Enterprise Computing

Band 2
z/OS Internet Integration

Prof. Dr.-Ing. Wilhelm G. Spruth
Prof. Dr. Martin Bogdan

spruth@informatik.uni-leipzig.de
bogdan@informatik.uni-leipzig.de

September 2013.

Die Vorderseite des Buches zeigt einen zEC12 Rechner mit angeschlossener zEnterprise Blade Center Extension (zBX). Die Darstellung entstand aus 2 Abbildungen in dem IBM Redbook : IBM zEnterprise EC12 Technical Guide, August 2012, SG24-8049-00.

Vorwort

Band 2 ist die Fortsetzung von Band 1 und deckt die Themen ab, die im Sommersemester behandelt werden. Dies sind:

- **Sysplex**
- **Virtualisierung und logische Partitionen**
- **Work Load Manager**
- **zBX, PureData System for Analytics**
- **Java Enterprise Edition**
- **RMI**
- **WebSphere Application Server**
- **Java Connection Architecture**
- **Transaktionsverarbeitung mit Java**
- **SOA**

Ein Verzeichnis der Akronyme, dal Literaturverzeichnis und ein Index (Stichwortverzeichnis) befinden sich für beide Bände am Ende von Band 2.

Inhaltsverzeichnis

Seite

11. Sysplex

11-1 Mehrfachrechner

11.1.1 Taxonomie von Mehrfachrechnern

Eine einzelne CPU hat eine nur begrenzte Rechenleistung. Die Forderung nach mehr Rechenleistung führte dazu, mehrere Prozessoren zu koppeln.

Wir unterscheiden zwischen Parallelrechnern und Mehrfachrechnern. Bei Parallelrechnern wird ein einziger Prozess auf vielen CPUs gleichzeitig ausgeführt. Beispiele für solche Prozesse sind Wetter- und Klimamodelle, der Automobil Crash Test, Kosmologische Simulationen oder Seismische Exploration.

Bei Mehrfachrechnern laufen zahlreiche Prozesse oder Threads gleichzeitig auf eben so vielen CPUs. In Systemen mit mehr als einer CPU teilt das Betriebssystem die einzelnen Prozesse oder Threads auf die verschiedenen Prozessoren auf und sorgt damit für eine höhere Leistungsfähigkeit.

Mainframe Systeme erfordern meistens die Leistung von mehr als einer CPU, arbeiten aber fast immer als Mehrfachrechner. Ausnahmen sind z.B. Data Mining Anwendungen, siehe Abschnitt 14.4.

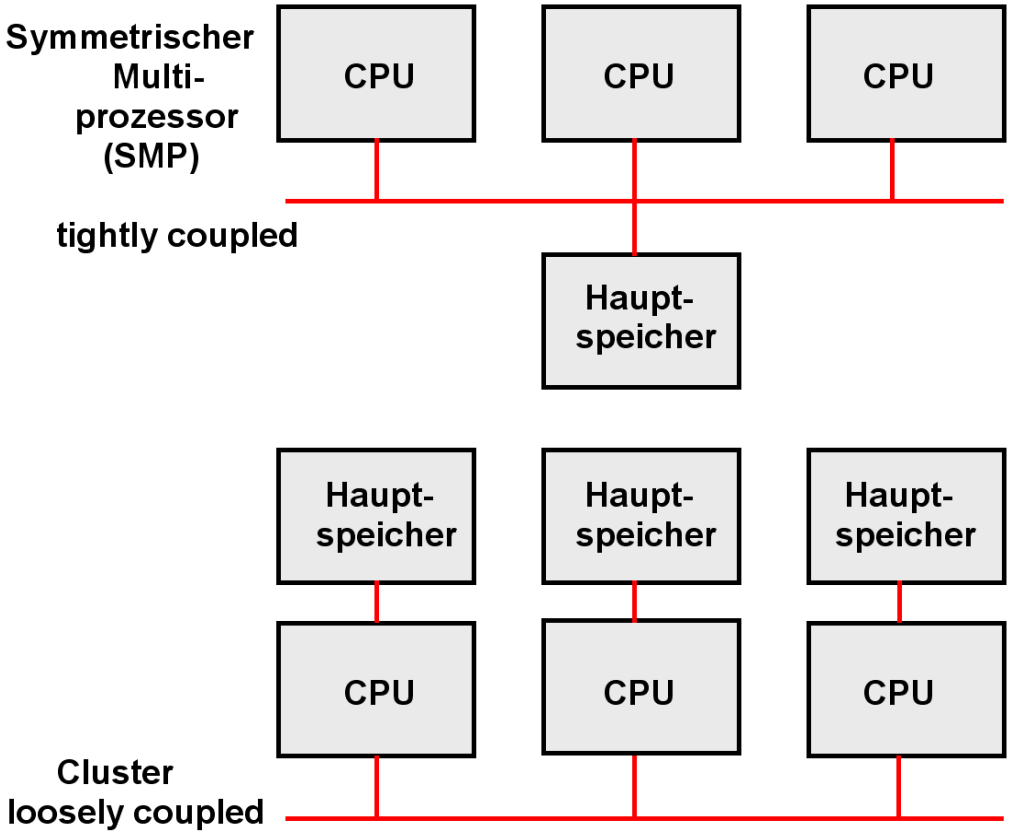


Abb. 11.1.1
SMP und Cluster

Bei Mehrfachrechnern unterscheiden wir 2 Grundtypen:

- **Ein Symmetrischer Multiprozessor (SMP) wird auch als eng gekoppelter (tightly coupled) Mehrfachrechner bezeichnet. Er besteht aus mehreren CPUs (über 100 bei einem zEC12 Mainframe), die alle auf einen gemeinsamen Hauptspeicher zugreifen. In dem Hauptspeicher befindet sich eine einzige Instanz des Betriebssystems.**
- **Ein Cluster (loosely coupled Mehrfachrechner) besteht aus mehreren Rechnern, von denen jeder seinen eigenen Hauptspeicher und seine eigene Instanz eines Betriebssystems hat**

11.1.2 Crossbar Switch

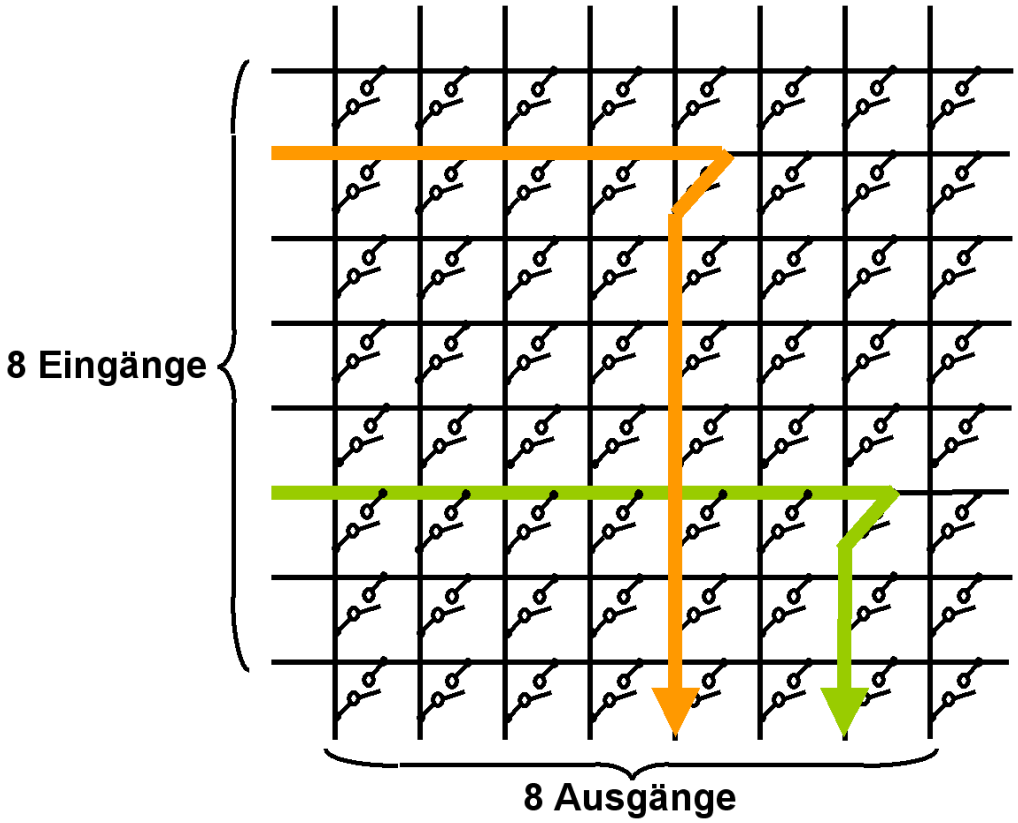


Abb. 11.1.2
Ein Crossbar Switch kann mehrere Nachrichten gleichzeitig übertragen

Die CPUs eines Mehrfachrechners sind über ein Verbindungsnetzwerk miteinander verbunden. Dies kann ein leistungsfähiger Bus sein, z.B. der PCI Bus. Ein Bus hat aber nur eine begrenzte Datenrate. Deshalb setzen viele Implementierungen statt dessen einen Kreuzschienenverteiler (Crossbar Switch, Crossbar Matrix Switch) als Verbindungsnetzwerk ein, der die gleichzeitige Verbindung mehrerer Eingänge mit mehreren Ausgängen ermöglicht.

Mit derartigen Switches können fast beliebige Datenraten erreicht werden. In Abb. 11.1.2 ist als Beispiel ein Switch mit 8 Eingängen und 8 Ausgängen gezeigt, der gleichzeitig 8 parallele Verbindungen ermöglicht. In jedem Augenblick kann durch entsprechende Steuerung jeder Eingang mit jedem Ausgang verbinden sein.

Die Personalisierung der 8 x 8 = 64 Transistorswitche erfolgt in einfachsten Fall durch einen 8 x 8 Bit Speicher, der durch einen eigenen Mikroprozessor angesteuert wird. Durch Änderung des Speicherinhalts können die Verbindungen dynamisch geändert werden.

Zur Erhöhung der Bandbeite lassen sich 8 oder 32 derartige Switche übereinanderstapeln, um einen 8 oder 32 Bit breiten Bus für jeden Eingang/Ausgang zu implementieren.

Crossbar Switch Silizium Chips mit je 256 Ein/Ausgängen lassen sich in der heutigen Silzium Technik kostengünstig herstellen.

11.1.3 Moderne Cluster Implementierung

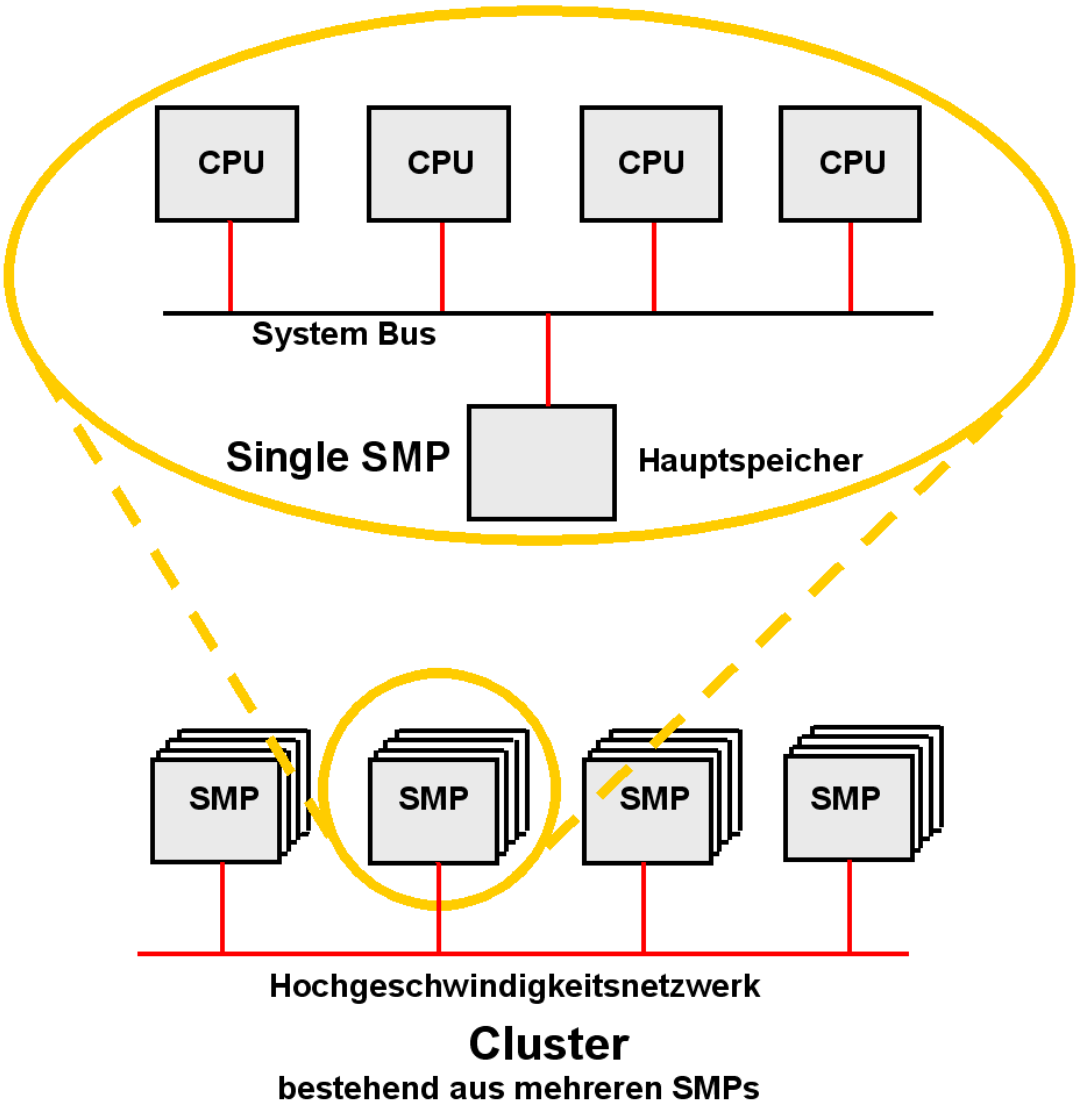


Abb. 11.1.3
Der Knoten eines Clusters besteht aus mehreren CPUs

Häufig (heute fast immer) sind die Elemente eines Clusters nicht einzelne CPUs, sondern SMPs, die als Knoten (und in der IBM Terminologie als „Systeme“) bezeichnet werden.

Jeder Knoten ist ein SMP. Er besteht aus mehreren CPUs, die auf einen gemeinsamen Hauptspeicher mit einer einzigen Instanz des Betriebssystems zugreifen.

Die Knoten sind über ein Hochgeschwindigkeitsnetzwerk miteinander verbunden, das in der Regel als Crossbar Switch implementiert wird.

Die Großrechner von HP, IBM und Sun haben diese Struktur.

Ein SMP wird als eng gekoppelter, ein Cluster als loose gekoppelter Multi-Prozessor bezeichnet.

