produktes und des Herstellers, in dem Web – Archiv Verwendung finden.**

Außerdem werden auch Sicherheitsaspekte, wie zum Beispiel der Loginmechanismus einer Anwendung definiert.

Der Servlet Container eines Web Archives wird analog häufig als Web Container bezeichnet. Um eine Verwechselung mit dem Web Server (z.B. Apache) zu vermeiden, bezeichnet man letzteren dann als http Server. Um Missverständnisse zu vermeiden, empfehlen wir dringend, den Begriff Web Server zu vermeiden, und statt dessen die Begriffe http Server und Web Container (oder Servlet Container) zu benutzen.

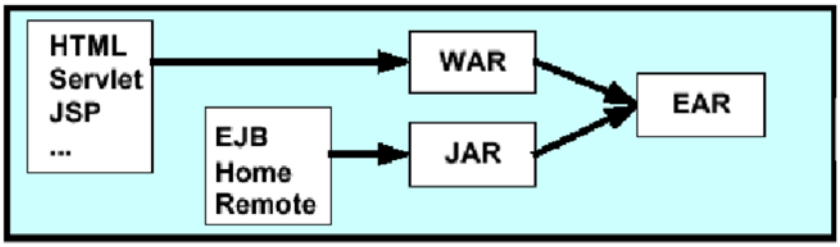


Abb. 17.2.6
Zusammenspiel von WAR, JAR und EAR

Eine J2EE Anwendung besteht aus Präsentationslogik und aus Business Logik, aus Web – und EJB – Komponenten. Beide Arten von Komponenten sind in Archiven verpackt, Verhalten und Eigenschaften sind in Deployment Deskriptoren (DD) definiert.

Alle Komponenten (Business Logik und Präsentationslogik) einer Anwendung sind in einem EAR (Enterprise Application Ressource) zusammengestellt. Eine vollständige Anwendung kann durch Installation eines EAR – Files auf einen anderen Web Application Server verteilt werden und sollte sich dort wie erforderlich konfigurieren lassen.

Hierbei wird die heute übliche Konfiguration unterstellt, bei der der http Server eine vom Web Application Server getrennte Einheit ist.

Die Web Application und die EJB Application werden häufig getrennt und von unterschiedlichen Entwicklern erstellt. Die Web Application wird in ein WAR (Web Archive) gepackt; die EJB Application in ein JAR (Java Archive). Beide werden in ein EAR (Enterprise Archive) kombiniert. Eine neue Anwendung wird in einen JEE Application Server (wie Weblogic oder WebSphere) geladen, indem man mittels dessen "Deployment Tools" die entsprechende Application EAR spezifiziert.

Anmerkung: Nach dem neuen EJB 3.1 Standard müssen EJBs nicht mehr in einer separaten EJB-JAR Datei deployed werden, sondern können direkt in der WAR Datei mit paketiert werden.

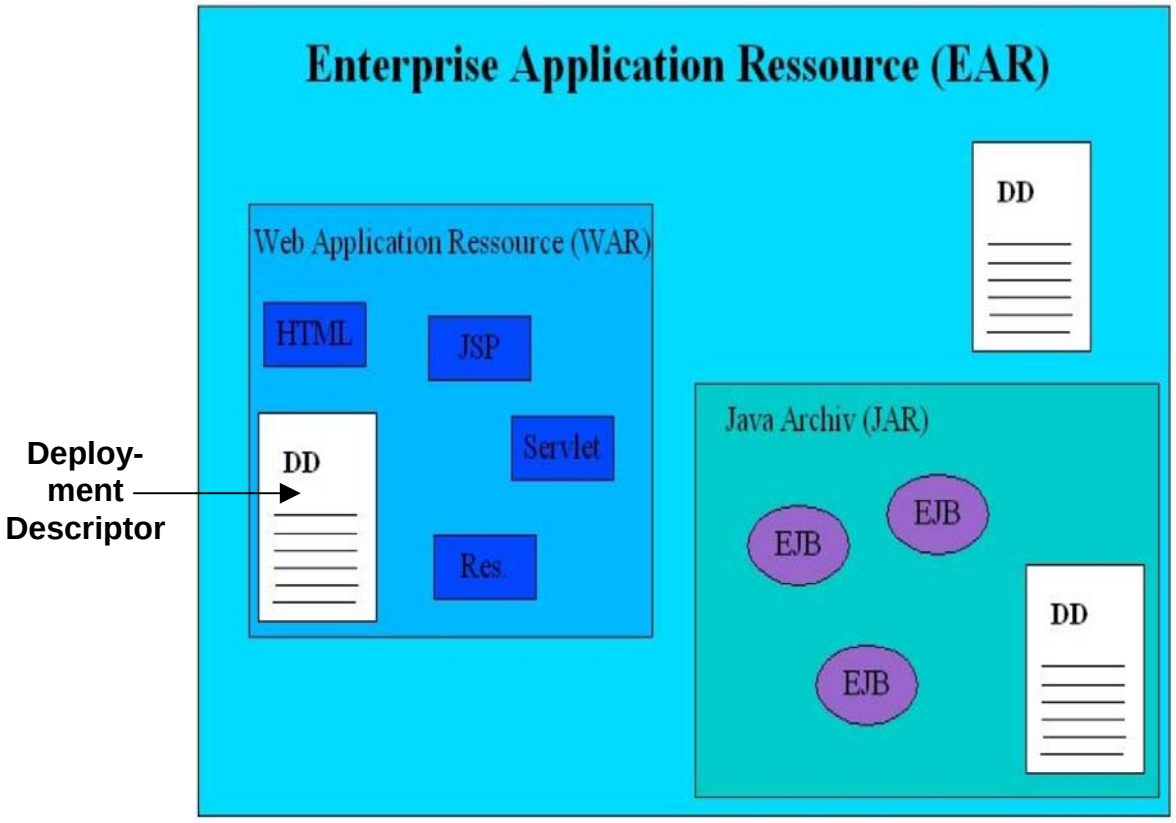


Abb. 17.2.7
Bestandteile der Enteprise Application Resource

Die EAR– Datei (application.xml) enthält nur den Namen der Enterprise – Anwendung und deren Bestandteile . Dies ist ein sehr primitives Beispiel einer EAR Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.3//EN"
  "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Hello World Web App</display-name>
  <module>
     <web>
       <web-uri>[b]hello.war[/b]</web-uri>
       <context-root>hello</context-root>
     </web>
  </module>
</application>
```

17.2.9 JEE Komponenten

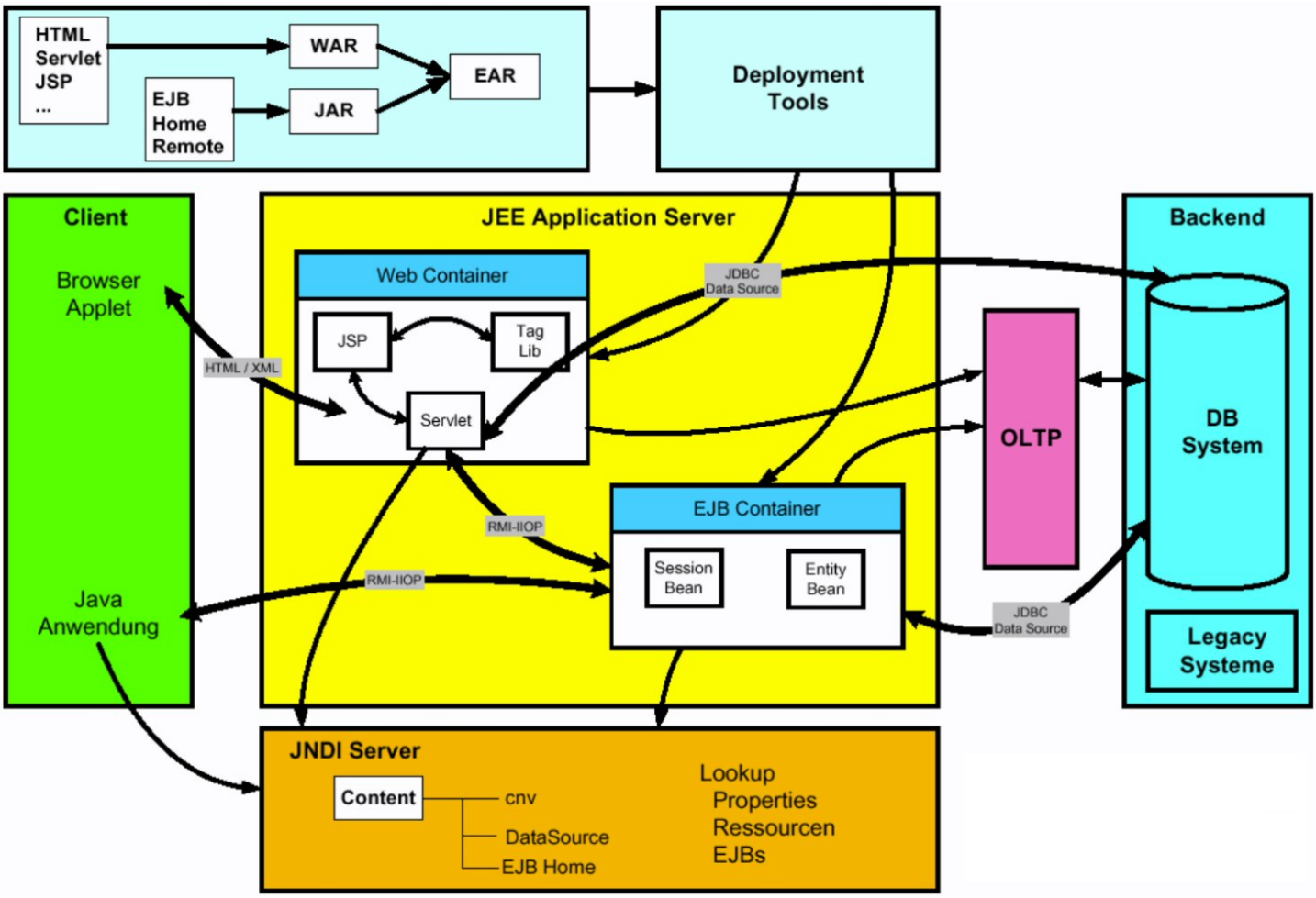


Abb. 17.2.8
Zusammenspiel Web Container, EJB Container und Backend

Abbildung 17.2.8 enthält einen Überblick über die Komponenten eines JEE Web Application Servers.

Die JEE Anwendung besteht aus 2 Komponenten: Business Logik und Presentation Logic.

Die Business Logic besteht aus einer EJB Anwendung, die in einem EJB Container läuft. Die EJBs greifen entweder auf eine Datenbank direkt zu (über JDBC oder SQLJ), oder sie benutzen einen zwischengeschalteten Transaction Monitor (OLTP, On-Line Transaction Processing System), z.B. CICS.