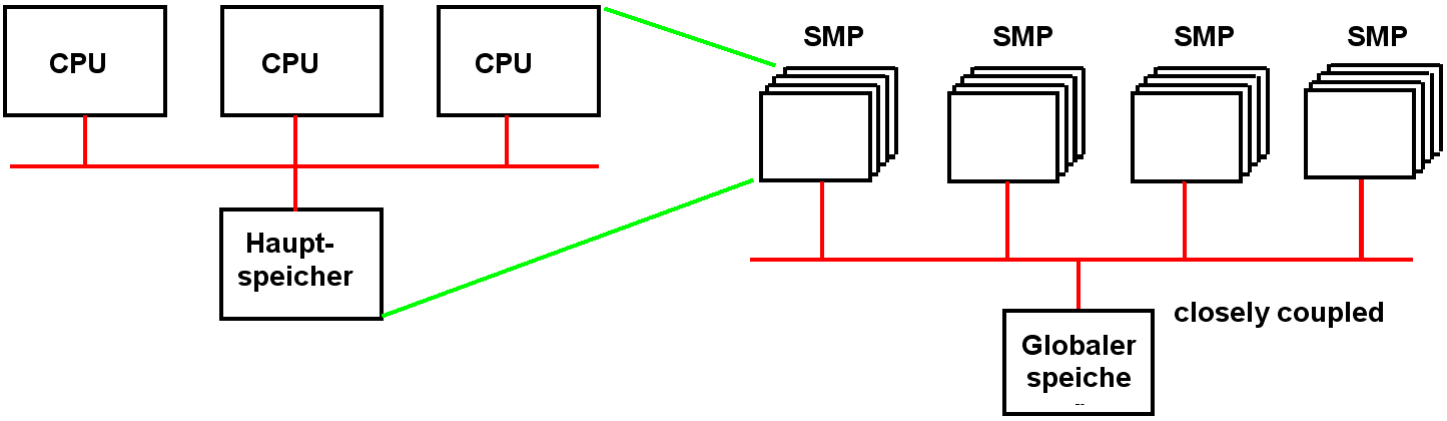


Abb. 11.1.4
Cluster mit globalem Speicher

Ein Cluster, bei dem die Knoten aus einzelnen CPUs oder aus SMPs bestehen, kann durch einen globalen Speicher erweitert werden. Diese Konfiguration wird als nahe gekoppelt (closely coupled) bezeichnet. Der Mainframe Sysplex ist eine derartige Konfiguration.

11.1.4 Unix Großrechner

Abb. 11.1.5
512 MByte DIMM (Wikipedia)

Abb. 11.1.6 – 11.1.9 zeigen Beispiele von Unix basierten Großrechnern der Firmen Sun und Hewlett Packard. Das erste Beispiel betrifft den E 15 000 Rechners der Firma Sun.

Zentralstück des Rechners ist ein Crossbar Switch, der sich auf einem Motherboard zusammen mit 16 Steckplätzen für sog. „System Boards“ befindet. Abb. 11.1.6 zeigt ein derartiges System Board.

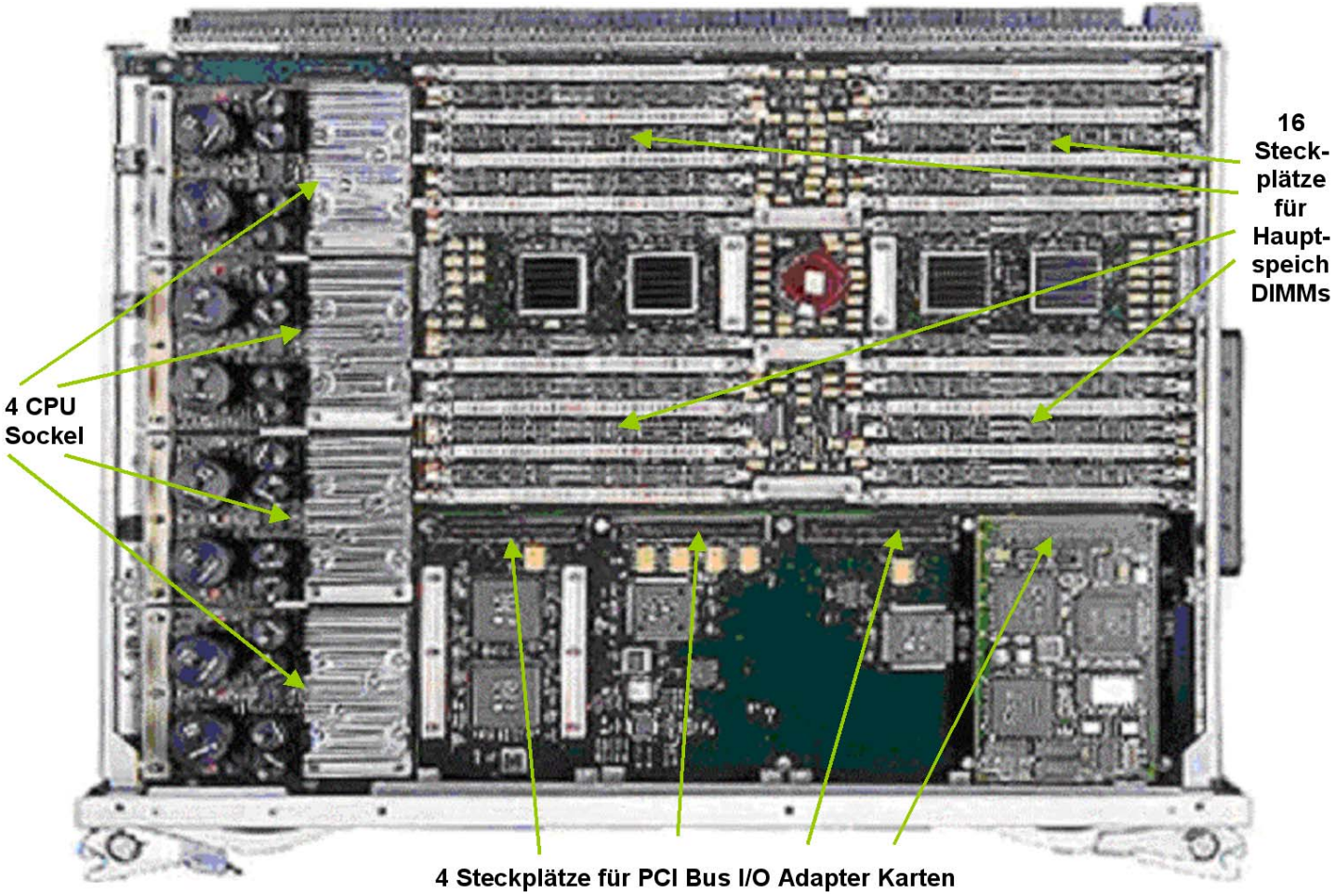


Abb. 11.1.6
Sun Fire 15 000 System Board [Sun2006]
Wiederholung von Abb. 4.1.3

Auf dem System Board befinden sich 4 CPU Chip Sockel. Zu sehen sind die Kühlkörper. CPU Chips waren früher Single Core, sind heute (2012) in der Regel aber Dual Core oder Quad Core (oder mehr). Weiterhin befinden sich auf dem System Board 16 Steckplätze für Hauptspeicher DIMMs. Als Dual Inline Memory Module (DIMM) werden Speichermodule für den Arbeitsspeicher von Computern bezeichnet. Im Gegensatz zu Single Inline Memory Modulen (SIMM) führen DIMMs auf den Anschlusskontakten auf der Vorderseite und auf der Rückseite der Leiterplatte unterschiedliche Signale.

Zusätzlich enthält das System Board 4 Steckplätze für sog. Daughter Cards. Es stehen eine Reihe unterschiedlicher Arten von Daughter Cards zur Verfügung, aber die allermeisten Steckplätze werden von I/O Adapter Karten eingenommen, die z.B. mit einem SCSI Kabel eine Verbindung zu Plattenspeichern ermöglichen.

http://docs.oracle.com/cd/E19065-01/servers.12k/806-3509-13/806-3509-13.pdf, oder als Mirror http://www.cedix.de/VorlesMirror/Band1/SunFire15k.pdf

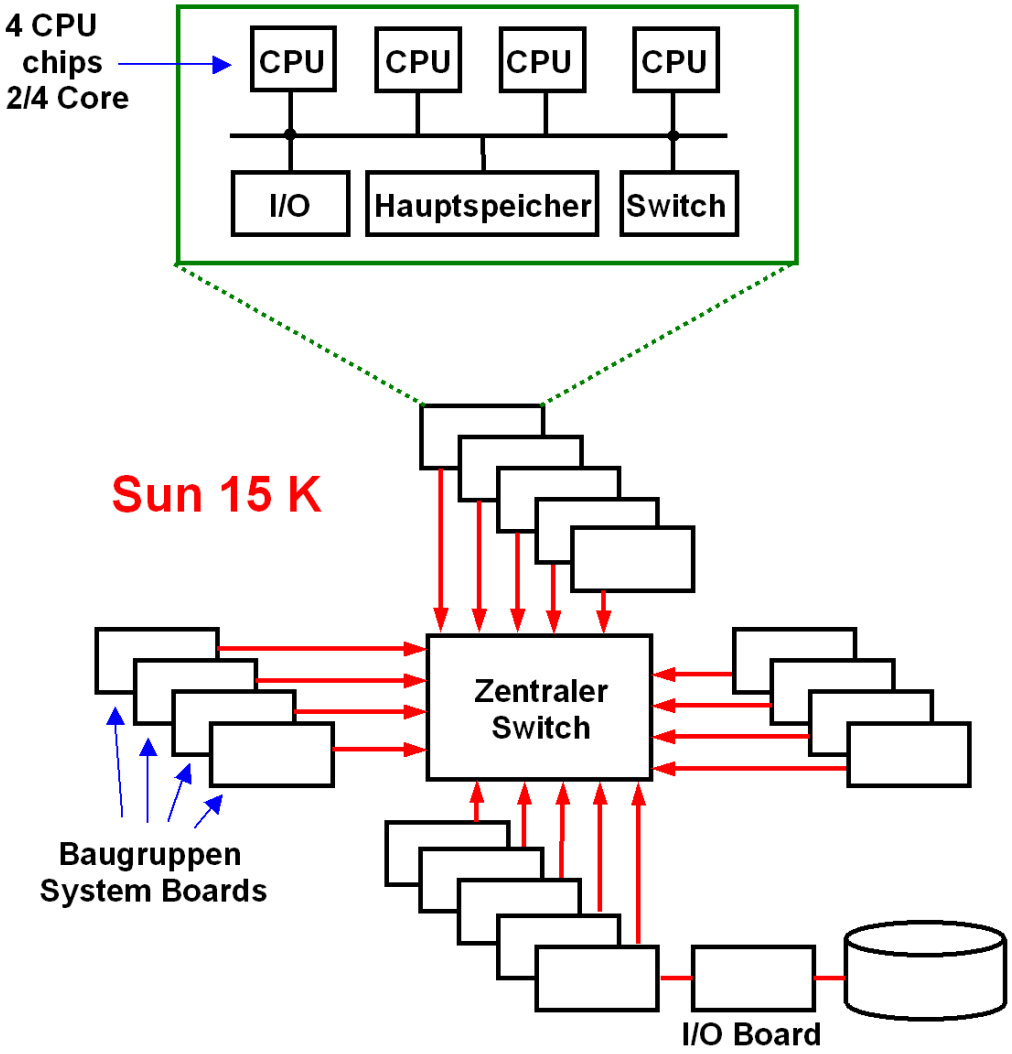


Abb. 11.1.7
Sun Fire 15 000 System Konfiguration

Ein Sun 25K oder HP Superdome Großrechner enthält 16 (oder 18) Printed Circuit Board (PCB) Baugruppen (System Boards), jedes mit 4 CPU Chip Sockeln und bis zu 128 GByte Hauptspeicher, einem Anschluss an einen zentralen Switch sowie I/O Adapter auf jedem System Board.

Die I/O Adapter sind mit PCI Bus Kabeln mit einer Reihe von I/O Cages verbunden, in denen sich Steckkarten für den Anschluss von I/O Geräten, besonders Festplattenspeichern, befinden.

Die CPUs eines jeden System Boards können nicht nur auf den eigenen Hauptspeicher, sondern auch auf den Hauptspeicher eines jeden anderen System Boards zugreifen. Hiermit wird eine „Non-Uniform Memory Architecture“ (NUMA) verwirklicht.

Eine moderne Form eines System Boards wird als „Blade“ bezeichnet, und befindet sich in vielen Low End Produkten der Firma Sun.

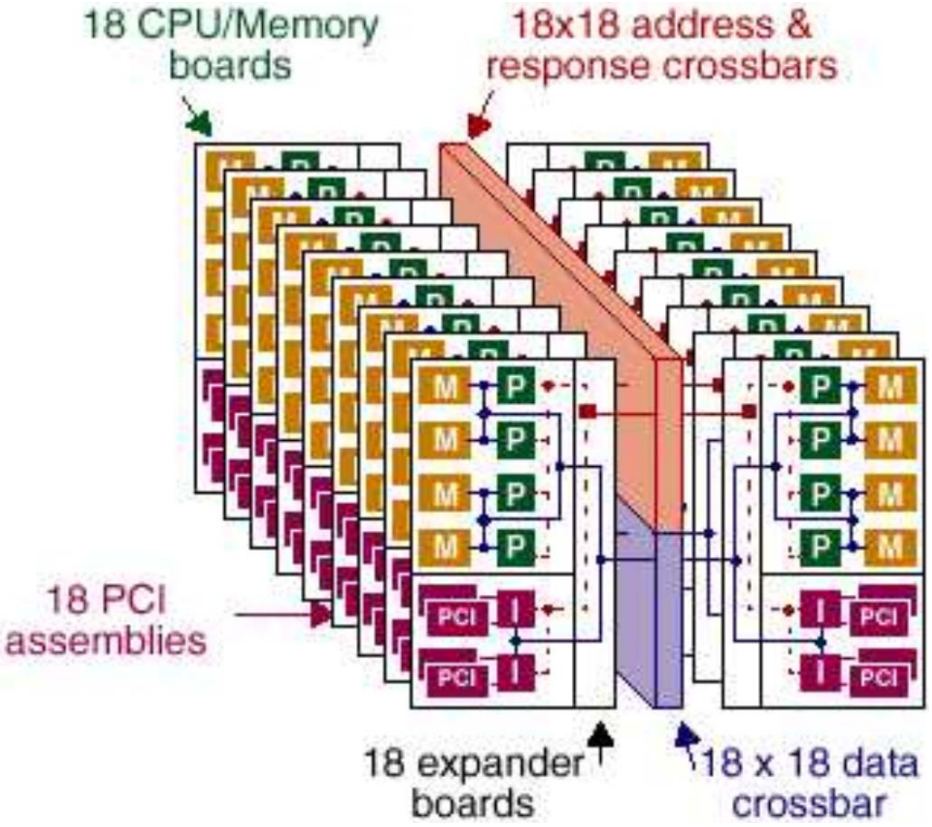


Abb. 11.1.8
Assembly mit 18 Sun Fire 15k System Boards

Die 16 (oder 18) System Boards befinden sich in Steckplätzen auf einem Crossbar Board. Dieses enthält eine Reihe von Crossbar Chips, welche alle System Boards miteinander verbindet. Dies sind spezifisch ein 16 x 16 Daten Crossbar Chip, ein 16 x 16 Response Crossbar Chip und ein 16 x 16 Adressen Crossbar Chip.

Das Crossbar Board enthält eine Reihe von (elektronischen) Schaltern. Mit diesen lassen sich die System Boards in mehrere Gruppen aufteilen und die Gruppen voneinander isolieren.

Da ein Unix SMP mit 64, 128 oder 256 CPUs bei Transaktions- und Datenbankanwendungen nicht mehr skaliert, kann der Rechner mit Hilfe der Schalter in mehrere voneinander isolierte SMPs aufgeteilt werden.

Eine solche Aufteilung wird als „Harte Partitionierung“ bezeichnet (siehe Abschnitt 12.1).