Für die Presentation Logic existieren 2 Alternativen. Ein thin Client Ansatz benutzt einen Browser, sowie entweder HTTP oder XML für den Aufruf der Server-zentrischen Präsentations-Logik, die in einem Servlet Container (Web Container) untergebracht ist. In diesem Fall bezeichnet man die Presentation Logic auch als “Web Application”. (Nicht gezeigt ist der http Server, der die http Requests von dem Browser entgegennimmt und an den Web Container weiterreicht).

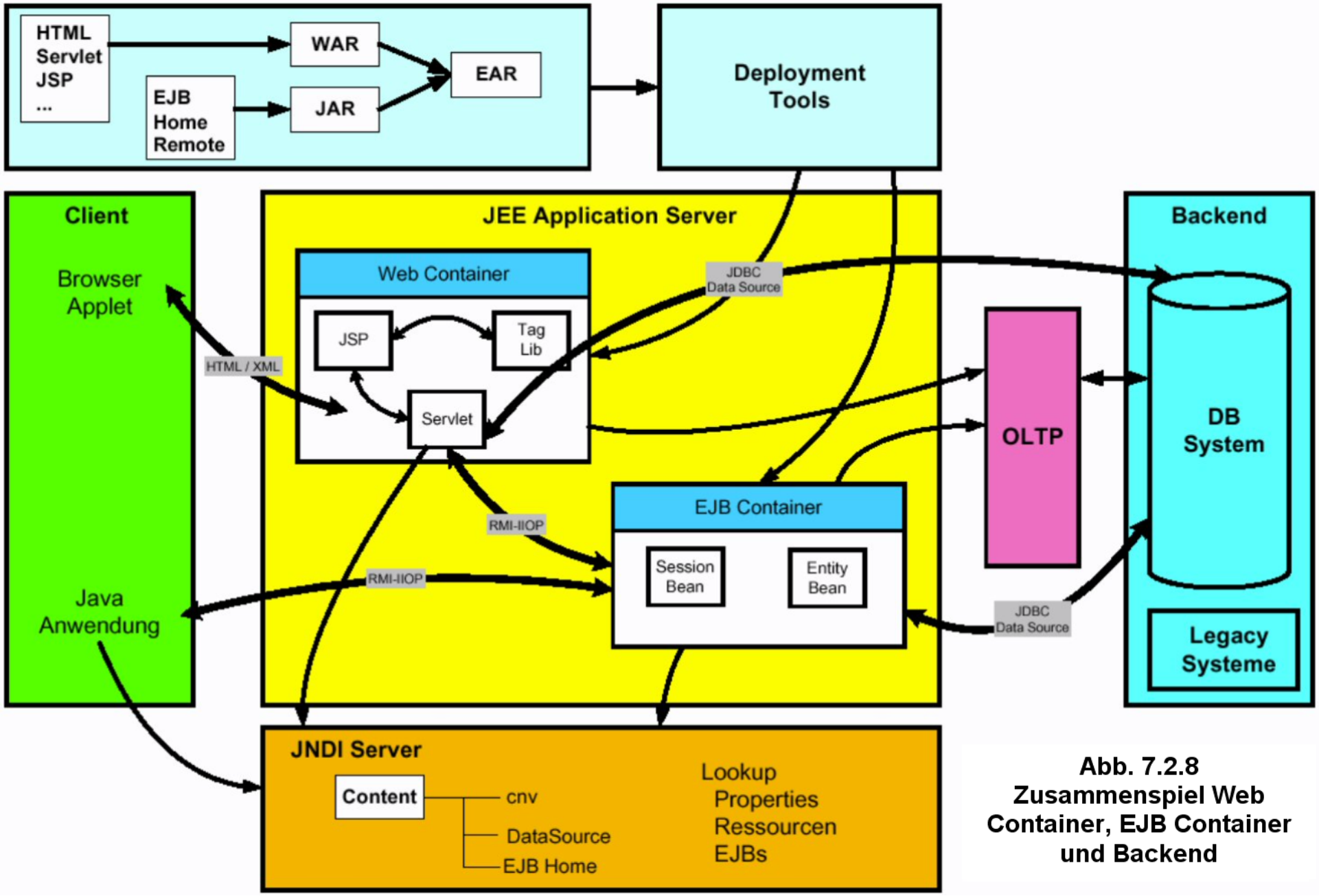


Abb. 7.2.8
Zusammenspiel Web Container, EJB Container und Backend

Alternativ wird die Presentation Logic in einem thick Client Ansatz implementiert. Hierbei greift die thick Client Logik auf die EJBs direkt mittels JRMP oder RMI/IIOP zu. z/OS verwendet hier ausschließlich RMI/IIOP.

Wenn der Web Container und der EJB Container als getrennte Prozesse in getrennten JVMs implementiert sind, kommunizieren sie über RMI/IIOP. Wenn sie in der gleichen JVM laufen, ist der RMI/IIOP Overhead nicht erforderlich.

Alle Java Komponenten benutzen JNDI (Java Naming and Directory Service) "to locate each other".

17.2.10 e-Business

E-Business-Anwendungen kombinieren Web Application Server, Web-Clients (z. B. Web-Browser) und Standard-Internet-Protokolle, um den Zugriff auf Daten und Anwendungen zwischen mehreren Unternehmen zu erleichtern. Es gibt viele Technologien, die bei der Entwicklung der Client-Seite einer Web-Anwendung in einer E-Business-Anwendung genutzt werden kann. Dazu gehören Java, TCP / IP, HTTP, HTTPS, HTML, DHTML, XML, MIME, SMTP, IIOP und X.509, unter anderem. Der Erfolg des Projekts und künftige Anpassungsfähigkeiten hängen davon ab, welche Technologien einbezogen werden.

IBM ermutigt Entwickler, die Client-Seite von Web-Anwendungen leicht-gewichtig zu machen. Dies wird üblicherweise als Thin-Client bezeichnet. Der Thin Client besteht aus einem Java-fähigen Web-Browser und eine Java Virtual Machine (JVM), eine TCP/IP-Stack, und (möglicherweise) einer Verschlüsselungs- (Encryption) Bibliothek. Mit dieser Funktion hat der Client drei wesentliche Vorteile für E-Business-Anwendungen:

- **Jeder Browser hat das Potential, eine Web-Anwendung auszuführen, die auf dem Thin Client-Modell basiert.**
- **Der Client-Teil der Web-Anwendung ist klein und schnell heruntergeladen.**
- **Der Server ist in der Lage, maßgeschneidertes HTML an den Klienten zurück zu geben, indem Client oder Benutzer Attributen ausgewertet werden.**

17.3 Load Balancing

17.3.1 Skalierbarkeit

Die Anzahl der Web Browser Zugriffe wachsen von Jahr zu Jahr. Hilfreich ist, dass die große Mehrzahl aller Zugriffe sich auf statische Web Seiten bezieht, die einen nur relativ geringen Verarbeitungsaufwand erfordern. Faustformel: Rund 80 % aller Zugriffe sind statisch.

Dennoch reicht ein einziger physischer Server häufig nicht aus, um die Anfragen mit einer vertretbaren Antwortzeit abzuarbeiten.

Erschwerend kommt hinzu, dass die Arbeitslast kurzfristig sehr stark schwanken kann. Wenn z.B. die Firma Otto Versand kurz vor den ARD Fernsehnachrichten um 20:00 eine Anzeige schaltet, geht die Anzahl der Zugriffe auf ihre Home Page sprunghaft um einen Faktor 10 oder 100 in die Höhe. Bricht der Server zusammen, war das Geld für die Fernsehwerbung umsonst ausgegeben.

Von Jahr zu Jahr müssen die Unternehmen sich auf ein starkes Kapazitäts-Wachstum einstellen. Zur Abhilfe ist es üblich, mehrere physische Server parallel zu schalten, und bei Wachstumsbedarf um zusätzliche Server zu erweitern.

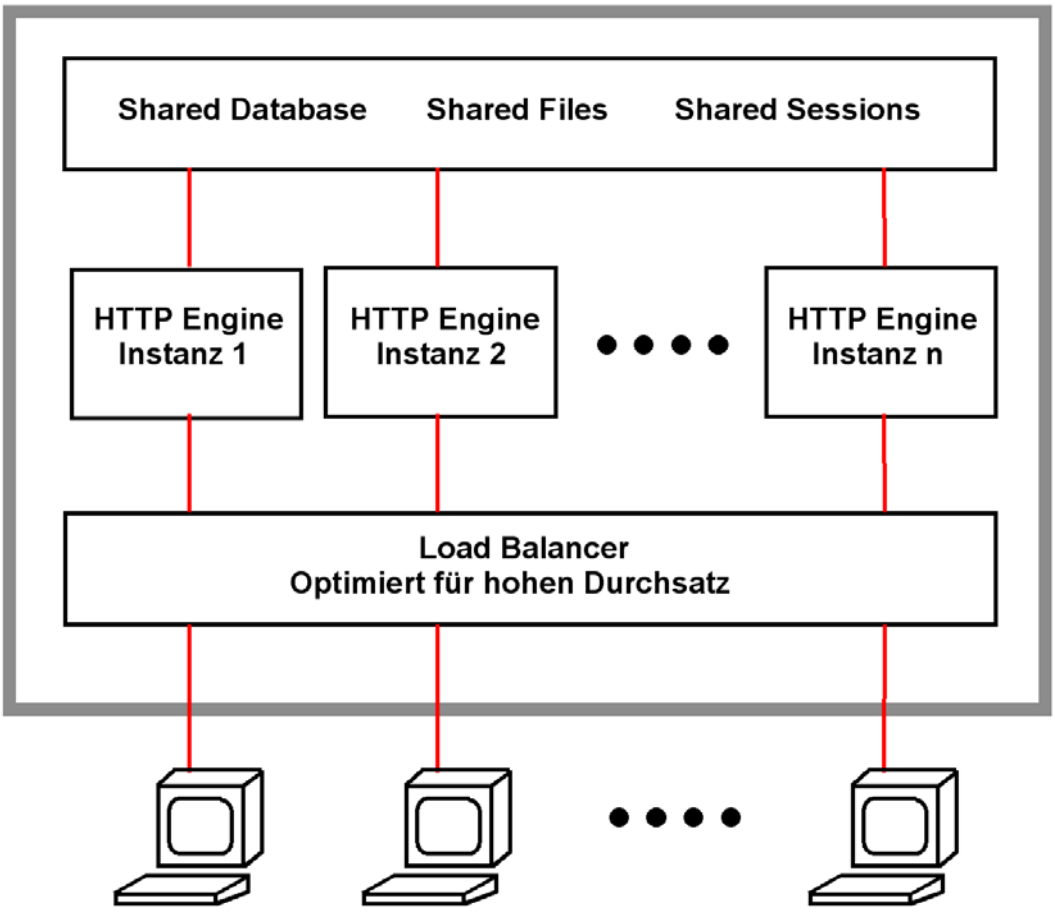


Abb. 17.3.1
Server Farm

Der http Server behandelt Anforderungen für (meistens) statische Ressourcen: HTML Seiten, GIF Dateien, sowie für Servlet oder CGI Aufrufe.