Die Firma Sun hat ihre Produkte kontinuierlich weiterentwickelt ohne dass sich am Konzept in den letzten 10 Jahren viel geändert hat. Die neuesten Modelle werden als SPARC Enterprise M9000 Server oder M-Series Server bezeichnet, und gemeinsam von Sun und von Fujitsu/Siemens vertrieben. Sparc M5-32 ist das neueste (2013) Modell mit 192 Cores und 32 TByte Hauptspeicher.

Ein derartiger Rechner ist nicht gerade billig. Eine Minimalkonfiguration hat einen Listenpreis von über 1 Million $. Je nach Ausstattung gehen die Preise von dort steil nach oben. Die Konkurrenzprodukte von Hewlett Packard und IBM sind eher noch teurer.

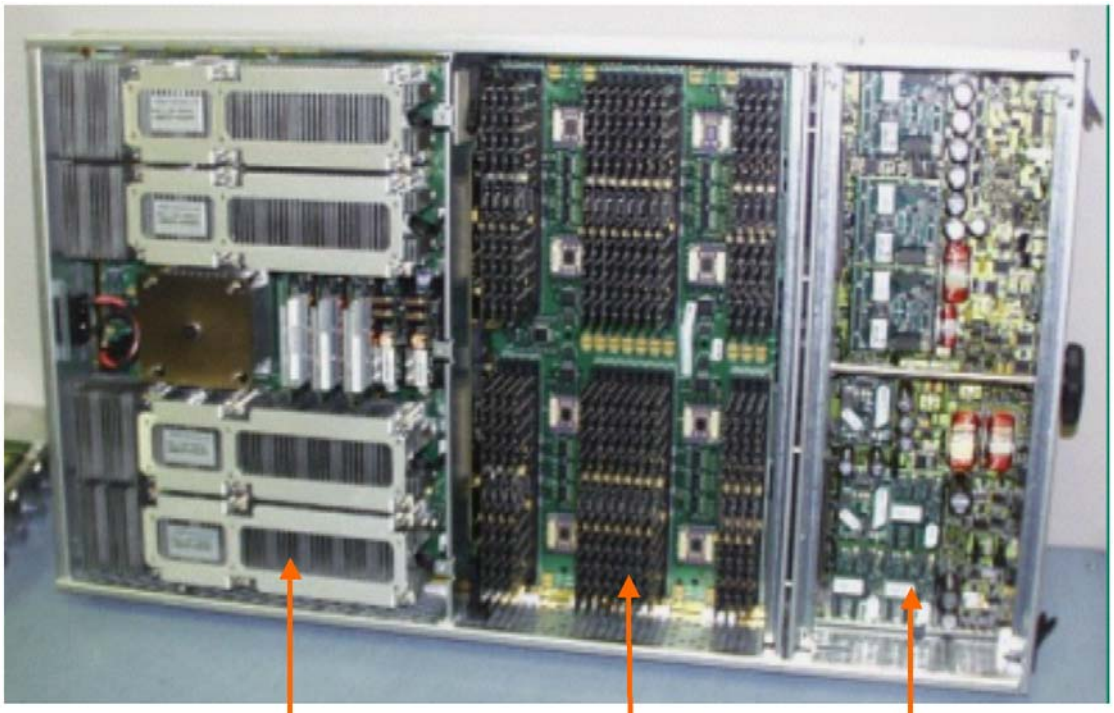


Abb. 11.1.9
HP Superdome Cell Board [HP2005], [HP2010]

Der Hewlett Packard (HP) Superdome Rechner ist ähnlich aufgebaut wie der Sun Fire 25K bzw. Sun M9000 Rechner. Das System Board wird als Cell Board bezeichnet und hat 4 Sockel für 4 Itanium CPU Chips mit je 4 CPU Cores. Neben den CPU Chips befinden sich auf dem Cell Board Sockel für Hauptspeicher DIMMs sowie Steckplätze für PCI Bus I/O Adapter Karten. Wie beim Sun Fire 25K sind die Cell Boards über einen zentralen Cross Bar Matrix Switch miteinander verbunden.

Auch die Firma Hewlett verwendet eine vereinfachte Form eines Cell Boards (als „Blade“ bezeichnet) in vielen Low End Produkten des Unternehmens. Cell Boards haben gegenüber Blades zahlreiche zusätzliche Funktionen. Beispiele sind:

- **Verbindung über einen zentralen Switch**
- **NUMA Fähigkeit**
- **Partitionsmöglichkeiten (siehe nächstes Thema)**
- **Zusätzliche unterstützende Hardware für das HP-UX Betriebssystem**
- **Verbesserte I/O Einrichtungen, höhere I/O Kapazität**
- **Einrichtungen für eine zentrale Administration**
- **Zusätzliche Verbesserungen für Ausfallsicherheit, Zuverlässigkeit und Verfügbarkeit**
- **.....**

IBM bietet als Alternative zu den Sun M9000 oder HP Superdome Systemen mit den Solaris oder HP-UX Betriebssystemen das ähnliche Produkt „System p“ mit den Betriebssystemen AIX und i5/OS an. Aus Platzgründen wird auf diese Entwicklung nicht näher eingegangen. AIX ist ein besonders funktionsreiches Unix Betriebssystem.

http://www.theregister.co.uk/2012/11/09/hp_integrity_superdome_itanium_9500/
http://www.serverworldmagazine.com/webpapers/2001/05_hpsuperdome.shtml

11.1.5 Sysplex

Der Sysplex (SYStem Processing compLEX) ist eine 1990 für Mainframes eingeführte lose Rechnerkopplung von IBM Mainframes. Oft wird diese einfache Form des Clusters auch Base Sysplex genannt.

Der Parallel Sysplex ist eine Weiterentwicklung des Base Sysplex, dessen Einführung 1994 erfolgte. Hierbei handelt es sich um eine nahe Rechnerkopplung (closely coupled). Da der Base Sysplex praktisch ausgestorben ist, bezeichnet ein Sysplex in der Umgangssprache fast immer einen Parallel Sysplex.

Beim Parallel Sysplex übernimmt die Coupling Facility (CF) die Funktion eines globalen Arbeitsspeichers. Innerhalb der CF können drei verschiedene Datenstrukturtypen erzeugt und verwaltet werden. Dies sind Sperr- (Lock), Listen- und Cache-Strukturen, die u.a. genutzt werden um Konkurrenzsituationen beim Zugriff auf Resourcen zu verwalten. Die Hardware der Coupling Facility besteht aus einem regulären Mainframe Rechner, auf dem der „Coupling Facility Control Code“ (CFCC) an Stelle eines Betriebssystems läuft.

Nutzen Subsysteme wie CICS, DB2 oder IMS die durch den Parallel Sysplex bereitgestellten Einrichtungen, so können sie sich gegenüber dem Anwender als eine einzige Anwendung präsentieren. Man spricht dann auch von einem Single System Image (SSI).

Literatur: Wilhelm G. Spruth, Erhard Rahm, Sysplex-Cluster Technologien für Hochleistungs-Datenbanken. Datenbank-Spektrum, Heft 3, 2002, S. 16-26,
http://www.cedix.de/Publication/Mirror/Sysplex3.pdf

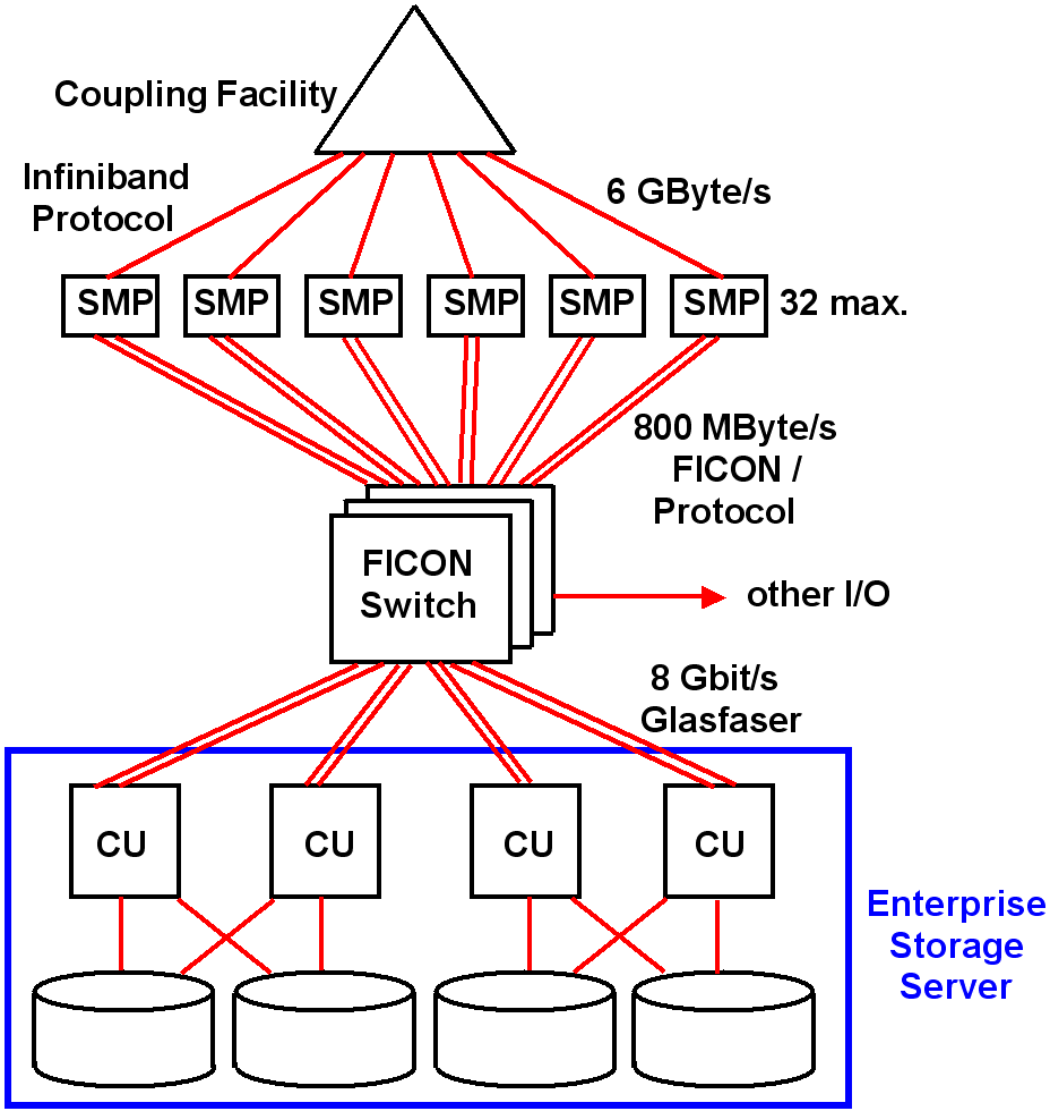


Abb. 11.1.10
Sysplex mit Coupling Facility

Die folgende Abbildung zeigt den Aufbau der Parallel Sysplex-Architektur. Während die vorgestellten Cluster von Sun und HP eine lose Kopplung verwenden, handelt es sich beim Parallel Sysplex um eine nahe gekoppelt (closely coupled) Cluster-Architektur. Der Sysplex besteht aus bis zu 32 System z Rechnern (Knoten), fast immer SMPs, gemeinsam genutzten Festplattenspeichern (Shared Storage Devices) in der Form von einem oder mehreren Enterprise Storage Servern, Netzwerk-Controllern und den Kern-Cluster-Technologie-Komponenten. Letztere umfassen den (die) als „FICON Director“ bezeichneten Crossbar-Switch(es), den Sysplex-Timer und einer (oder in der Regel 2) „Coupling Facility“ (CF). Die Coupling Facility enthält den für die nahe Kopplung charakteristischen globalen Speicher zur Realisierung globaler Kontrollaufgaben und ist von allen Knoten schnell zugreifbar.

Eine Coupling Facility ist ein regulärer System z Rechner, auf dem „Coupling Facility Code“ an Stelle eines Betriebssystems läuft. Es ist üblich, die CF in grafischen Darstellungen als Dreieck wiederzugeben.

Weiterhin sind 1 oder 2 Zeitgeber (Sysplex Timer) vorhanden. Bei den heutigen Rechnern ist die Zeitgeberfunktion in die CPU Chips integriert.

Zu den CPUs kommen noch weitere Ein-/Ausgabe-Prozessoren (System Assist-Prozessoren, SAPs). Heutige Installationen haben bis zu 200 CPUs. Die Knoten müssen nicht homogen sein, d.h. es können unterschiedliche Mainframe Modelle eingesetzt werden.

Zu den Sysplex-Komponenten werden spezifische Maschinenbefehle sowie Betriebssystemdienste zur Verfügung gestellt, mit denen eine effiziente Durchführung der Cluster-Aufgaben für alle Knoten erreicht wird, insbesondere zur Kommunikation, Ein/Ausgabe, sowie für globale Steuerungsaufgaben wie Synchronisation, Kohärenzkontrolle und Lastbalancierung. Diese Dienste werden in allgemeiner Form realisiert und von unterschiedlichen Software-Subsystemen, insbesondere Web Application Server, Datenbanksystemen und Transaktionsservern für den Cluster-Einsatz verwendet. IBM hat dazu die wichtigsten Subsysteme an das Arbeiten innerhalb eines Sysplex angepasst. Eine Anpassung der Anwendungen an den Sysplex ist in der Regel nicht erforderlich und i.A. auch nicht möglich.

Jeder Knoten kann mit bis zu 256 „Kanälen“ mit der Außenwelt verbunden werden, mit bis zu 800 MByte/s pro Kanal. Die Verbindung der Systeme sowohl untereinander als auch zu den Festplattenspeichern erfolgt über ein oder mehrere FICON (FIbre CONnection)-Switches. Da jedes System 256 Ein-/Ausgabe-Kanäle anbinden kann, hat diese Verbindungsstruktur die Aufgabe, bei 32 Systemen maximal 256 * 32 = 2^{13} Kanäle zu verwalten. Damit ist jeder Knoten in der Lage, auf alle Plattenspeicher und alle anderen Knoten des Sysplex direkt zuzugreifen (Shared Disk-Modell). Die Steuereinheiten der Plattenspeicher (Control Units) implementieren eine Storage Server- und SAN-Funktionalität.

Die von IBM entwickelte FICON-I/O-Architektur basiert auf dem Kanal-Subsystem (channel subsystem) der Mainframe I/O-Architektur. Dieses integriert System Assist Prozessoren (SAPs), Kanäle sowie die „Staging Hardware“. Die SAPs führen die Kommunikation zwischen den Knoten und den Kanälen durch. Diese führen zur Datenübertragung ein Kanalprogramm aus, wobei über die Steuereinheiten (Control Units) auf die Plattenspeicher zugegriffen wird. Die Staging Hardware stellt die Kommunikationspfade zwischen den I/O-Prozessoren, den Kanälen und dem Rest des Systems zur Verfügung.

In der FICON-Architektur bildet der FICON-Switch (FICON Director) die Kerneinheit. Dies implementiert eine Switched Point-to-Point-Topologie für System z I/O-Kanäle und Steuereinheiten. Ein FICON-Switch kann bis zu 60 Kanäle und Steuereinheiten dynamisch und nicht-blockierend (Crossbar Switch) über seine Ports miteinander verschalten. Normalerweise werden eine Reihe von FICON-Switches parallel genutzt. Entfernungen von bis zu 3 km für optische Übertragungen sind möglich. Die zulässigen Entfernungen erhöhen sich beim Einsatz einer sogenannten Extended Distance Laser Link-Einrichtung auf 20, 40 oder 60 km. Dies ist für Unternehmen wichtig, die aus Katastrophenschutz Gründen zwei getrennte Rechenzentren betreiben.

Die Kommunikation zwischen den Knoten und den Plattenspeichern erfolgt über das FICON-Protokoll. Zur Kommunikation der Knoten untereinander wird das „Channel-to-Channel“ (CTC)–Protokoll eingesetzt, normalerweise in einer Full-Duplex-Anordnung. Hierbei betrachtet jeder sendende Knoten den Empfänger als eine Ein-/Ausgabe-Einheit. Der Empfänger simuliert für diesen Zweck einen eigenen Typ einer Mainframe-Steuereinheit. Das z/OS-Betriebssystem stellt einen Basisdienst, die „Cross System Coupling Facility“ (XCF) zur Verfügung, um über eine CTC-Verbindung eine Kommunikation mit einer anderen z/OS-Instanz innerhalb eines Sysplex zu bewerkstelligen. Dieser Dienst wird wiederum von Subsystemen wie den Datenbanksystemen DB2 und IMS für den Nachrichtenaustausch eingesetzt.

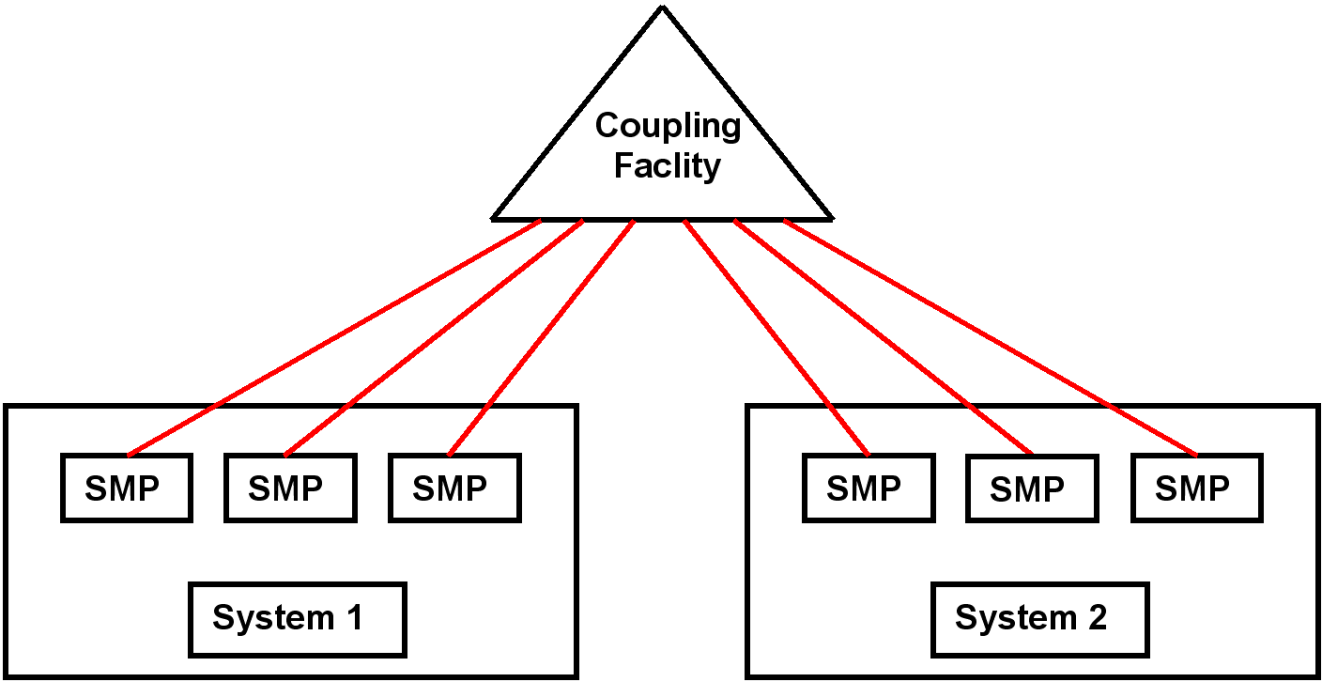


Abb. 11.1.11
Ein physische Mainframe Rechner (System) besteht häufig aus mehreren SMPs

In einem Sysplex werden bis zu 32 Knoten unterstützt. Jeder Knoten stellt eine einzelne CPU (oder in den allermeisten Fällen einen SMP) dar und enthält maximal 101 CPUs, insgesamt also maximal 3232 CPUs. Dieser Wert kann in der Praxis nicht erreicht werden, weil SMPs in der Regel nicht bis zu 101 CPUs skalieren. Eine große Sysplex Installation besteht deshalb in der Regel aus mehreren physischen Rechnern (Systeme in der IBM Terminologie), von denen jeder mehrere SMPs (Knoten) enthält. Aus logischer Sicht spielt es dabei keine Rolle, ob zwei SMPs im gleichen oder in getrennten physischen Rechnern untergebracht sind.

Jeder Knoten ist über eine eigene Glasfaserverbindung (Coupling Link) mit der Coupling Facility verbunden.

Zur Erhöhung der Reliability und Availability werden in der Praxis fast immer 2 Coupling Facilities eingesetzt, von denen die zweite CF als Backup für die erste CF dient.

11.1.6 Rechner zu Rechner Kommunikation

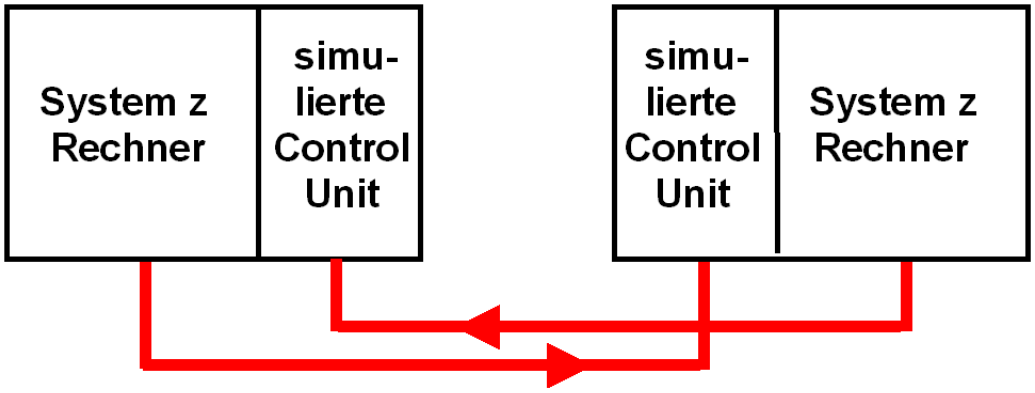


Abb. 11.1.12
Duplex Channel- to Channel Verbindung

Die Cross-System Coupling Facility (XCF) ist eine Komponente des z/OS Kernels.

Sie verwendet das CTC (Channel To Channel) Protokoll und stellt die Coupling Services bereit, mit denen z/OS Systeme innerhalb eines Sysplex miteinander kommunizieren.

Für eine CTC Verbindung stellt ein System z oder S/390 Rechner (als Slave bezeichnet) eine emulierte Control Unit zur Verfügung. An diese ist ein anderer Rechner (Master) über eine normale FICON Glasfaserverbindung angeschlossen. Der Master kommuniziert mit dem Slave wie mit einer normalen I/O Einheit.

Die CTC Verbindung ist unidirektional. Aus Symmetriegründen werden CTC Verbindungen normalerweise paarweise eingerichtet.

Mittels CTC und XCF können die Rechner eines Sysplex miteinander kommunizieren. Dies geschieht über FICON Directors, und nutzt das gleich FICON Netzwerk, was auch für die Verbindung der Rechner mit Enterprise Storage Servern und Magnetbändern benutzt wird.

Zu den Parallel Sysplex Cluster Technology Komponenten gehören:

- **Prozessoren mit Parallel Sysplex Fähigkeiten**
- **Coupling Facility**
- **Coupling Facility Control Code (CFCC)**
- **Glasfaser Hochgeschwindigkeitsverbindungen**
- **FICON Switch**
- **Gemeinsam genutzte Platten (Shared DASD)**
- **System Software**
- **Subsystem Software**

Die Coupling Facility ermöglicht Data Sharing einschließlich Datenintegrität zwischen mehrfachen z/OS Servern.

11.2 Coupling Facility

11.2.1 zSeries Coupling Facility

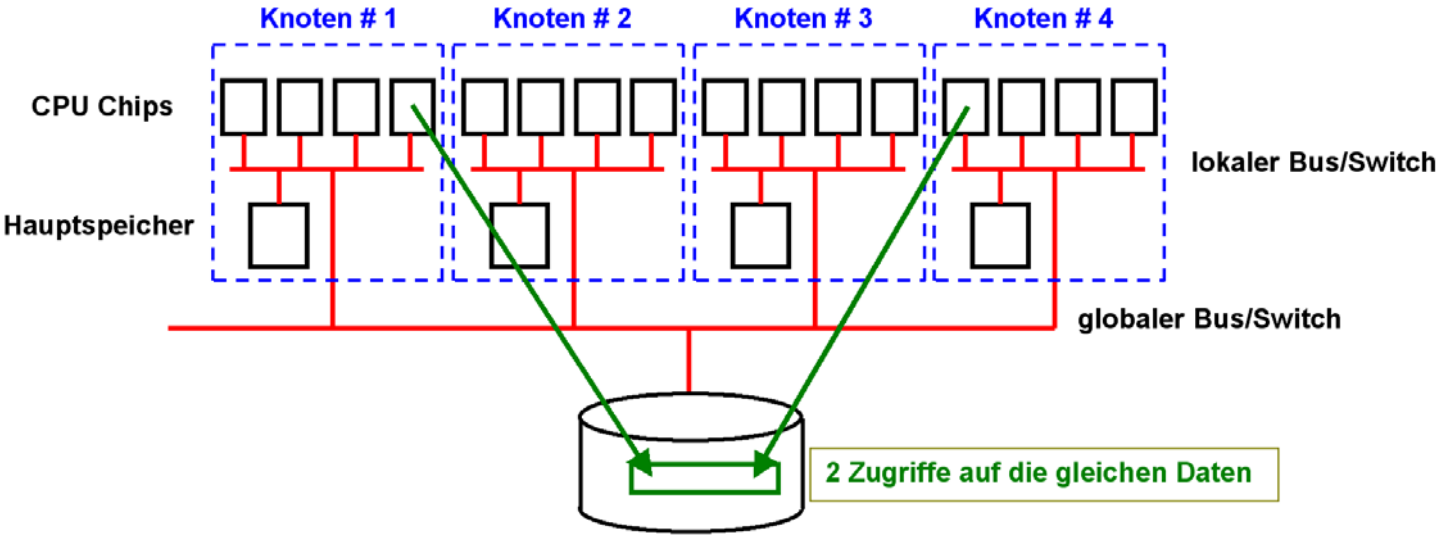


Abb. 11.2.1
Zwei Systeme greifen gleichzeitig auf die gleichen Daten zu

Cluster haben den Vorteil, dass mehrfache Instanzen (Kopien) des Betriebssystems vorhanden sind. Deshalb bestehen alle modernen Großrechner aus einem Cluster von SMPs.

Allerdings besteht hierbei das Problem, den gleichzeitigen Zugriff mehrerer Knoten auf den gleichen Datenbestand zu lösen. Dies geschieht mit Hilfe von Locks (Sperren) auf Teile der gemeinsam genutzten Daten. Die Effektivität des Lock Managements (Sperrverwaltung) bestimmt die Skalierungseigenschaften eines Clusters.

Die Coupling (CF) Facility ist in Wirklichkeit ein weiterer System z Rechner mit spezieller Software. Die Aufgaben der CF sind:

- **Locking**
- **Caching**
- **Control/List Structure Management**

Der Hauptspeicher der Coupling Facility enthält einen als „Coupling Facility Control Code“ (CFCC) bezeichnetes Betriebssystem, sowie Speicherbereiche für Locking, Caching und Control/List Strukturen.

Die Coupling Facility ist über Glasfaser Verbindungen mit einem optimierten Protokoll (Infiniband) und spezieller Hardware Unterstützung mit den Knoten (Systemen) des Sysplex verbunden.

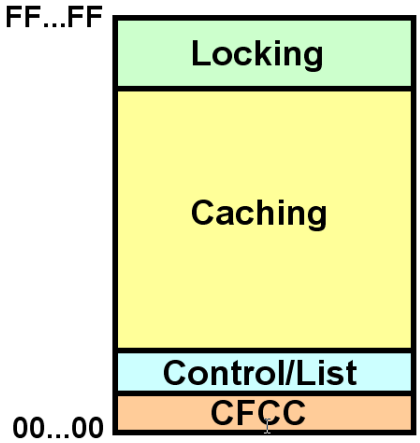


Abb. 11.2.2
Locking, Caching und Control/List sind die 3 wichtigsten Funktionen der CF

Die wichtigste Aufgabe der Coupling Facility ist ein zentrales Lock Management für die angeschlossenen Knoten. Der zentrale Lock Manager des SAP System R/3 hat in Ansätzen eine ähnliche Funktionalität, jedoch ohne die CF spezifischen Eigenschaften.

Der größte Teil des Hauptspeichers der Coupling Facility wird als Plattenspeicher Cache genutzt. Der CF Cache dupliziert den Plattenspeicher Cache (Buffer Pool) in den einzelnen Systemen. Ein Cast out der CF Cache auf einen Plattenspeicher erfolgt über ein System (Knoten).

CF Cache Cross-Invalidate (ungültig machen) Nachrichten gehen nur an die betroffenen Systeme.

Control und List Strukturen dienen der Sysplex Cluster weiten Verwaltung. Ein Beispiel ist ein den ganzen Cluster umfassendes RACF Sicherheits-Subsystem. Ein weiteres Beispiel ist der bereits erwähnte WebSphere MQ Cluster (siehe Wintersemester MQSeries, Teil 4).

Eine interessante Überlegung: Das SAP System R/3 würde erheblich vom Vorhandensein einer Coupling Facility profitieren. Das Problem ist: SAP System R/3 ist als Multi-Plattform Software ausgelegt, lauffähig nicht nur unter z/OS, sondern auch unter den verschiedenen Unix Dialekten, Linux und Windows. Um eine CF nutzen zu können, wären identische Hardware Einrichtungen auf Sparc, Itanium, PowerPC und x86 Rechnern erforderlich.

Die Lauffähigkeit auf unterschiedlichen Platformen hat ihren Preis.