Hohes Verkehrsaufkommen, kurzlebige Anforderungen. Vielfach reicht die Verarbeitungskapazität eines einzelnen Servers nicht aus. Die Skalierung wird durch mehrfache parallele http Server Engines erreicht.

Der „Load Balancer“ verteilt die Anforderungen auf die einzelnen Web Engines.

Meistens wird angenommen, dass die einzelnen Anforderungen keine großen Unterschiede bezüglich der erforderlichen Resourcen haben.

Ein extremes Beispiel ist www.google.com

17.3.2 www.google.com

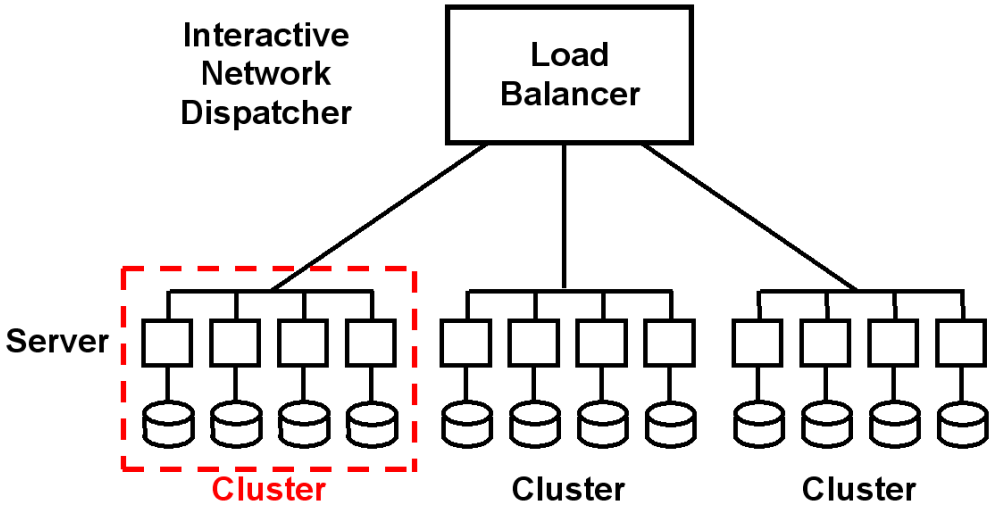


Abb. 17.3.2
Google Cluster

Google unterhält 2009 mehr als 30 Rechenzentren mit mehreren 10 000 Servern (Blades) in jedem Rechenzentrum. In einem Rechenzentrum stehen zahlreiche Cluster. Jeder Cluster dupliziert den ganzen Google Datenbestand. Das Update der Datenbestände in den einzelnen Clustern erfolgt nicht notwendigerweise synchron; es ist deshalb möglich, dass man bei zwei gleichzeitigen Anfragen unterschiedliche Antworten erhält.

Der „Load Balancer“ ist eine zentrale Komponente, welcher einkommende Anfragen auf die einzelnen Rechenzentren und von dort auf die einzelnen Cluster verteilt. Jeder Cluster ist in der Lage, jede Art von Anfrage zu bearbeiten.

Ein einfacher Workload Algorithmus, z.B. Round Robin, verteilt Anfragen der Reihe nach auf die einzelnen Cluster und Server.

Die Google Datenbank ist zwischen den Rechenzentren und zwischen den Clustern dupliziert. Die Kopien sind nicht notwendigerweise auf dem gleichen Änderungsstand. Die allermeisten Abfragen sind read-only.

Abb. 17.3.3
Weltweite Standorte der Google Rechenzentren

Google hat (April 2008) 19 Data Center Standorte in den USA, 12 in Europa, 1 in Russland, 1 in South America, und drei in Ostasien.

Abb. 17.3.4
Close-up of a Google Server Rack

Die Server benutzen kostengünstige Standard PC Komponenten.
http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/

17.3.3 Google Infrastruktur

Die Google Server Infrastruktur besteht aus mehreren Server Typen, von denen jeder eine andere Aufgabe hat:

- **Google Load Balancer nehmen die Client Request (Anfrage) entgegen, und leiten sie an einen freien Google Web Server mittels eines Squid (Open Source) Proxy Servers weiter.**
- **Squid proxy Server erhalten die Client Request und geben das Ergebnis zurück, wenn es sich in einem lokalen Cache befindet. Andernfalls wird sie an einen Google Web Server weitergegeben. Squid ist ein Proxy Server und Web Cache Daemon. Es verbessert die Performance indem wiederholte Requests in dem Cache gehalten werden.**
- **Google Web Server koordinieren die Ausführung der Client Queries und formatieren das Ergebnis in eine HTML Seite. Zur Ausführung gehen die Anfragen an Index Server. Diese berechnen den Rank, holen Vorschlägen von den Spelling Servern ein, und besorgen eine Liste von Advertisements von den Ad Servern.**
- **Data-gathering Servers durchsuchen (spider) ständig das Web. Google's Web Crawler wird als GoogleBot bezeichnet; Sie updaten die Index und Dokument Database Server und benutzen Google's Algorithmen um den Rank zu berechnen.**
- **Jeder Index Server enthält einen Set von Indexes. Sie erstellen eine Liste von Dokument IDs ("docid"), dergestalt, dass Dokumente mit einer spezifischen docid das Query Wort enthalten.**
- **Dokument Server speichern Dokumente. Jedes Dokument ist auf Dutzenden von Dokumenten Servern gespeichert. Bei einer Search gibt der Dokumentenserver ein Summary des Dokuments auf der Basis der Query Worte zurück. Dokument Server können auch das vollständige Dokument laden wenn gewünscht.**
- **Ad Servers manage Anzeigen (Advertisements).**
- **Spelling Servers erstellen Vorschläge für das Spelling von Anfragen.**

Was ist der Unterschied zwischen der Google Infrastruktur und einem Web Application Server?

Die Google Infrastruktur verwendet Standard x86 und PC Bauteile, Linux als Betriebssystem, sowie proprietäre Software. Kennzeichnend sind die verteilten (distributed) Data Base Characteristics mit (fast ausschließlich) read-only Queries, sowie (fast) keinen Anforderungen an Datenintegrität.

Amazon und eBay verwenden einen ähnlichen Ansatz für ihre Query Front-Ends (Beispiel: Durchsuchen aller Romane vom Autor xyz). Für das Backend, (ein Buch kaufen oder ein Gebot abgeben), ist ein anderer Ansatz erforderlich. Für die Backend Verarbeitung verwendet Amazon ein großes Unix System, eBay ein z/OS System. Backend Verarbeitung erfordert transaktionale Integrität für alle verarbeiteten Daten. Bei Google existiert (fast) keine äquivalente Backend Verarbeitung und auch kaum Bedarf an Datenintegrität.

Anders als bei Goggle müssen die betriebswirtschaftlichen Installationen großer Unternehmen oder Organisationen eine sehr große Anzahl von stark unterschiedlichen Anwendungen unterstützen, mit unterschiedlichen runtime Anforderungen und sehr unterschiedlichen Ausführungszeiten. Transaktionale Integrität sowie Synchronisation bei verteilten Datenbanken sind Schlüsseleigenschaften.

JEE Web Application Server wie WebLogic, Netweaver und WebSphere, sowie Transaction Server wie CICS und IMS/DC, MS Transaction Server und Tuxedo erfüllen diese Anforderungen.

17.3.4 WebSphere Produkt Familie

Der Begriff WebSphere ist doppelt belegt.

Offiziell bezeichnet WebSphere eine Produktlinie der Firma IBM, die unterschiedliche Software für Anwendungsintegration, Infrastruktur (z. B. Transaktionen und Warteschlangen) und eine integrierte Entwicklungsumgebung umfasst.

Zu den WebSphere-Produkten gehört unter anderem:

- **WebSphere Application Server (WAS)**
- **WebSphere Rational Application Developer und dessen RDz Erweiterungen für z/OS**
- **WebSphere Process Server**
- **WebSphere Integration Developer**
- **WebSphere MQ (andere Bezeichnung für MQSeries)**
- **WebSphere Portal Server**

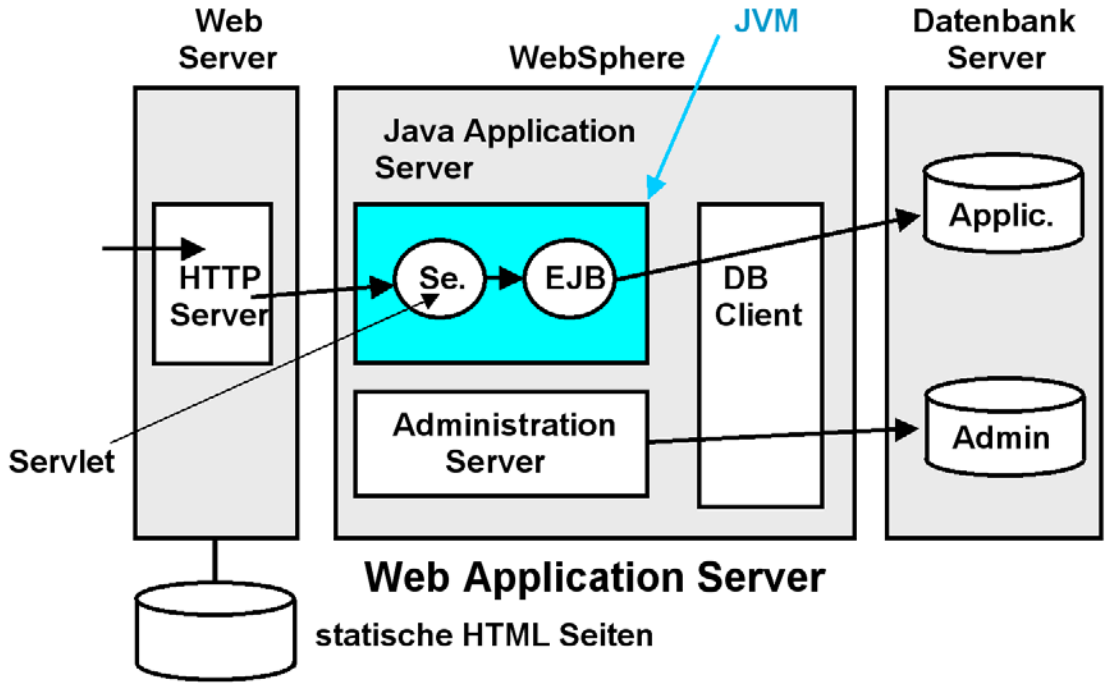


Abb. 17.3.5
WebSphere WAS Grundstruktur

Der WebSphere Application Server (WAS) stellt eine Plattform für das Deployment und die Ausführung (execution) von Java Anwendungen und Web Services zur Verfügung, besonders Servlets, JSPs und EJBs. WAS ist eine der führenden Implementierungen des JEE Standards. In den meisten Fällen wird WAS gemeint, wenn man von WebSphere spricht.

WebSphere Rational Application Developer und dessen RDz Erweiterungen für z/OS sind die integrierte Entwicklungsumgebung (IDE) für WAS.

Für den WebSphere Application Server ist eine Erweiterung unter dem Namen WebSphere Process Server verfügbar. Dieser enthält Unterstützung für Java Business Prozesse und IBM JEE Programming Extensions. Eine Vorgänger Version hatte den Namen WebSphere Business Integration Server Foundation.

WebSphere Integration Developer ist die Entwicklungsumgebung für den WebSphere Process Server

WebSphere MQ ist die neuere Bezeichnung für MQSeries und ist lose in die WebSphere Familie integriert. WebSphere MQ unterstützt Java Message Driven Beans.

Der WebSphere Portal Server ist ein Web-basiertes User-Interface, das flexibel angepasst und personalisiert werden kann. Es ist dafür gedacht, ein einheitliches Front End mit flexibler Anbindung von verschiedensten Backend-Systemen schnell herzustellen. IBM hat maßgeblich die beiden im Portal-Umfeld relevanten Standards beeinflusst: Den Java Portlet Standard, JSR 286, sowie den WebServices for remote Portlets Standard, WSRP 2.0.

Von allen WebSphere Produkten existieren mehrere Implementierungen mit jeweils unterschiedlichem Funktionsumfang.

17.3.5 WebSphere Administration

Ein WebSphere Web Application Server (WAS) enthält neben einem Servlet Container (andere Bezeichnung Web Container) und einem EJB Container noch weitere Komponenten.

Bei einem CICS Server erfolgt die Administration durch eine eingebaute Kommando Shell, die durch die CEDA Transaktion (und weitere Transaktionen) implementiert wird, Bei WebSphere fasst man alle administrativen Funktionen in einem eigenen Server, dem WAS Administration Server zusammen. Dieser kommuniziert über eine grafische Oberfläche mit dem Benutzer.

Weiterhin gehört zur WAS Grundausstattung ein DB Client für Zugriffe auf eine DB2 Datenbank.

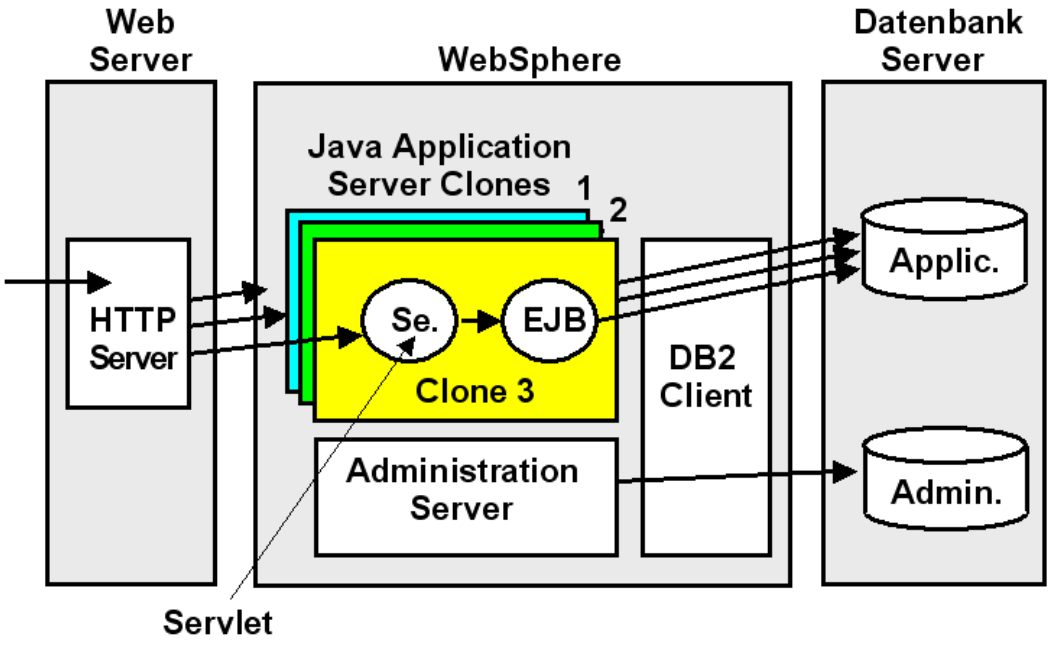


Abb. 17.3.6
Zur besseren Ausnutzung existieren mehrfache Clones der Java Application Server

Aus Performance-Gründen muss der Application Server multiprogrammiert arbeiten. Im Prinzip ist das mit Hilfe von Threads innerhalb der gleichen JVM möglich. Hierbei gibt es jedoch Grenzen. Deshalb sind mehrfache Clones des Application Servers vorhanden, die Java Anwendungen parallel verarbeiten können. Die Clones des Java Application Servers verfügen jeder über eine eigenen Java Virtuellen Maschine.

Auf mehreren Clones kann die gleiche Anwendung laufen. Es ist auch möglich, dass auf den Clones unterschiedliche Anwendungen laufen, oder dass eine bestimmte Anwendung erst bei Bedarf geladen wird.

17.3.6 Distributed Server

Middleware Produkte wie WebSphere, aber auch CICS und DB2 sind grundsätzlich in zwei verschiedenen Versionen verfügbar:

- **Die z/OS Version läuft unter dem z/OS Betriebssystem**
- **Die „Distributed“ Version läuft auf einem getrennten Linux, Unix oder Windows Server, der über ein geeignetes Protokoll (TCP/IP, Corba, IIOP, http, andere) mit dem zentralen z/OS Server verbunden ist. Auch Betriebssysteme wie Solaris und HP-UX sind denkbar.**

Die Distributed Version ist in der Regel bezüglich Hardware- und Software Lizenzkosten günstiger, verzichtet aber auf z/OS Funktionen wie Reliability, Availability, Sicherheit, LPARs, Coupling Facility, WLM, usw. Es ist nicht unüblich, dass in einer Organisation manche WAS Anwendungen auf distributed Servern, andere unter z/OS ausgeführt werden.

Für CICS und DB2 handelt es sich bei der distributed Version und der z/OS Version um jeweils zwei unterschiedliche Software Produkte. Wegen der ausgezeichneten Kompatibilität ist der Unterschied für den Benutzer kaum bemerkbar.

Für den WebSphere Application Server existiert nur eine einzige Quellcode Version für alle Plattformen. Durch eine Kompilierung mit unterschiedlichen Compiler Optionen werden die Versionen für die unterschiedlichen Plattformen erzeugt. Dabei werden bei der distributed Version viele Funktionen disabled, die unter z/OS verfügbar sind.

Ein Beispiel ist die Benutzung des z/OS Work Load Managers an Stelle des eingebauten einfachen Work Load Managers. Sysplex und Coupling Facility Unterstützung sind weitere Beispiele. Insgesamt verhält sich die z/OS Version in Bezug auf Funktionsumfang und Leistungsverhalten deutlich anders als die distributed Version. Lediglich der eigentliche Java Quell Code ist bei beiden Versionen identisch.

Der z/OS WebSphere Application Server (WAS) ist als Unix Anwendung konzipiert. Die z/OS Version läuft deshalb unter Unix System Services.

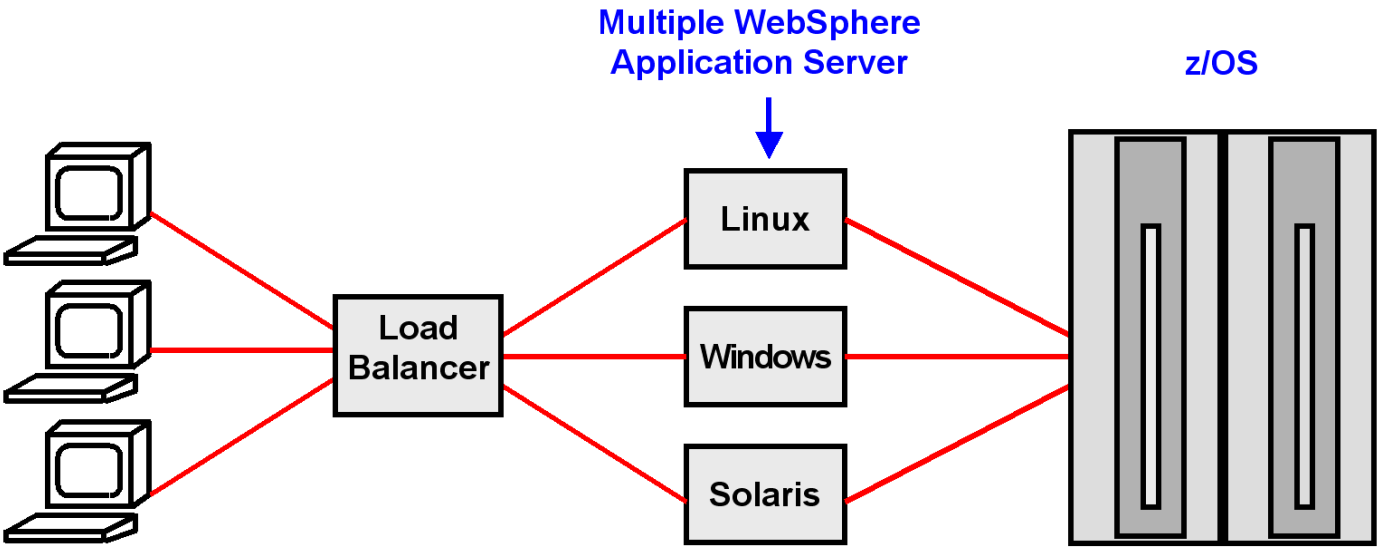


Abb. 17.3.7
Distributed WAS auf unterschiedlichen Platformen

Beim Einsatz der distributed WebSphere Application Server Version läuft auf einem physischen Server häufig nur eine einzige Anwendung, möglicherweise parallel auf mehreren Clones. Für mehrere Anwendungen sind dann mehrere physische Server vorhanden, z.B. 30 Server für 30 unterschiedliche Anwendungen.

Der http Server braucht eine Einrichtung, um eintreffende Nachrichten auf die richtigen WebSphere Server und Clones zu routen. Dies ist die Funktion Load Balancers. In einfachen Fällen wird ein simpler Round Robin Algorithmus benutzt.