11.2.2 Coupling Support Facility

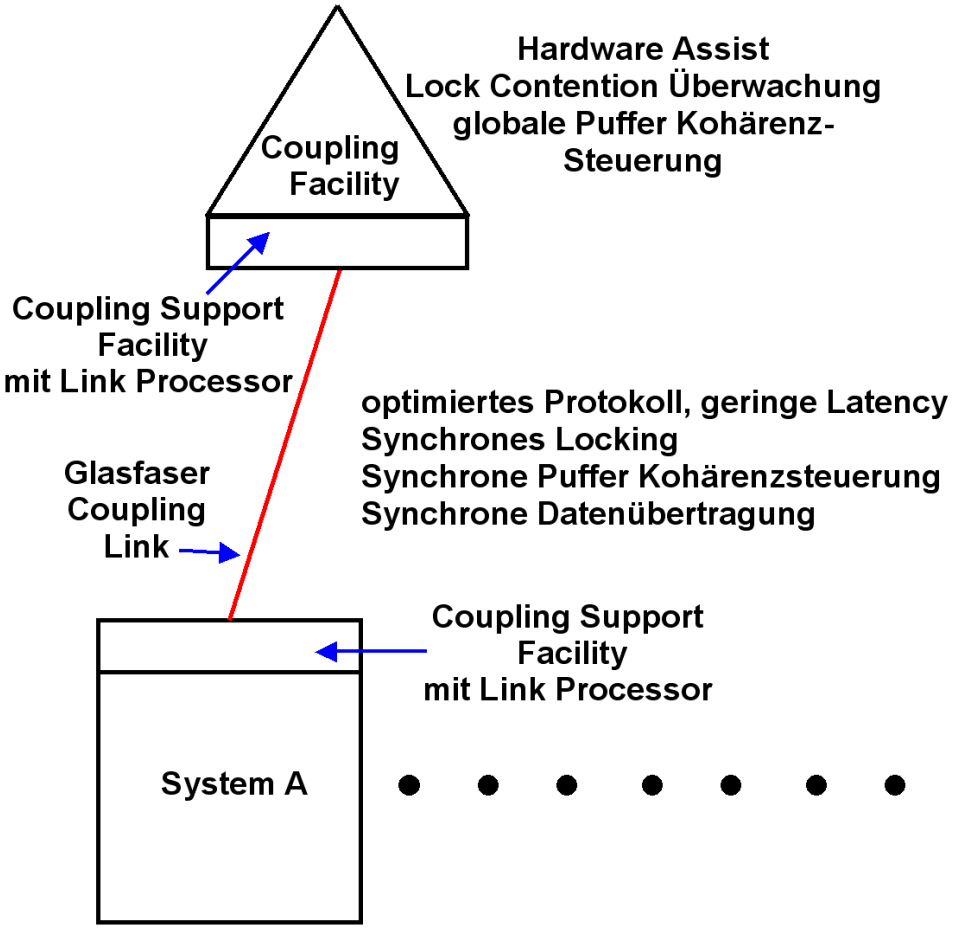


Abb. 11.2.3
Anbindung eines Systems an die Coupling Facility

Jeder Knoten (System) eines Sysplex sowie die CF selbst enthalten eine spezielle Hardware Einrichtung, die „Coupling Support Facility“. Diese bewerkstelligt die Kommunikation und den Datenaustausch zwischen Knoten und CF.

Die Coupling Support Facility besteht aus je einer Glasfaserverbindung (800 MByte/s theoretisches Maximum), einem dedizierten Link-Prozessor für die Verarbeitung des (Infiniband) Übertragungsprotokolls und einer Erweiterung des System z Maschinenbefehlssatzes. Die zusätzlichen Maschinenbefehle realisieren die nahe Kopplung zwischen Knoten und CF und ermöglichen allen Knoten den Zugriff auf die Inhalte in den CF-Datenstrukturen.

Das Kommunikationsprotokoll wurde für eine besonders geringe Latenzzeit optimiert, welches synchrone Zugriffe eines jeden Knotens auf die CF ermöglicht. Synchron bedeutet, dass ein auf einem Knoten laufender Prozess während eines Zugriffs auf die Coupling Facility blockiert: Er bleibt im Zustand „running“, und es findet kein Prozesswechsel und kein Übergang "User-Status - Kernel-Status" statt. Die Zugriffsgeschwindigkeit liegt im Mikrosekundenbereich und ist somit weitaus effizienter als die Kommunikation über allgemeine Protokolle bei loser Rechnerkopplung.

Die Coupling Facility ist durch eine Punkt-zu-Punkt Glasfaserleitung mit jedem System (Knoten) des Sysplex verbunden (Coupling Link).

Es wird ein spezielles Verbindungs-Protokoll (Infiniband) mit besonders geringer Latency eingesesetzt .

Die CF Glasfaser Verbindung wird durch spezielle Hardware Einrichtungen und durch zusätzliche Maschinenbefehle in jedem angeschlossenen System unterstützt (Coupling Support Facility).

Die Coupling Facility kann in einem angeschlossenen Rechner ohne Unterbrechung des laufenden Prozesses Daten in spezielle Speicherbereichen (Bit Vektoren) abändern.

11.2.3 Komponenten einer Coupling Facility

Die nahe Kopplung über die CF wird für die leistungskritischen Kontrollaufgaben in Shared-Disk-Clustern genutzt, um durch die effiziente Realisierung eine hohe Skalierbarkeit zu erreichen. Dies betrifft die globale Synchronisation über Sperrverfahren (Locking), die Kohärenzkontrolle der Pufferinhalte mit schnellem Austausch geänderter Daten zwischen den Knoten sowie die flexible Lastverteilung. Entsprechend werden spezifische Funktionen (Maschinenbefehle) für Locking, Caching und Queuing sowie zugeschnittene Datenstrukturen im Hauptspeicher der CF bereitgestellt. Die Hauptspeicher-Ressourcen der CF können dynamisch partitioniert und einer der CF-Strukturen zugewiesen werden. Innerhalb derselben CF sind mehrere CF-Strukturen desselben oder unterschiedlichen Typs möglich. Die auf diesen Strukturen bereitgestellten Funktionen bzw. Cluster-Protokolle repräsentieren drei Verhaltensmodelle:

Host hardware: Wenn eine CPU eine Anforderung an die Coupling Facility startet, wird diese zunächst mittels Firmware (Microcode, erläutert später im Modul Virtualisierung Teil 3) und spezieller Hardware verarbeitet, ehe sie über das Coupling Link weitergereicht wird.

Coupling Link: Die Anforderung wird dann über das Coupling Link an die Coupling Facility übertragen. Die Übertragungszeit wird durch die Link Geschwindigkeit, die Datenmenge, die Distanz zur Coupling Facility (Lichtgeschwindigkeit) bestimmt. Die Übertragung erfolgt nach Möglichkeit synchron, d.h. der laufende Prozess wird nicht unterbrochen. Deshalb ist die Latenz (die Zeit, die für die Übertragung eines einzelnen Bytes benötigt wird) besonders kritisch. Um die Latency zu minimisieren, verwenden CF Links deshalb entweder ein spezielles CF Link Protokoll, oder das Infiniband Protokoll. Beide zeichnen sich durch eine besonders geringe Latency aus.

Coupling Facility hardware: Die eintreffende Anforderung wird durch die Hardware (Coupling Support Facility) und den Code in der Coupling Facility (CFCC) bearbeitet. Je nach Bedarf sendet die Coupling Facility Nachrichten an die Knoten Rechner des Sysplex zurück.

Coupling Facility Control Code: Eine Coupling Facility ist ein normaler System z Rechner, der CFCC ausführt. Es handelt sich hierbei um ein normales hochspezialisiertes Betriebssystem mit einigen Besonderheiten. CFCC ist jedoch Firmware – erläutert im Abschnitt 12.3.1. CFCC Code ist aber gleichzeitig normaler System z Maschinencode.

Dies bedeutet, eine Coupling Facility verhält sich wie eine Black Box. z/OS kann der CF eine Nachricht schicken, und die CF reagiert irgendwie darauf. Weder der z/OS Systemprogrammierer, und erst Recht nicht der Anwendungsprogrammierer sind in der Lage, die CF zu programmieren.

Es ist deshalb möglich, für Experimentierzwecke CFCC Code in einer virtuellen Maschine unter z/VM laufen zu lassen. Dies haben wir auf dem Mainframe Rechner unseres Lehrstuhls installiert. Es wurde ein vollständiger virtueller Sysplex eingerichtet, der für Ausbildungs- und Übungszwecke genutzt werden kann.

11.2.4 Coupling Facility Structure

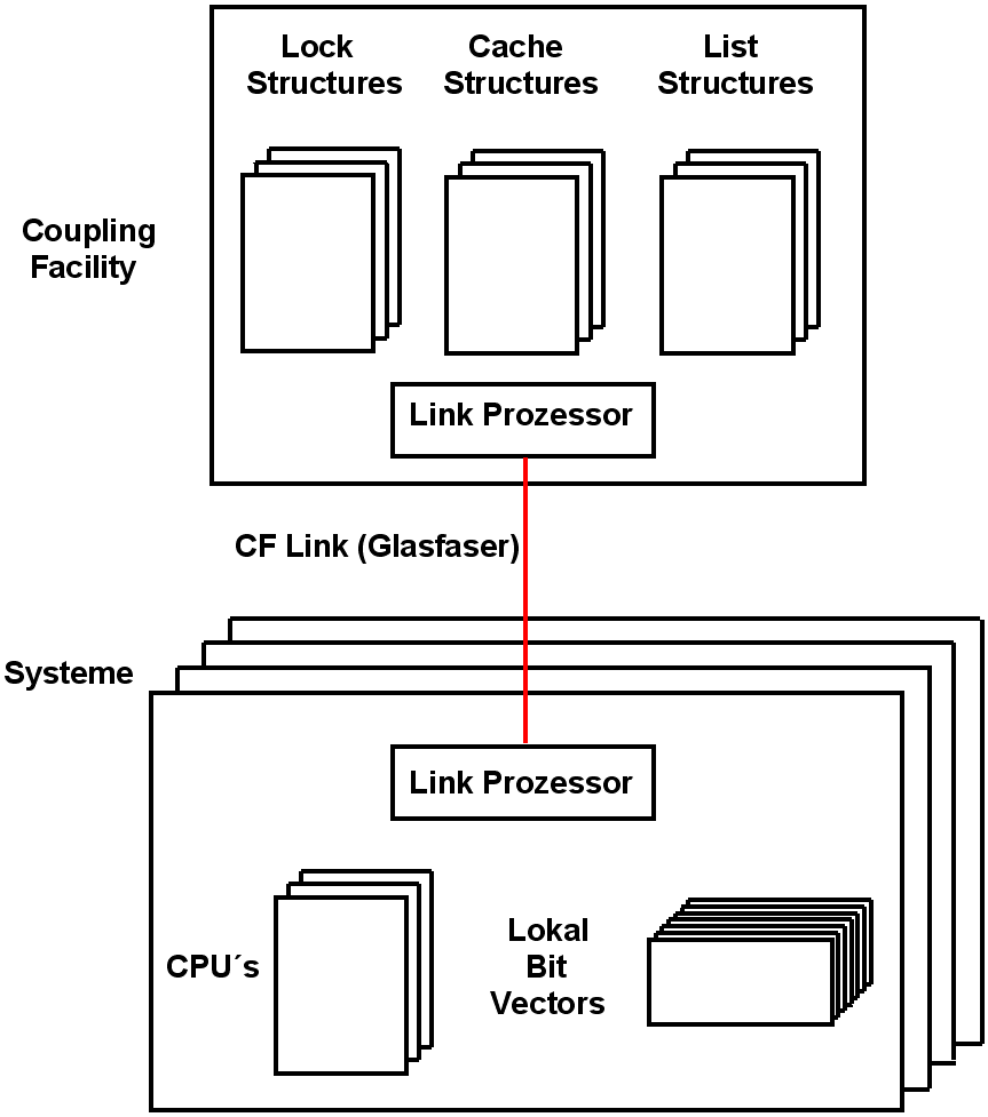


Abb. 11.2.4
Ein spezieller Link Prozessor koppelt die CF an jedes der beteiligten Systeme

Die nahe Kopplung über die CF wird für die leistungskritischen Steuerungslaufgaben genutzt, um durch die effiziente Realisierung eine hohe Skalierbarkeit zu erreichen. Dies betrifft die globale Synchronisation über Sperrverfahren (Locking), die Kohärenzkontrolle der Pufferinhalte mit schnellem Austausch geänderter Daten zwischen den Knoten sowie die flexible Lastverteilung. Entsprechend werden spezifische Funktionen (Maschinenbefehle) für Locking, Caching und Queuing sowie zugeschnittene Datenstrukturen im Hauptspeicher der CF bereitgestellt. Die CF Hauptspeicher-Ressourcen können dynamisch partitioniert und einer der CF-Strukturen zugewiesen werden. Innerhalb derselben CF sind mehrere CF-Strukturen desselben oder unterschiedlichen Typs möglich.

Die auf diesen Strukturen bereitgestellten Funktionen bzw. Cluster-Protokolle repräsentieren drei Verhaltensmodelle:

- **Lock-Modell: Es unterstützt feingranulares, globales Locking für hohe Transaktionsverarbeitungs-Performance und eine Signalisierung von Zugriffskonflikten.**
- **Cache-Modell: Es liefert eine globale Kohärenz-Steuerung für die verteilten Pufferinhalte der einzelnen Knoten sowie einen globalen Puffer in der CF (Shared Data Cache).**

List Modell (Queue-Modell): Es implementiert einen umfangreichen Satz an Listen und Queuing-Konstrukten für die Verteilung von Arbeitslasten, zur Realisierung einer schnellen Nachrichtenübertragung (Message Passing) sowie für die Verwaltung globaler Status-Informationen. Ein Beispiel für letzteres ist die Verwaltung globaler Sicherheitsrechte.

- **Locks (Sperren)**
- **Sysplex weiter Daten Cache**
- **Listen für Sysplex-weite Aufgaben.**

Die einzelnen Systeme des Sysplex sind mit jeder dieser Strukturen über das CF Link und die Link Prozessoren (Teil der Coupling Support Facility) direkt verbunden. Die Kommunikation CPU – CF wird durch spezifische Maschinenbefehle unterstützt.

Jedes System unterhält lokale, als „Bit Vectoren“ bezeichnete Speicherbereiche für eine logische Verbindung zu jeder Struktur in der CF. Es existieren getrennte Bit Vektoren für die Lock Strukturen, Cache Strukturen und List (Queue) Strukturen.

11.2.5 Zuordnung von Bit Vektoren zu CF Strukturen

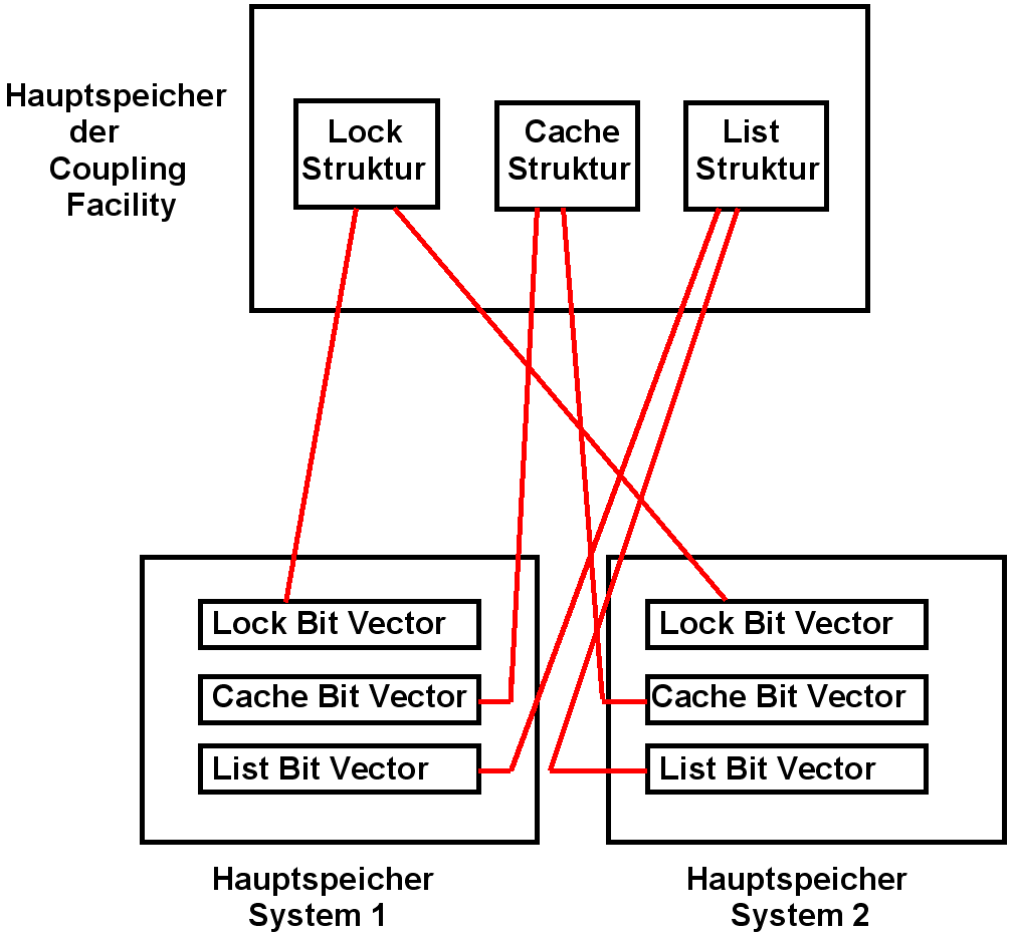


Abb. 11.2.5
Die Bit Vektoren speichern eine Untermenge der CF Information

Abb. 11.2.5 zeigt die Anbindung der Knoten (Systeme) an die CF und die dort verwalteten globalen Datenstrukturen.

Jeder angeschlossene Rechner dupliziert in seinem Hauptspeicher eine Untermenge der in der Coupling Facility gespeicherten Daten. Diese Untermengen werden als Bit Vektoren bezeichnet.

Der Zugriff erfolgt dabei nicht nur von den Systemen (Knoten) auf die CF, sondern die CF kann auch direkt (ohne Involvierung des Betriebssystems) auf bestimmte Hauptspeicherinhalte der Systeme, spezifisch die Bit-Vektoren, zugreifen und ändern. Solche Bit-Vektoren existieren für jede logische Verbindung zu einer CF-Datenstruktur und erlauben z.B. die schnelle Signalisierung von Zugriffskonflikten (s.u.).

In den nächsten Abschnitt betrachten wir die Nutzung der CF-Lock- und Cache-Strukturen zur Realisierung der clusterweiten Synchronisation (Locking) und Kohärenzkontrolle.

11.2.6 Locking Problem

Angenommen zwei Transaktionen, die auf unterschiedlichen Systemen des Sysplex laufen.

Beispiel: Beide Transaktionen greifen auf die zwei Variablen d1 und d2 zu.

Anfangswerte: d1 = 15, d2 = 20 .

Transaktion 1	Transaktion 2
read d1	
if d1 > 10	
●	read d1
●	if d1 > 10
sub d1, 10	●
●	sub d1, 10
add d2, 10	●
●	add d2, 10
end if	●
	end if

Abb. 11.2.6
In diesem Beispiel ist Locking erforderlich

Das Ergebnis ist: d1 = - 5, d2 = 40 , obwohl die if-Bedingung ein negatives Ergebnis verhindern sollte.

Die beiden Abhängigkeiten:

> **Dirty Read**
> **Eine Transaktion erhält veraltete Information**
>
> **Lost update**
> **Eine Transaktion überschreibt die Änderung einer anderen Transaktion**

müssen gesteuert werden.

Concurrency Control (Synchronisierung) ist zwischen den beiden Transaktionen erforderlich, um dies zu verhindern. Der Begriff, der dies beschreibt ist Serialisierung (serializability)

Erforderlich für Application Server, die Daten selbst speichern und auch auf Datenbanken zugreifen.

Es existieren viele Implementierungsmöglichkeiten. Gebräuchlich ist „two-phase locking“ (nicht zu verwechseln mit dem two-phase Commit Protokoll).

11.2.7 Two-Phase Locking

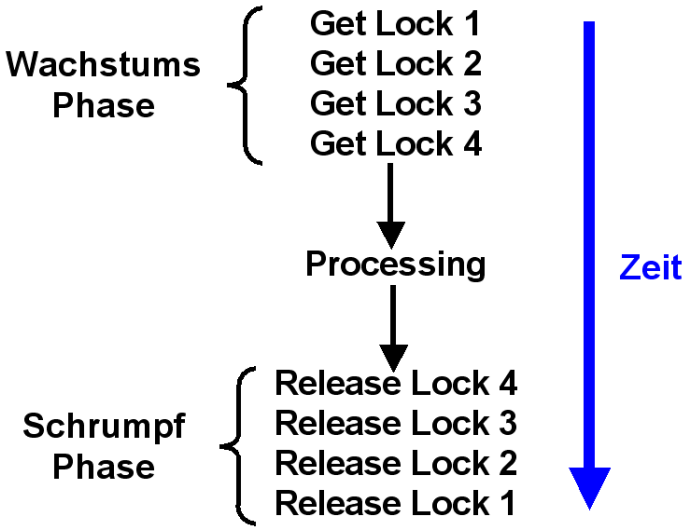


Abb. 11.2.7
Growth und Shrink Phase für die Beschaffung und Freigabe der Locks

In Transaktionssystemen und Datenbanksystemen werden Locks (Sperren) benutzt, um Datenbereiche vor einem unautorisierten Zugriff zu schützen. Jedem zu schützenden Datenbereich ist ein Lock fest zugeordnet. Ein Lock ist ein Objekt welches über 4 Methoden und zwei Zustände S und E verfügt. Die Methode

- **GetReadLock** — **reserviert S Lock (shared),**
- **GetWriteLock** — **reserviert E Lock (exclusive),**
- **PromoteReadtoWrite** — **bewirkt Zustandswechsel S → E,**
- **Unlock** — **gibt Lock frei.**

Mehrere Transaktionen können ein S Lock für den gleichen Datenbereich (z.B. einen Datensatz) besitzen. Nur eine Transaktion kann ein E Lock für einen gegebenen Datenbereich besitzen. Wenn eine Transaktion ein S Lock in ein E Lock umwandelt, müssen alle anderen Besitzer des gleichen S Locks benachrichtigt werden.

Normalerweise besitzt eine Transaktion mehrere Locks.

In einer Two-Phase Locking Transaktion finden alle Lock Aktionen zeitlich vor allen Unlock Aktionen statt. Eine Two-Phase Transaktion hat eine Wachstumsphase (growing), während der die Locks angefordert werden, und eine Schrumpf (shrink) Phase, in der die Locks wieder freigegeben werden.

11.2.8 Locking Protokoll

Mehrere Transaktionen können das gleiche Lock im Zustand S besitzen. Nur eine Transaktion kann ein Lock im Zustand E besitzen.

Eine Transaktion kann ein Lock vom Zustand S in den Zustand E überführen. Hierzu ist es erforderlich, dass eine Nachricht an alle anderen Transaktionen geschickt wird, die das gleiche Lock im Zustand S besitzen. Mittels dieser Nachricht wird das S Lock für ungültig erklärt.

derzeitiger Status / Anforderung	kein	Lesen shared	Schreiben exclusive
Lesen Shared	bewilligt, share-mode	bewilligt, share-mode	abgelehnt, Mitteilung über Besitzer
Schreiben Exclusive	bewilligt, exclusive mode	bewilligt, Warnung über Besitzer	abgelehnt, Mitteilung über Besitzer

Abb. 11.2.8
Bewilligen von Locks

Vorgehensweise:

- **Shared Lock (S) erwerben vor dem erstmaligen Lesen**
- **Exclusive Lock (E) erwerben vor dem erstmaligen Schreiben**

In Abb. 11.2.6 informiert das Senden einer Nachricht an Transaktion 2 diese, dass ihr Wissen über den Zustand der beiden Variablen d1 und d2 nicht mehr gültig ist.

Transaktion 2 muss sich neu über den Zustand der Variablen d1 informieren, ehe sie ein Update von d1 vornimmt.

Frage: Woher weiß Transaktion 1, dass Transaktion 2 ein Interesse an der Variablen d1 hat (ein shared Lock besitzt) ?

11.3 Lock Strukturen

11.3.1 Lock-Verwaltung

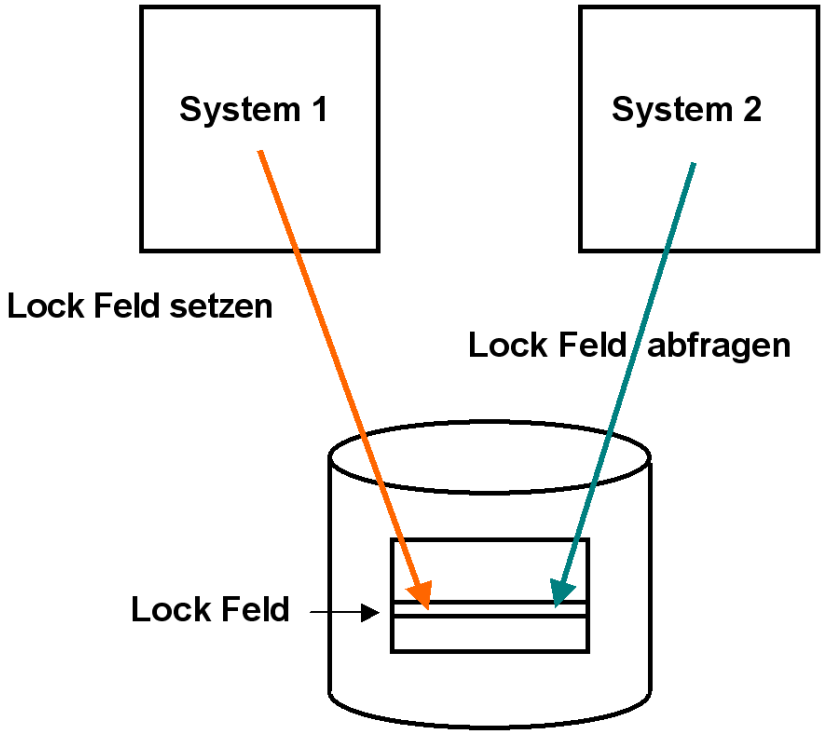


Abb. 11.3.1
Lock Speicherung auf der Festplatte bewirkt Performance Probleme

Frage: Wie speichert man die Locks?

Einfachste Lösung: Locks mit den Daten auf dem Plattenspeicher speichern.

Jeder zu schützende Datenbereich auf der Festplatte (z. B. eine Zeile in einer relationalen Datenbank-Tabelle) erhält ein zusätzliches Lock-Feld.

Bei einem Zugriff wird das Lock zunächst geprüft und dann gesetzt, ehe ein Zugriff erfolgt.

Nachteil: Die erforderlichen zusätzlichen Zugriffe auf den Plattenspeicher sind in Hochleistungssystemen nicht akzeptabel.

11.3.2 Verteilte Lock-Tabelle

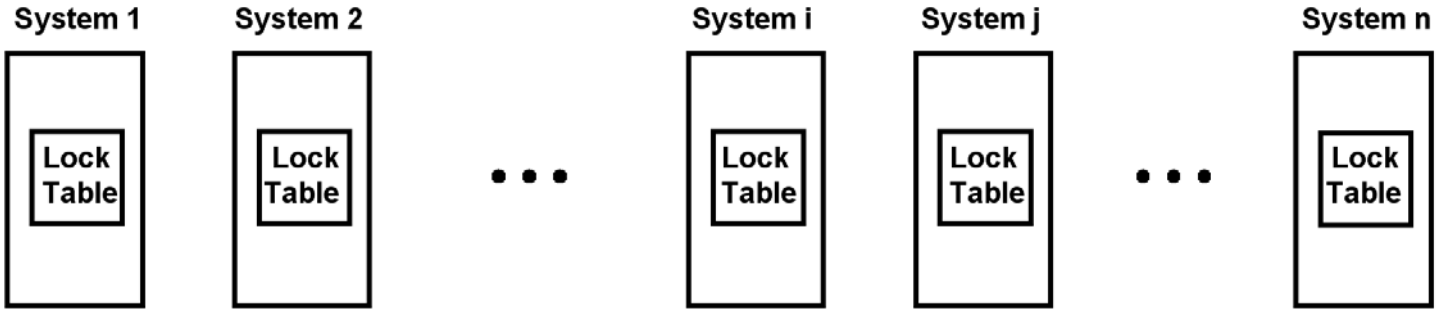


Abb. 11.3.2
Lock Tabellen in den Hauptspeichern der einzelnen Systeme müssen synchronisiert werden

Zugriffe auf Datenbank-Tabellen mit Locks erfolgen recht häufig. Deswegen werden Lock-Tabellen in der Regel im Hauptspeicher gehalten. Bei einem SMP mit nur einem Hauptspeicher ist dies unkritisch.

Bei einem Cluster mit mehreren SMPs und mehreren Hauptspeichern sind auch mehrere Lock Tabellen in den Hauptspeichern der beteiligten Systeme vorhanden, die alle auf dem gleichen Stand gehalten werden müssen (verteilte Lock-Tabelle).

Zur Auflösung von Lock-Konflikten erfolgt bei jeder Änderung in einer Lock-Tabelle entweder ein Broadcast (Invalidate-Broadcast-Kohärenzsteuerung) oder eine gezielte Nachricht von System i an System j.

Ersteres erfordert, in jedem System des Clusters die Verarbeitung der dort laufenden Transaktion auszusetzen, um ein Update der Lock-Tabelle vorzunehmen. Dies erfordert einen Prozesswechsel. Der Overhead kann 20 ms CPU Verarbeitungszeit erfordern, auch dann, wenn das System an dem Zustand des Locks überhaupt nicht interessiert ist.

Beispiele für diese Lösung sind die VAX DBMS und VAX Rdb/VMS Datenbanksysteme.

Gray/Reuter p.380

11.3.3 Invalidate-Broadcast-Kohärenzsteuerung

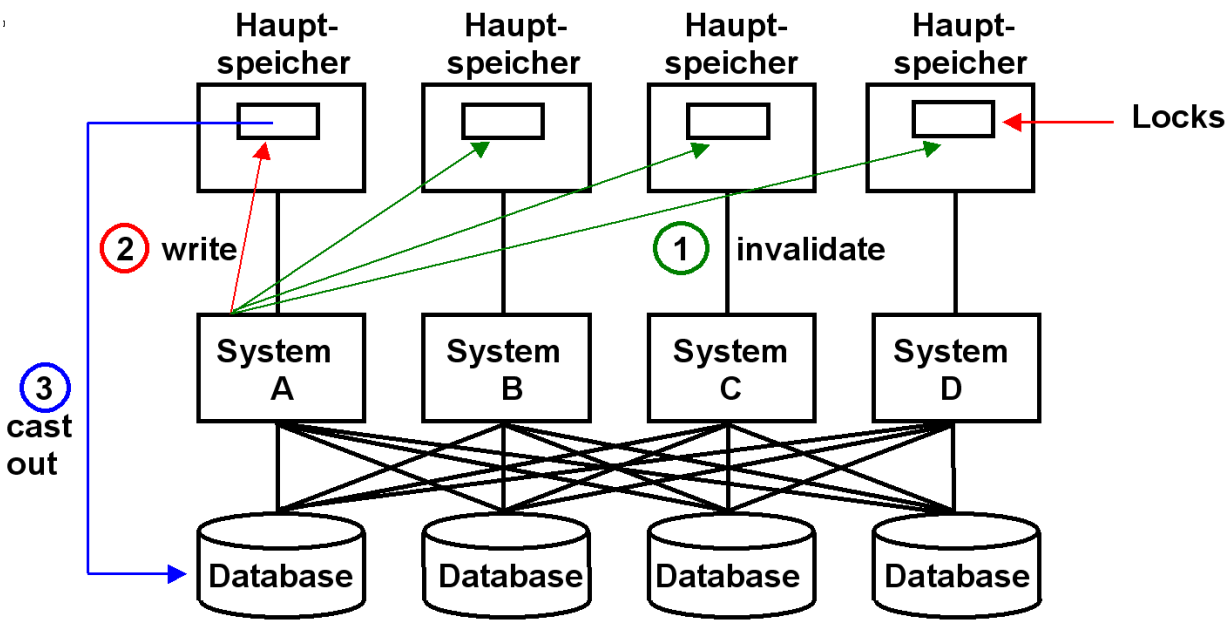


Abb. 11.3.3
System A sendet eine Invalidate Lock Nachricht an alle anderen Systeme

Systeme A, B, C und D besitzen ein Read (S) Lock. System A möchte das Read Lock in ein Write (E) Lock umwandeln. Eine Invalidate-Broadcast-Nachricht an alle Knoten des Clusters benachrichtigt B, C und D, dass die Kopie des Locks nicht mehr gültig ist. Dies ist wegen des TCP/IP Overheads aufwendig, weil auch diejenigen Knoten die Nachricht erhalten und verarbeiten müssen, die an diesem spezifischen Lock gar nicht interessiert sind.