Load-Balancing Internet Servers. IBM Redbook, SG24-4993-00, Dezember 1997, http://www.redbooks.ibm.com/redbooks/pdfs/sg244993.pdf

17.3.7 Structured File Server (SFS)

Ein Structured File Server ist ein File Server, der u.A. mit der distributed Version von CICS oder WAS eingesetzt wird. Er hat gegenüber einem normalen File Server diese zusätzlichen Eigenschaften:

- **Record-orientierte Dateien. SFS unterstützt Record orientierte Dateitypen. SFS organisiert die Datensätze in Feldern und bietet sowohl Record-Level als auch Feld-Level Zugriff auf Daten. Der Zugang auf Records erfolgt durch Indizes, so dass eine Anwendung schnell und einfach Zugriff auf Records hat.**
- **Transaction Protection. SFS bietet transaktionalen Zugriff auf Daten, die in einer Datei gespeichert sind. Dateien, die von SFS verwaltet werden, sind somit vollständig wiederherstellbar im Falle von Server Problemen, Netzwerkausfällen und Medien Ausfällen. SFS protokolliert automatisch alle Änderungen von Daten, die in SFS-Dateien gespeichert sind. SFS stellt sicher, dass alle Änderungen, die während der Verarbeitung einer Transaktion anhängig sind, entweder abgeschlossen oder vollständig rückgängig gemacht werden.**

SFS ist eine Art distributed Ersatz für VSAM.

17.3.8 WebSphere Application Server for z/OS

Die z/OS Version von WAS benutzt an Stelle der in Abb. 17.3.7 dargestellten distributed WAS Version die „Controller-Servant“ Konfiguration. Der Controller übernimmt die Aufgabe des Load Balancers für eine Gruppe von Servants. Letztere stellen die eigentlichen Application Server dar. Controller und Servant besitzen jeder eine eigene JVM.

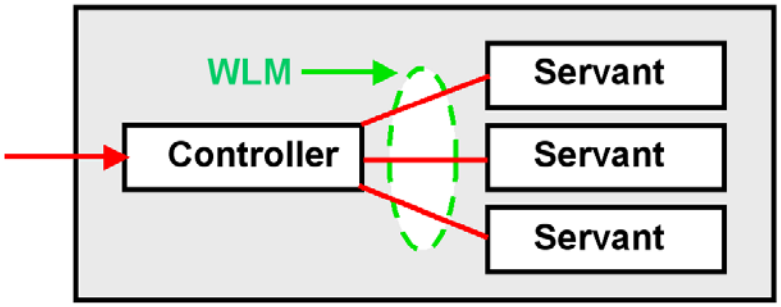


Abb. 17.3.8
z/OS Version des WebSphere Application Server (WAS)

Controller und Servant laufen in unterschiedlichen virtuellen Adressenräumen; der Controller läuft mit Speicherschutzschlüssel 2 und im Kernel Status. Die Servants laufen mit Speicherschutzschlüssel 8 im User Status.

Vielfach läuft eine einzige (oder nur wenige) Anwendung(en) in einem Servant.

Der Controller benutzt den z/OS Work Load Manager (WLM, Kapitel 13) um eingehende Anfragen an die Servants zu verteilen.

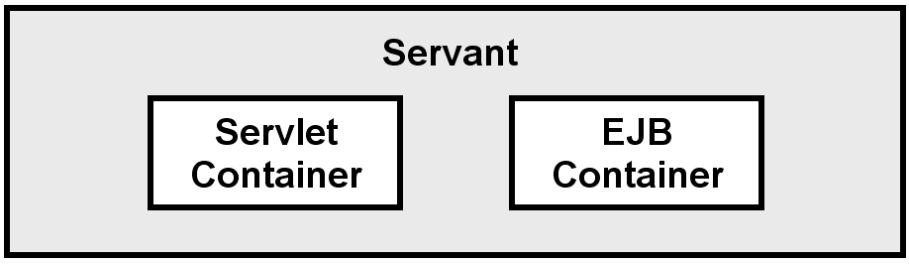


Abb. 17.3.9
WebSphere Runtime Structure on z/OS

Dies sind die Merkmale

- **Die grundlegende Ausführungsumgebung für den WebSphere Application Server (WAS) unter z/OS ist ein Server.**
- **Ein Server ist ein Controller / Servant Konfiguration**
- **Controller und Servant laufen in getrennten Regions (virtuelle Adressenräume)**
- **Für jede Controller-Region existiert eine oder mehrere Servant Regionen**
- **Der Controller ist der Kommunikationsprotokoll Einstiegspunkt (empfängt Nachrichten)**
- **Unterschiedliche Nachricht Kommunikationsprotokolle werden von dem Controller unterstützt, besonders HTTP(S) und IIOP**
- **Die JEE-Komponenten (besonders EJBs) werden in der Servant Region ausgeführt, mit einer WLM Queue zwischen Controller und Servant**

Darüber hinaus:

- **Es können mehrere WAS auf einem einzigen z/OS-System existieren, jeder mit seinem eigenen Controller und mehreren Servants.**
- **Auf einem z/OS System, spezifisch einem Sysplex, laufen häufig mehrere WebSphere Web Application Server.**

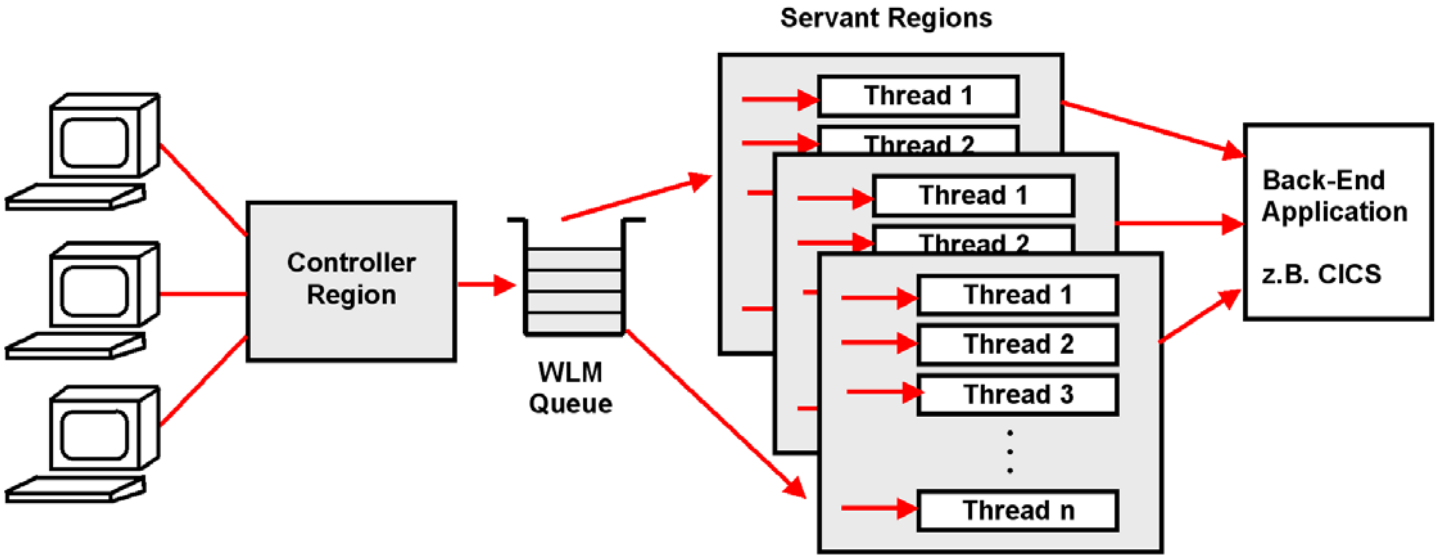


Abb. 17.3.10
z/OS Work Load Manager Steuerung der Input Queue

Zur Verbesserung des Leistungsverhaltens laufen mehrere Servant Prozesse auf dem Web Applikations-Server. In jeder Servant Region existiert (normalerweise) eine einzige JVM. In der Servant JVM können multiple Java Threads individuelle Anwendungen parallel verarbeiten.

Mögliche Backend Applications sind z.B. CICS, SAP/R3, DB2 oder IMS.

In der WebSphere Application Server für z/OS erfolgt die Zuordnung (classification) jeder Transaktion durch eine Controller Region. Der Controller ist ein getrennter Process, der in einem eigenen Address Space läuft. Die Controller Region agiert als ein Queuing Manager. Sie verwendet den z/OS Work Load Manager, und schedules eintreffende Work Requests zur Ausführung in multiple Server Address Spaces, die als Servants bezeichnet werden.

17.3.9 z/OS and Work Load Manager

Aufgaben der Control Region und ihrer Queues sind:

- **Lastverteilung**
- **Schutz (Isolation) der Anwendungen gegeneinander**
- **Availability und Reliability 24 Stunden/Tag, 7 Tage/Woche. Verabschiedet sich ein Prozess, läuft der Rest weiter**
- **Austesten neuer Anwendungen**

Unter z/OS optimiert der Work Load Manager die Lastverteilung. Die Distributed WebSphere Application Server Version (Abb. 17.3.7) hat statt dessen eine eigene primitiven Work Load Manager Funktion als Teil ihres Load Balancers. Anforderungen von dem Controller gehen (je nach Policy) zu einer von mehreren Queues. Jede Queue wird von mehreren Java Prozessen bedient.

Der Administrator legt die Anzahl und die Policies jeder Queue fest. Die Queue Policy bestimmt

- **URLs, die von der Queue bedient werden**
- **Anzahl der Prozesse für diese Queue**
- **Sicherheitsumgebung**

Ein z/OS WebSphere Application Server (WAS) Servant beherbergt Servlets, EJBs und andere Java Klassen. Die vom Work Load Manager verwalteten Queues verteilen eintreffende Nachrichten auf die einzelnen Servants. Jeder Servant läuft in seiner eigen z/OS Region.

Um die Arbeit des Systems auszubalanzieren, verwendet WebSphere die Dienste des z/OS Workload Managers (WLM). Drei verschiedene WLM Dienstleistungen werden von WebSphere eingesetzt:

1. **Routing**
   Die WLM Routing Service wird benutzt, um Klienten mit einem bestimmten Servant zu verbinden, unter Berücksichtigung der derzeitigen Systemauslastung und des WLM Performance Indexes.

2. **Queuing und Adressraum Management**
   WLM manages Servant Adressenräume um Performance Goals und deren WLM Performance Index (PI) für die derzeitige Work Load zu optimieren.

3. **Prioritisierung um Performance-Ziele zu erreichen**
   WebSphere delegiert die Verantwortung für das Starten und Stoppen von Servant Regionen an den WLM Address Space Management Service.