Eine Faustregel besagt: You cannot build a cluster that scales, if you do not solve the locking problem. Jim Gray, Andreas Reuter, 1993.

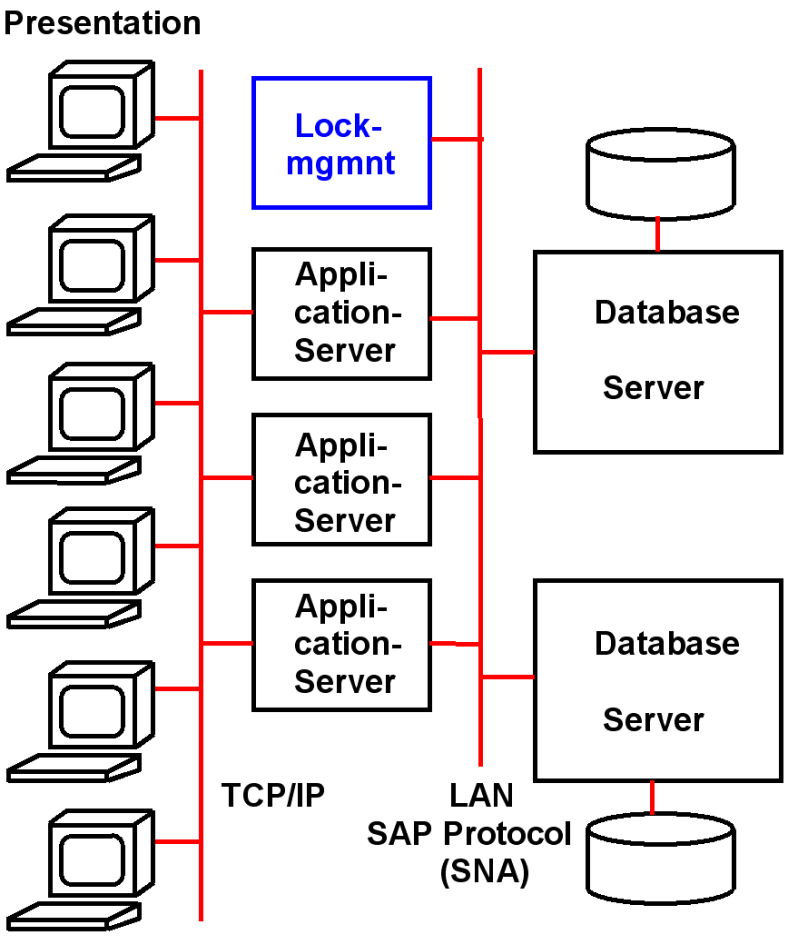


Abb. 11.3.4
Typische SAP/R3 configuration

Dezentrale Locks arbeiten mit einer Invalidate-Broadcast-Kohärenzsteuerung, die bei großen Transaktionsraten schlecht skaliert. Dieses Problem kann mit Hilfe einer zentralisierten Lockverwaltung und einem zentralen Sperrverwaltungsserver adressiert werden.

Eine typische SAP/R3 Konfiguration verwendet einen getrennten Server für das Lock Management (Sperr-Server). Wenn ein Knoten ein S Lock erwerben möchte, wendet er sich an den Sperr-Server um zu erfragen, ob ein anderer Knoten ein Interesse an diesem Lock hat. Wenn ja, wird nur dieser benachrichtigt.

Nachteilig: Die Anfrage an den Sperr-Server geht über ein LAN mit dem entsprechenden TCP/IP Stack Overhead.

Zur Information: Für die interne Kommunikation verwendet SAP eine Version des SNA-Protokolls, spezifisch auch de CPI-C Remote Procedure Call.

11.3.4 Locking mit der Coupling Facility

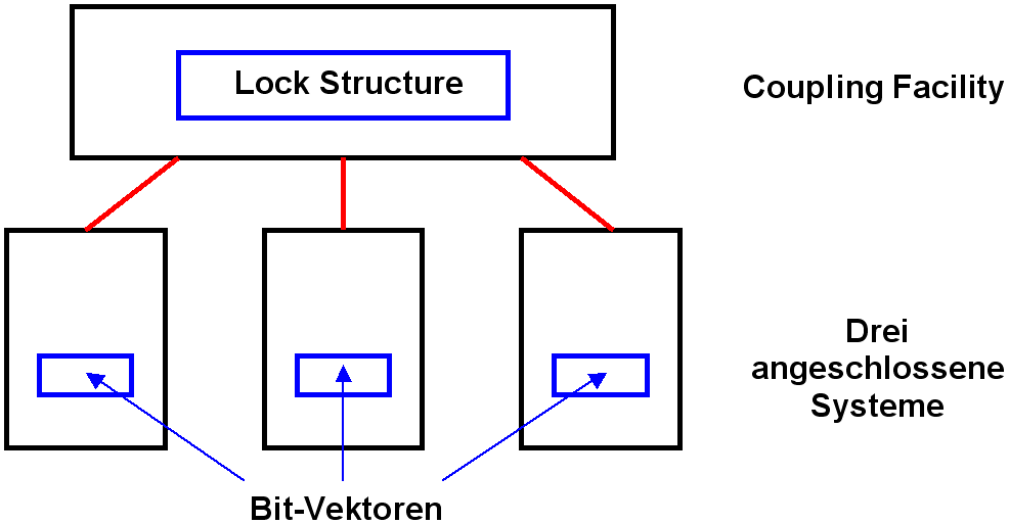


Abb. 11.3.5
Eine teilweise Kopie der CF Lockstruktur ist in den angeschlossenen Systemen enthalten

Das globale Lock Management) im Sysplex erfolgt in Zusammenarbeit der Coupling Facility (CF), der z/OS-Betriebssysteme sowie der auf jedem System laufenden Subsysteme (z.B. CICS, DB2), die eine globale Synchronisation benötigen.

Hierzu dient eine globale Locktabelle (Lock Structure) und lokale Bit-Vektoren. Die globale Lock-Tabelle ist in der Coupling Facility untergebracht und enthält einen Eintrag für jedes zu verarbeitende Lock. Jedes an die Coupling Facility angeschlossene System enthält eine lokale Lock-Tabelle, die als Bit-Vektor bezeichnet wird. Alle Bit-Vektoren gemeinsam enthalten eine Untermenge der in der CF Lock Structure enthaltenen Information.

Änderungen in der Lock Structure werden unmittelbar von der Coupling Facility in die betroffenen Bit Vektoren kopiert. Dies geschieht automatisch, ohne Kenntnisnahme durch das betroffene System.

Wie in Abb. 11.3.6 gezeigt, läuft auf jedem Rechner ein lokaler Lock Manager (LLM), der für die Lockanforderungen aller Transaktionen und Prozesse des jeweiligen Knotens zuständig ist.

In der CF-Lock-Struktur wird eine zentrale Locktabelle verwaltet, auf die LLMs aller Knoten zugreifen, um eine schnelle systemweite Synchronisation konkurrierender Datenzugriffe zu erreichen. Diese Zugriffe erfolgen synchron über die Coupling Links und eine als „Cross System Extension Services" (XES) bezeichnete z/OS-Komponente, die u.a. spezielle Maschinenbefehle zum Setzen und Freigeben von globalen Sperren in der CF-Sperrtabelle verwendet.

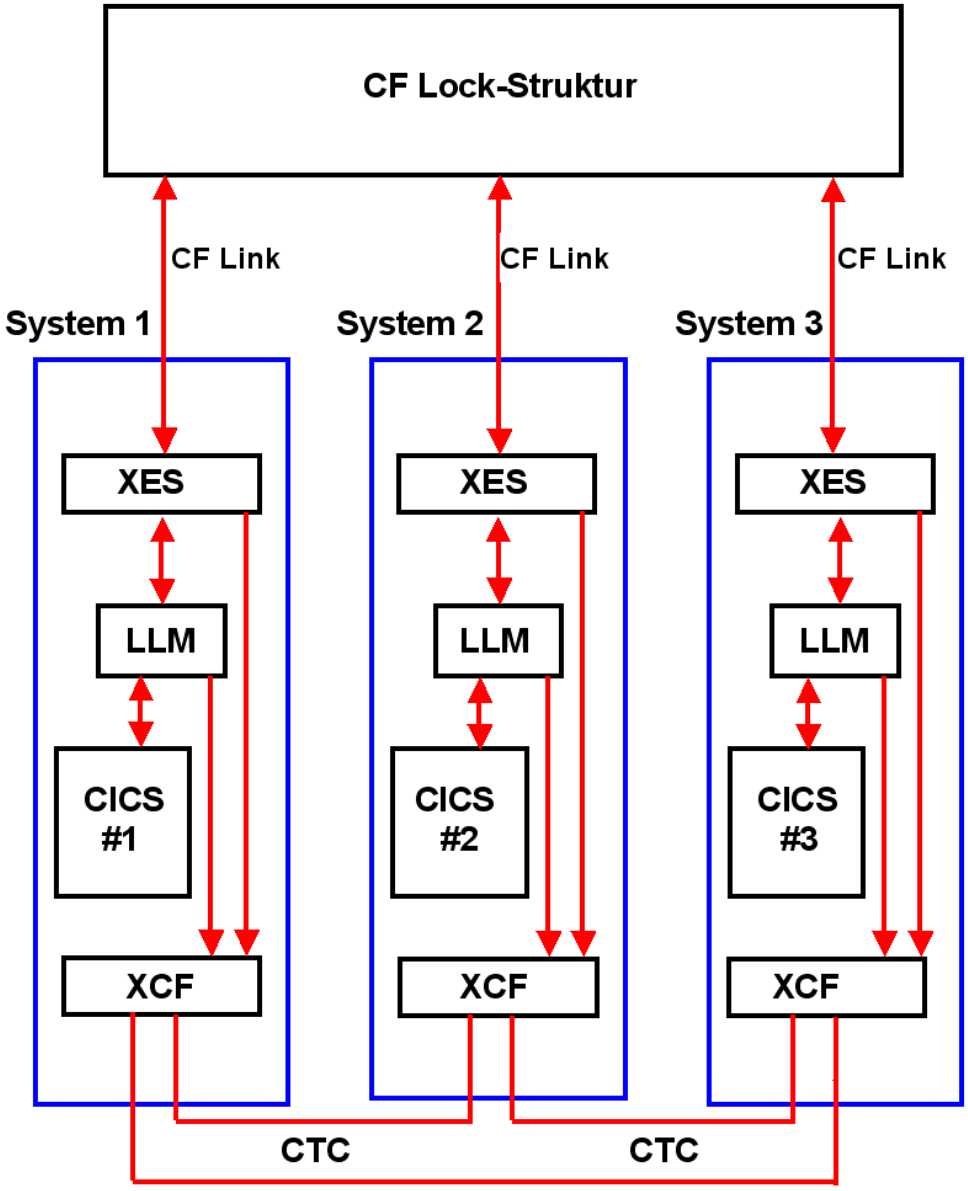


Abb. 11.3.6
Lokaler und globaler Lock Manager

Wenn ein Lock Anforderung auftritt, sind drei Alternativen möglich:

1. **Die Auflösung der Lock-Anforderung kann mittels der Information im lokalen Bit-Vektor und evtl. eines Lesezugriffs auf die Lock-Struktur der CF bewältigt werden.**
2. **Die Auflösung der Lock-Anforderung erfordert einen Schreibzugriff auf die Lock-Struktur der CF. Die CF bewirkt daraufhin ein Update der Bit-Vektoren in den anderen Knoten (Systemen) des Sysplex. Dies erfolgt in den anderen Knoten von diesen unbemerkt, ohne dass die dort laufenden Prozesse deshalb unterbrochen werden müssen.**
3. **Die Auflösung eines Lock-Konfliktes erfordert auf einem (oder mehreren) betroffenen Knoten des Sysplex eine Aktion. In diesem Fall handelt der auslösende Rechner und der betroffene Rechner die erforderlichen Aktionen mittels einer Kommunikation über die CTC- und XCF-Verbindung direkt aus.**

Wichtig ist, dass die große Mehrzahl aller Lock-Anforderungen durch die Alternativen 1 und 2 abgedeckt werden, und kein Aufwand für die Unterbrechung eines laufenden Prozesses erforderlich ist. Alle konfliktfreien Lockanforderungen, die typischerweise über 99% aller Locks betreffen, können über die zentrale Sperrtabelle (Lock Struktur) mit minimaler Verzögerung behandelt werden.

Nur solche Sperranforderungen, für die zwischen verschiedenen Rechner ein Konflikt auftritt, erfordern die Synchronisierung zwischen den betroffenen Rechnern. Dies bewirkt ein globaler System Lock Manager (SLM).
Typischerweise sind lediglich 2 Systeme von dem Konflikt betroffen. Hierzu erfolgt ein Nachrichtenaustausch über das erwähnte CTC-Protokoll sowie über die z/OS-Komponente XCF.

Auf jedem System (Knoten) befindet sich ein lokaler Lock-Manager (LLM), der für die Sperrbehandlung aller seiner Prozesse (CICS in dem gezeigten Beispiel) zuständig ist.

Die XES (Cross System Extension Services) Komponente des z/OS Kernels ist für die Verbindung zur Lock Struktur in der Coupling Facility zuständig. XES benutzt hierfür die Coupling Link Prozessoren.

Zusätzlich können die Systeme (Knoten) des Sysplex über das CTC Protokoll und die XCF Komponente des z/OS Kernels paarweise (normalerweise über den FICON Switch) direkt miteinander kommunizieren.

11.3.5 Hash-Klasse

Locks können mit einer unterschiedlichen Granularität ausgestattet sein. Ein Lock kann sich z.B. auf eine Reihe in einer SQL-Tabelle, einen Teil einer SQL-Tabelle, eine ganze SQL-Tabelle, mehrere SQL-Tabellen, einen ganzen Plattenstapel usw. beziehen. Alle Locks haben einen eindeutigen Namen. Der Name könnte z.B. in einem Feld in einer Reihe in einer SQL-Tabelle untergebracht sein.

Die Lock-Struktur der CF enthält eine globale Tabelle. Jedes aktive Lock ist durch einen Eintrag in der Lock-Struktur Tabelle gekennzeichnet.

Frage: Wie wird einem bestimmten Lock-Namen ein eindeutiger Eintrag in der Lock-Tabelle zugeordnet?

Das Problem ist, dass Lock-Namen sehr lang sein können. Zum Beispiel haben Locks in IMS-Datenbanken bis zu 19 Bytes (152 Bits) lange Namen. Da eine Locktabelle mit 2^{152} Einträgen für den Hauptspeicher viel zu groß (und sehr dünn belegt) wäre, erfolgt eine Abbildung in eine Hash-Klassen-Bezeichnung von z.B. 20 Bit. Der Adressraum für die Lock-Einträge kann somit über eine CF-Locktabelle mit $n = 2^{20}$ = 1 048 576 Einträgen abgedeckt werden.