17.4 Leistungsverhalten

17.4.1 Arten von EJB Klienten

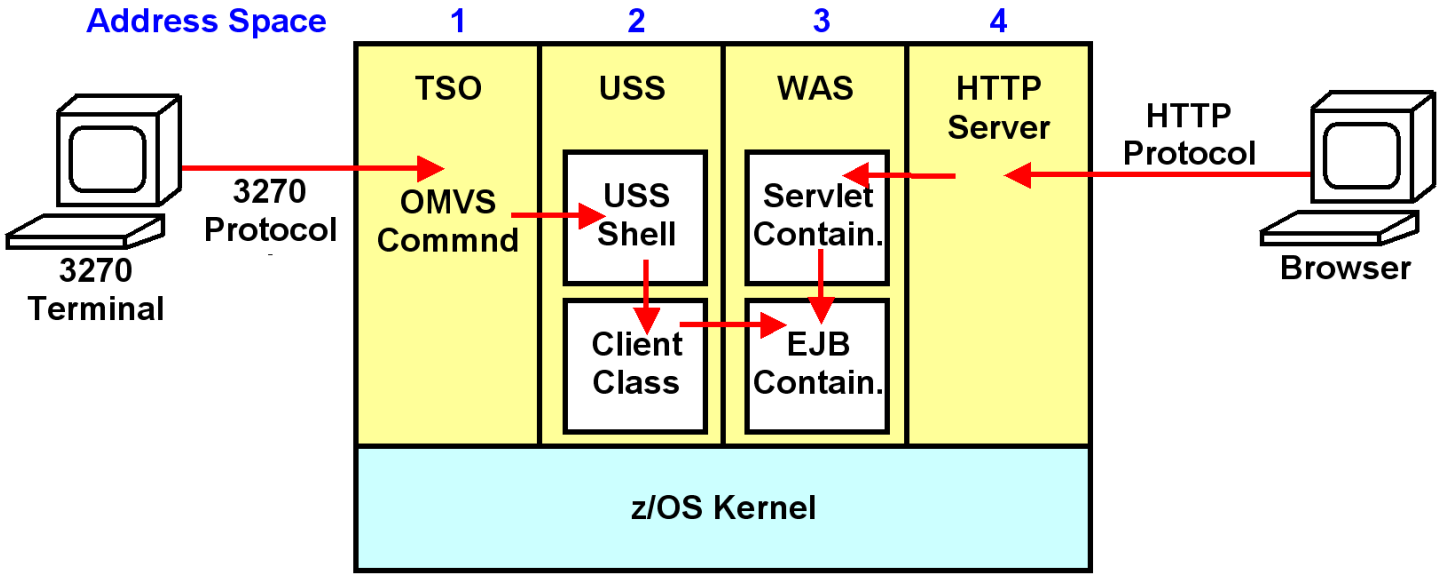


Abb. 17.4.1
Zugriff auf EJBs

Der Zugriff auf eine EJB (JEE Application) erfordert Code, der einen Klienten für diese EJB darstellt. Eine Möglichkeit besteht darin, mittels eines 3270 Terminals und TSO über das OMVS Command eine USS Session aufzurufen (links). In dieser Session wird eine Java Client Class mittels eines Unix System Services Shell Kommandos aufgerufen. Der Klient ruft über eine RMI/IIOP Verbindung eine EJB auf, die unter dem WebSphere Application Server läuft.

Der Vorteil dieses Vorgehens besteht darin, dass der Klient relativ leicht zu installieren ist, ohne dass man sich über Netzwerkprobleme Gedanken machen muss. Für das Austesten einer neuen Anwendung mag das günstig sein.

Viel üblicher ist es jedoch, mittels eines Browsers auf den z/OS http Server und ein Servlet, und über diesen eine EJB aufzurufen (rechts.

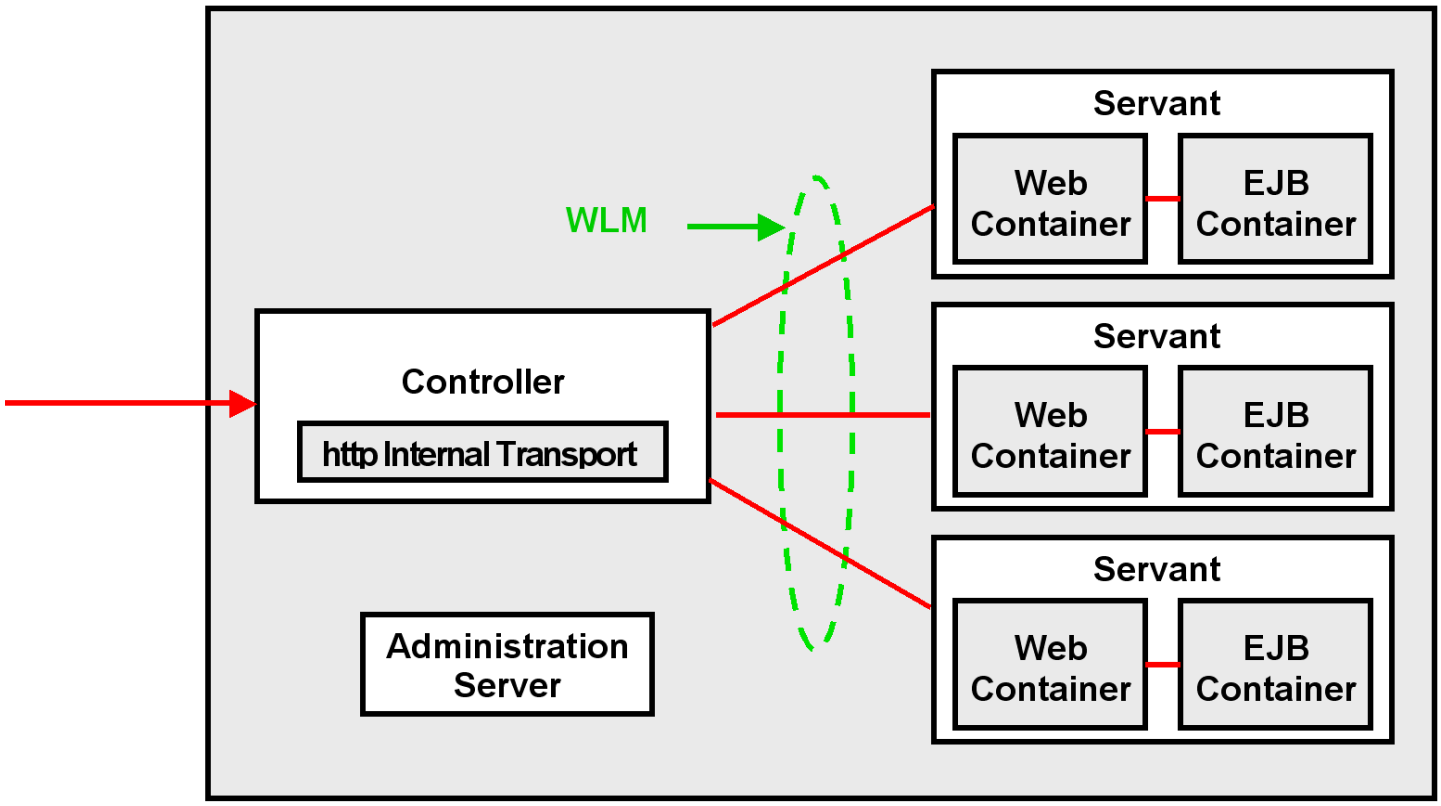


Abb. 17.4.2
WAS Controller mit der http Internal Transport Komponente

Abb. 17.4.2 zeigt nochmals die z/OS Version des WebSphere Web Application Servers. Der Controller übernimmt die Aufgabe des Load Balancers für eine Gruppe von Servants. Letztere stellen die eigentlichen Web Application Server dar. Controller und Servants laufen in getrennten Adressenräumen und besitzen jeder eine eigene JVM. Für Zugriffe von außen verfügt der WAS Controller über eine zusätzliche Komponente, den http Internal Transport.

Der Google Load Balancer hat sehr viel weniger Funktionalität als der WAS Controller Load Balancer .

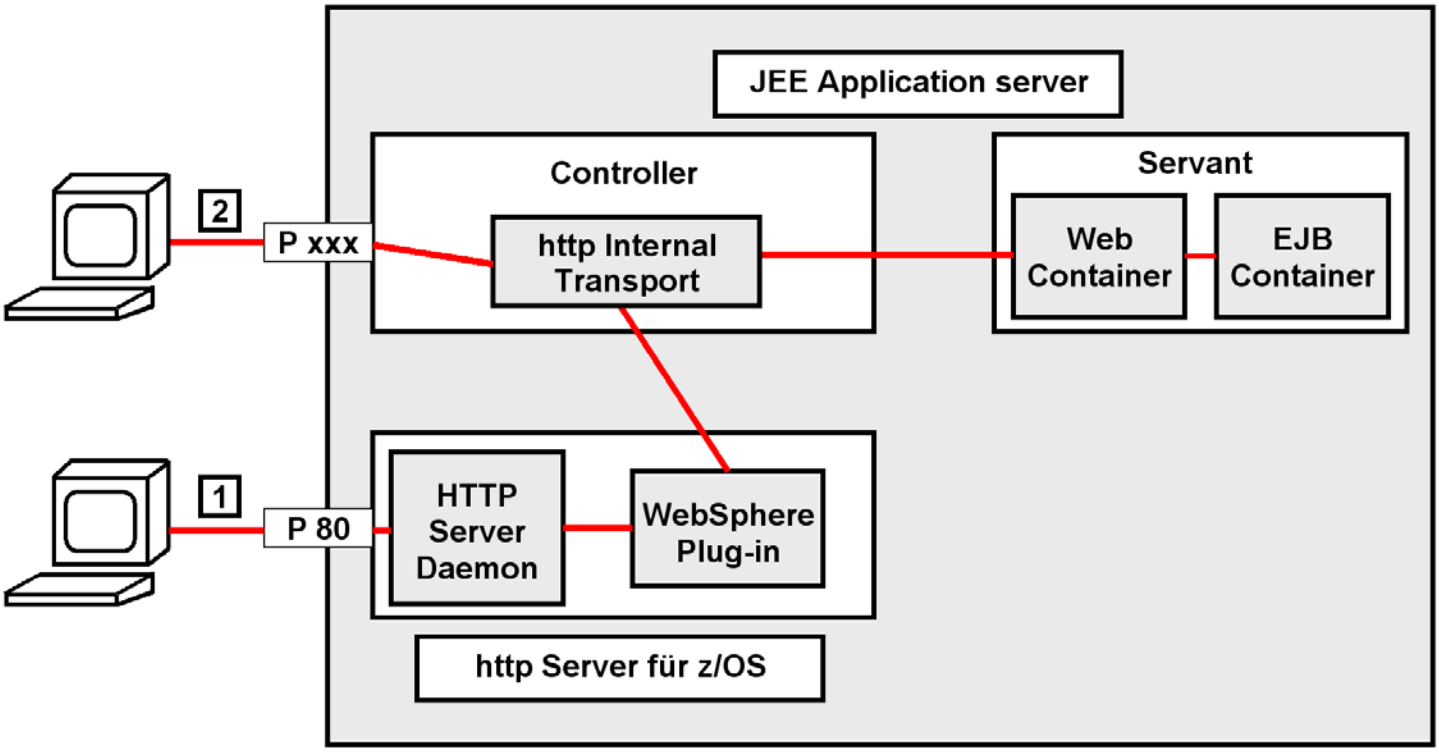


Abb. 17.4.3
http Internal Transport

Angenommen, ein Benutzer möchte mittels seines Browsers eine WebSphere Anwendung (spezifisch ein Servlet) aufrufen, das unter z/OS läuft. WebSphere für z/OS unterstützt zwei verschiedene Arten von Komponenten für den Empfang von Nachrichten:

1. **Die erste Komponente ist der IBM http-Server, der auf Apache-Basis arbeitet, den Port 80 als Standard verwendet und http Nachrichten empfängt und sendet.. Er benutzt das WebSphere Plug-in (Abschnitt 15.4.2), um die http Nachricht an den WAS Controller weiterzureichen. Der http Server aktiviert die WebSphere http Plug-In-Komponente in dem http-Server Adressraum. Das Plug-in kann konfiguriert werden, http-Requests vom http-Server zu filtern, und Nachrichten an einen von einem von mehren möglichen JEE-Anwendungsservern weiterzureichen.**

2. **In einer z/OS Controller-Servant Konfiguration enthält der Controller eine alternative Komponente, den „http Internal Transport“. Dieser ist in der Lage, auf einem beliebigen Port Nachrichten direkt entgegenzunehmen, und unmittelbar an ein spezifisches Servlet in einen Servant weiterzuleiten.**

   Der http Internal Transport läuft in dem "Controller" (CR) Adressraum eines WebSphere für z/OS Application Servers. Er hört ausgewählten Ports auf dem TCP/IP-Stack auf eintreffende Nachrichten ab. Im Prinzip können beliebige Protokolle eingesetzt werden. Wenn eine Nachricht empfangen wird, prüft der Internal Transport, dass die Nachricht in diesem Server ausführbar ist. Vorausgesetzt die Nachricht ist gültig, leitet der Internal Transport die Anforderung zur Ausführung an den Web-Container weiter. Der Web Container und der EJB-Container laufen in "Servant" (SR) Adressräumen.

18. 17.4.2 WebSphere Application Server Overview

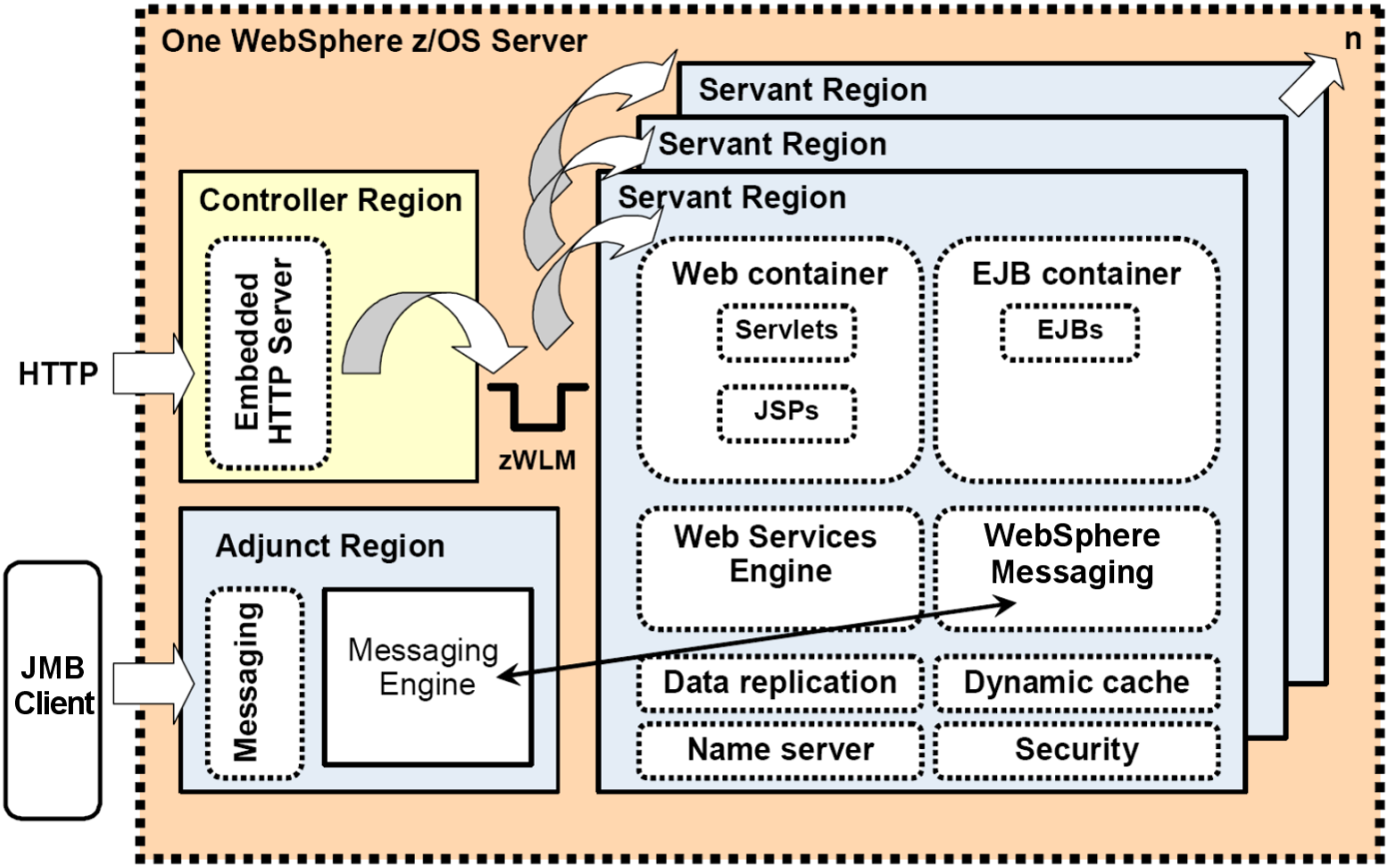


Abb. 17.4.4
Komponenten des WebSphere Application Servers

Abb. 17.4.4 zeigt Details eines z/OS WebSphere Servers.

Der Controller Region und die einzelnen Servants laufen in getrennten Adressenräumen (Regions). Die Controller Region enthält einen eigenen http Server, der http Requests direkt entgegen nehmen kann, ggf. über definierte Port Nummern, um sie an die http Internal Transport Komponente der Controller Region weiterzureichen. Daneben kann natürlich auch ein externer http Server eingesetzt werden. Neben dem IBM http Server kommen dafür besonders der Microsoft Internet Information Services (IIS), Lotus Domino Enterprise Server und Oracle iPlanet Web Server in Frage.

Weiterhin existiert eine getrennte Region (Adjunct Region) für die Verarbeitung von Java Message Driven Beans (Abschnitt 15.4.5). MDBs werden durch einen MDB Klienten aufgerufen, der als ein beliebiges Java Programm (POJO, Plain Old Java Program) implementiert werden kann. MDBs selbst laufen in dem EJB Container einer Servant Region. Jede Servant Region enthält eine Messaging Komponente, an welche die Messaging Engine der Adjunct Region MDB Client Requests weiterreichen kann.

Die Servant Region implementiert die eigentliche WAS Funktionalität. Neben dem Servlet (Web-) Container und EJB Container enthält sie weitere Funktionen für Data Replication, Security, Dynamic Caching und Naming Service.

Zusätzliche Funktionen existieren für Web Services (siehe Abschnitt 20.2).

17.4.3 WebSphere Skalierbarkeit unter z/OS

Die z/OS Version des Websphere Application Servers (WAS) hat gegenüber der distribited Version eine Reihe von Vorteilen. Diese resultieren u.A. aus der Nutzung von z/OS spezifischen Einrichtungen wie dem Sysplex, der Coupling Facility und den z/OS Work Load Manager.

Eine Folge ist eine unerreichte Skalierbarkeit von Java Transaktionen.

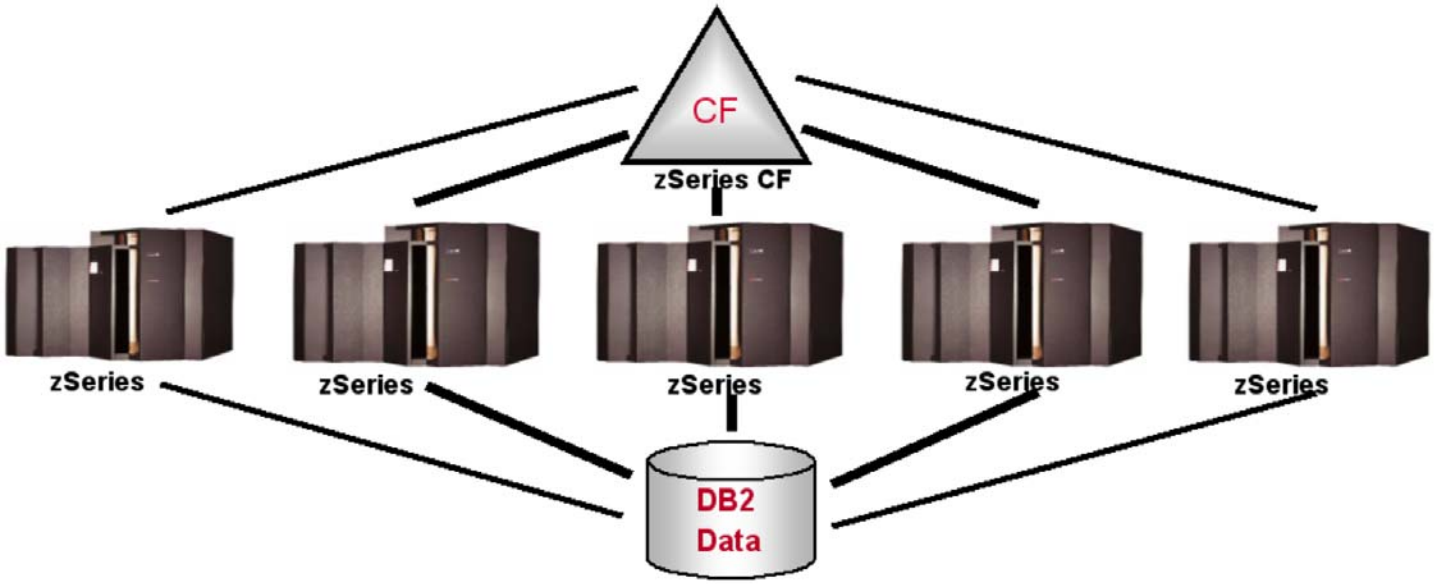


Abb. 17.4.5
Konfiguration für des Skalierbarkeit Test

Skalierbarkeit ist ein Problem mit den meisten Unix, Linux and Windows Implementierungen. Die folgenden Abbildungen geben die Ergebnisse eines WebSphere Skalierbarkeit Testes unter z/OS wieder.

Der Test wurde 2006 mit fünf z990 Systememen, mit je 16 CPUs durchgeführt. Eine weitere z990 mit 4 CPUs diente als Coupling Facility

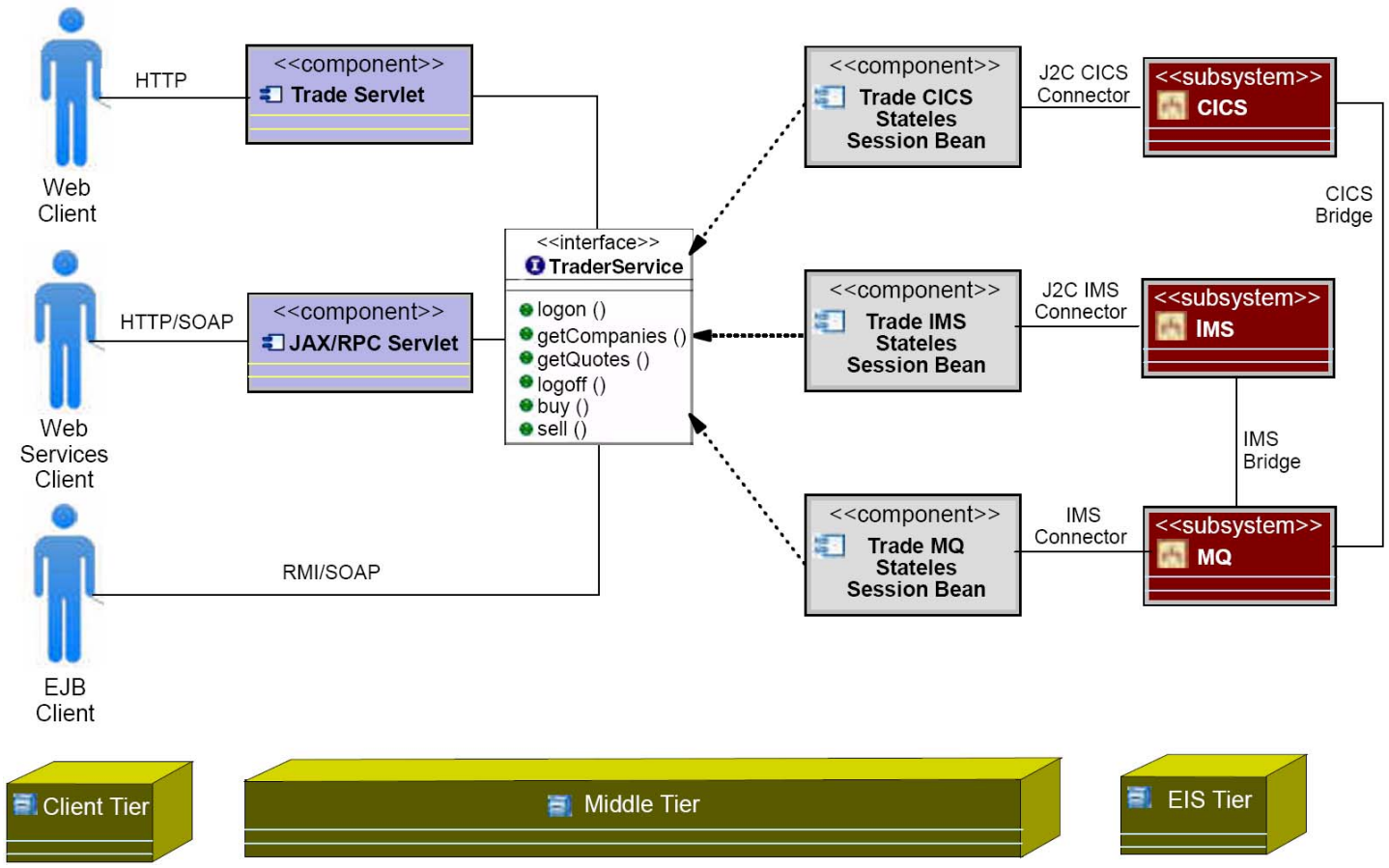


Abb. 17.4.6
Trader Benchmark

z/OS benutzt für viele Tests ein Standard Client/Server Benchmark, die “Trader Application“. Dieses Benchmark wurde über die Jahre immer wieder verbessert. Abb. 17.4.6 zeigt die Topologie des Trader Benchmarks.

Service name	Description
logon(userId, password)	Log on a user using userId and password.
getCompanies()	Get the list of companies.
getQuotes(userId, companyId)	Get the quotes of the company(companyId)’s stock as well as the user(userId)’s holding of that company.
sell(userId, companyId)	Sell the user(userId)’s holding of the company(companyId)’s stock.
buy(userId, companyId, quantity)	Buy the quantity of the company(companyId)’s stocks for the user(userId).
logoff(userId)	Log off the user(userId).

Abb. 17.4.7
Services des Trader Benchmarks

Abb. 17.4.7 zeigt die Services des Trader Benchmarks. Es benutzt eine 3-Tier Architektur, sowie JEE Komponenten wie Servlets, JavaServer Pages, EJB Session Beans und Web Services.

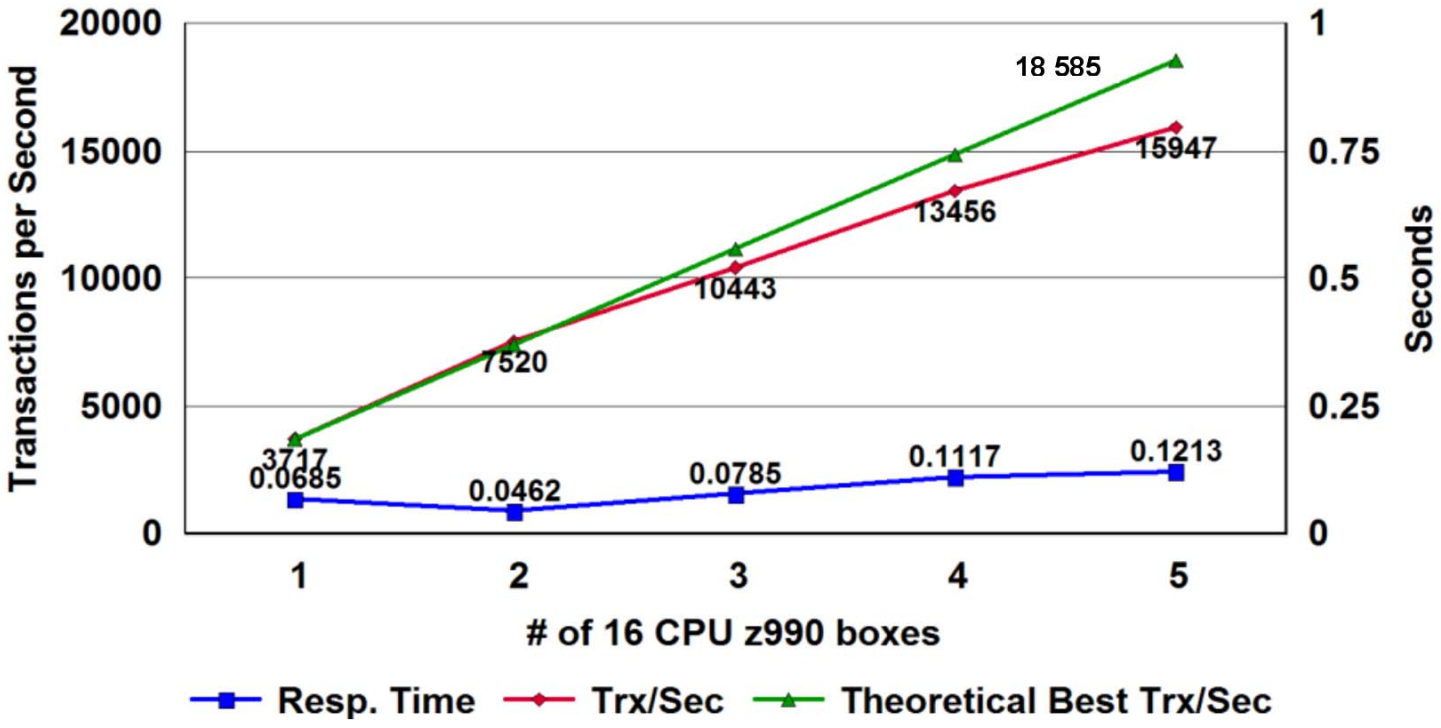


Abb. 17.4.8
Ergebnisse des Skalierbarkeit Tests

Dies sind die Ergebnisse:

Ein einzelnes System mit 16 CPUs ist in der Lage, 3 717 Trader Transaktionen/s durchzuführen. Das theoretische Maximum für 5 Systeme mit 80 CPUs ist 5 x 3717 Transaktionen/s = 18 585 Transaktionen/s. Der tatsächlich gemessene Durchsatz ist 15 947 Transaktionen/s, oder 15 947 / 18 585 = 86 % des theoretischen Maximums.

ftp://ftp.software.ibm.com/software/zseries/pdf/WASforzOSv6PerformanceReport.pdf, oder http://www.cedix.de/VorlesMirror/Band2/Perform01.pdf.

17.5 Weiterführende Information

Literatur zum Thema Unix System Services:

ABCs of z/OS System Programming Volume 9, November 2005, IBM Form No. SG24-6989-01

z/OS UNIX System Services User ’s Guide, IBM Form No. SA22-7801-04

OS/390 Unix System Services. IBM Form No. SC28-1891-8

Faszinierende Details über ein Google Data Center sind zu finden unter:

http://www.youtube.com/watch?v=zRwPSFpLX8I

http://www.golem.de/0904/66376.html

http://www.google.com/intl/de/about/datacenters/gallery/#/tech

Siehe dazu auch

http://www.golem.de/1109/86376.html

19. **Ein Tutorial “Developing and Testing a "Hello World" JEE Application” ist verfügbar unter**

http://www.ibm.com/developerworks/websphere/techjournal/0306_wosnick/wosnick.html

Ein Video zum Thema Integration von WebSphere und CICS finden Sie unter

http://www.youtube.com/watch?v=Yz6g4oSphZE&feature=related