Jeder Knoten (System) des Sysplex benutzt den gleichen Hash-Algorithmus, um den bis zu 152 Bit langen Namen in einen 20 Bit Hash-Wert zu übersetzen.

Da die n Einträge Hash-Klassen repräsentieren, können „unechte Konflikte“ auftreten, wenn zwei Locknamen zufällig in dieselbe Hash-Klasse abgebildet werden. Die Häufigkeit solcher Kollisionen ist eine Funktion der Locktabellengröße. Diese kann vom Systemprogrammierer dynamisch verändert werden, um die Menge der falschen Konflikte zu minimieren. Eine Faustregel besagt, dass nicht mehr als 1% der Lockanforderungen zu Konflikten führen sollte.

Ein einfaches Beispiel: Der Lock-Name hat eine Länge von 256 Bit

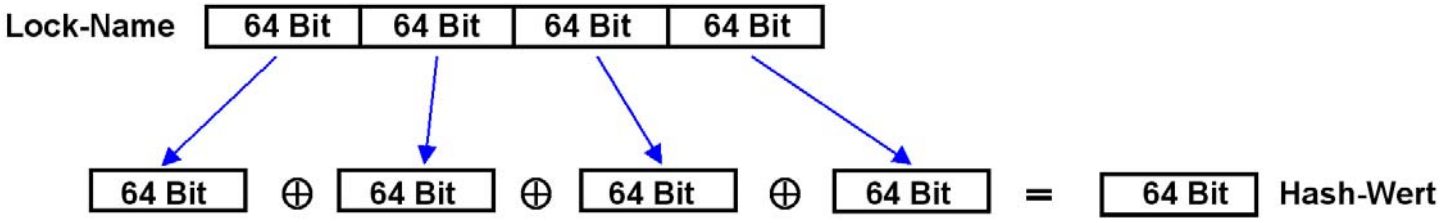


Abb. 11.3.7
Hashing Beispiel

Vorgehensweise **Name (hier 256 Bit) in 4 Teile gleicher Länge (hier 64 Bit) zerlegen. Teile mit Exclusive Oder (Symbol ⊕) verknüpfen. Das Ergebnis ist ein 64 Bit Hash-Wert.**

Problem: **Es kann sein, dass zwei unterschiedliche Namen den gleichen Hash-Wert ergeben (Hash-Synonym oder Hash-Konflikt). Das Arbeiten mit Hash-Werten braucht ein Verfahren, um Hash-Konflikte aufzulösen.**

Erwünscht:

Der Hash-Algorithmus (hier eine einfache Exclusive Oder-Verknüpfung) soll sicherstellen, dass für beliebige Nachrichten alle Bitmuster des Hashwertes mit möglichst gleicher Wahrscheinlichkeit vorkommen. Es existiert eine sehr große Auswahl an unterschiedlichen Hash-Algorithmen (Außerdem muss der Hash-Algorithmus performant in Hardware und/oder Software – je nach Einsatzzweck - implementierbar sein).

Wie groß muss die CF Lock-Tabelle sein ?

Unechte Konflikte treten auf, wenn zwei Locknamen in dieselbe Hash-Klasse abgebildet werden. Die Anzahl der unechten Konflikte sollte möglichst klein gehalten werden. Wie groß muss die Lock-Tabelle konfiguriert werden?

Schauen wir uns ein Beispiel an.

Nehmen wir einen Sysplex an, der 10 000 Transaktionen pro Sekunde verarbeitet, mit einer durchschnittlichen Verarbeitungsdauer (Antwortzeit) von 0,5 Sekunden pro Transaktion. In jedem Augenblick verarbeitet der Sysplex 5 000 Transaktionen gleichzeitig. Nehmen wir gleichzeitig an, jede Transaktion benötigt durchschnittlich 20 Locks. Die durchschnittliche Anzahl aller aktiven Locks beträgt damit 100 000.

Wenn die Lock-Tabelle 20 000 000 Einträge hat, beträgt die Wahrscheinlichkeit etwa 0,5 %, dass 2 Locks zufällig in der gleichen Hash-Klasse abgebildet werden. 16 777 216 Einträge erfordern eine Hasch Wert mit einer Länge von 24 Bit.

11.3.6 Lock-Management

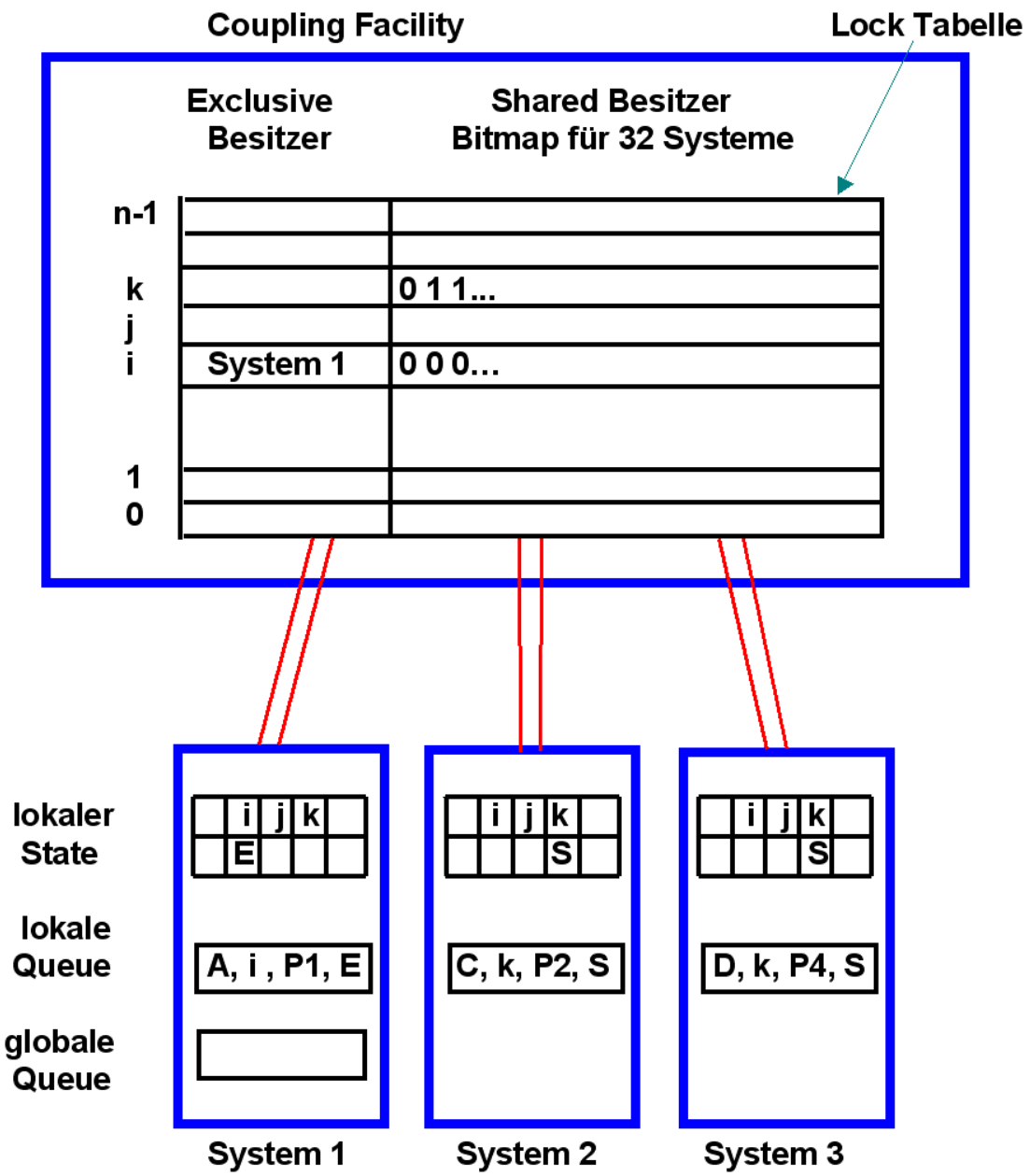


Abb. 11.3.8
Nutzung der CF Lock Tabelle

Abb. 11.3.8 verdeutlicht für eine Beispielkonfiguration mit drei Systemen (Knoten) den Aufbau der globalen Locktabelle in der CF sowie der lokalen Locktabellen der LLMs.

Der Lock-Zustand eines Data Items kann 3 Werte haben:

Frei	0
Shared	S = Shared
Exclusive	E = Exclusive

Die zentrale Lockinformation wird nicht auf Ebene einzelner Transaktionen, sondern nur für ganze Systeme geführt. Das Lockprotokoll unterscheidet dabei wie üblich Lese-Locks (Shared, S), welche pro Objekt gleichzeitig an mehrere Systeme gewährt werden können, und exklusive Schreiblocks (E). Die CF-Locktabelle (oberer Teil in der Abbildung) verwendet hierzu pro Lockeintrag (Hash-Klasse) zwei Felder: Für den Fall eines exklusiv gesetzten Locks (EXC) enthält das erste Feld die Knoten-Adresse des Besitzers.

Das zweite Feld enthält einen Bit-Vektor mit 32 Bit-Länge. Jedes Bit dieses Vektors zeigt an, welcher der (maximal möglichen) 32 Knoten (Systeme) eines Sysplex ein Lese-Lock erworben hat (Shared, S = 1) bzw. ob kein solches vorliegt (S = 0). Das Beispiel weist somit aus, dass System 1 exklusiver Besitzer von allen Objekten der Hash-Klasse i ist und dass Systeme 2 und 3 Leseberechtigungen für alle Objekte der Hash-Klasse k besitzen.

11.3.7 Nutzung der CF Lock-Tabelle

Wie der untere Teil von Abb. 11.3.8 zeigt, bestehen die lokalen Sperrtabellen der LLMs aus drei Teilen:

1. **lokale Sperrzustände (Local Lock State),**
2. **objektspezifische Sperranforderungen lokaler Transaktionen (Local Queue) und**
3. **globale Warteschlangen mit objektspezifischen Sperranforderungen mehrerer Rechner (Global Queue).**

Im lokalen Sperrstatus führt der LLM seine Kenntnis vom Zustand der CF-Sperreinträge bezüglich der Hash-Klassen: "0" bedeutet keine Kenntnis (Eintrag wird von keinem anderen Knoten benutzt), "S" (Shared-Kenntnis, LLM hat Leserecht), "E" (exklusive Nutzung durch den eigenen Knoten) oder "Gx" (ein anderer Knoten x besitzt exklusive Rechte und kontrolliert die Sperrvergabe im Sysplex).

Im Beispiel weisen die lokalen Einträge entsprechend der CF-Sperreinträge System 1 als exklusiven Besitzer der Hash-Klasse i und Systeme 2 und 3 als leseberechtigt für Hash-Klasse k aus. Der zweite Bereich der lokalen Sperrtabellen enthält Informationen zu den spezifischen Objekten und lokal laufenden Transaktionen bzw. Prozessen, welche Sperren besitzen oder darauf warten. So bedeutet der Eintrag "(A, i, P1, E)" in System 1, dass der lokale Prozess P1 die Sperre zu Objekt A (Lock mit dem Namen A), welche zur Hash-Klasse i gehört (A hashes auf i), im exklusivem Modus besitzt. Der dritte Bereich (Global Queue) enthält nur relevant für Objekte, an denen ein systemübergreifender Sperrkonflikt in der CF erkannt wurde und der LLM zur Konfliktauflösung mit anderen LLMs zusammenarbeiten muss. Hierzu werden in dem Sperrbereich die Sperranforderungen der anderen Rechner abgelegt, um diese global synchronisiert abzuarbeiten.

Der (symbolische) Name A eines Locks wird mit Hilfe eines Hashing-Algorithmus in die Hash-Klasse i abgebildet. Die Locking-Tabelle enthält für jede Hash-Klasse einen Eintrag.

Die Zuordnung Lock-Name zu Hash-Klasse erfolgt in der lokalen Queue des betreffenden Systems.

1. **Prozess P1 in System 1 möchte EXC-Rechte für ein Lock in der Hash-Klasse i erhalten. Anfrage an CF. Da niemand sonst Interesse hat, wird dem Request entsprochen. Im lokalen State Vektor von System 1 wird diese Berechtigung festgehalten.**

   In der lokalen Queue von System 1 wird festgehalten, dass Lock A, Hash-Klasse i von dem lokalen Prozess P1 mit der Berechtigung Exclusive gehalten wird.

   Wenn Prozess P2 in System 1 ebenfalls Lock-Rechte für i wünscht (möglicherweise für einen anderen Lock-Namen), ist kein Zugriff auf die CF erforderlich. System 1 kann dies alleine aussortieren.

2. **Sowohl System 2 als auch System 3 wünschen für ihre jeweiligen Prozesse P2 und P4 Shared Rechte für Locks C und D, die beide in die Hash-Klasse k fallen. Die CF registriert dies in der Bitmap für k und erteilt die Rechte.**

Wenn jetzt System 1 Exclusive Rechte für ein Lock der Hash-Klasse k will, erhält es von der CF die Bitmap der Klasse k zurück. System 1 hat jetzt die Aufgabe, weitere Maßnahmen mit den betroffenen Systemen 2 und 3 (und nur diesen, nicht aber den potentiell 29 weiteren Systemen) direkt auszuhandeln.

11.4 Cache- und Listen-Strukturen

11.4.1 „Eager“ und „Lazy“ Locking-Protokolle

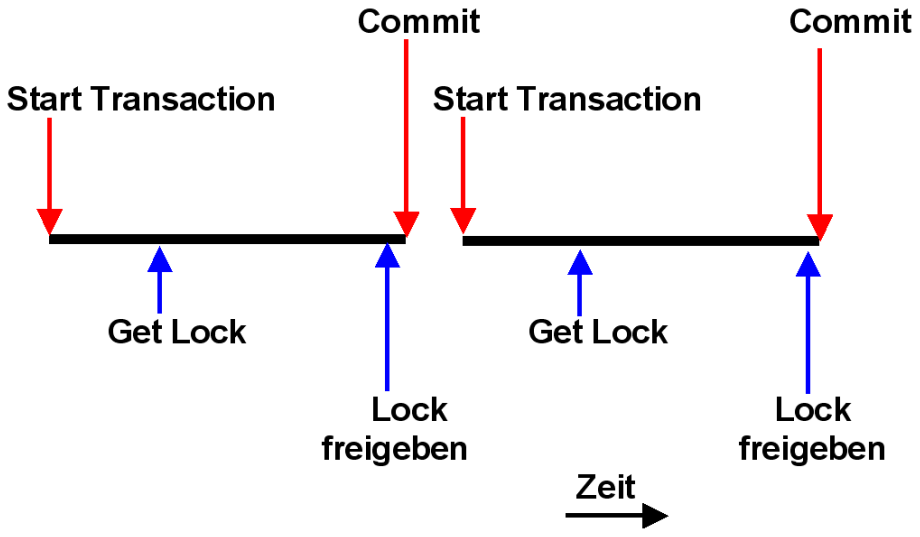


Abb. 11.4.1
Eager-Protokoll

Eager-Protokoll	**Lock freigeben, wenn Commit die Transaction beendet (siehe Abb. 11.4.1**
Lazy-Protokoll	**Lock freigeben, wenn eine Contention auftritt; nichts tun, bis jemand anderes das Lock benötigt.**

Das Lazy-Protokoll arbeitet besser, wenn Datenkonflikte selten auftreten. Ein Beispiel ist das TCP-C Benchmark des Transaction Processing Councils (www.tpc.org). Hersteller von Datenbank-Software benutzen TPC-C gerne, weil dann ihr Produkt besser aussieht. Die Unterschiede zu praktischen Anwendungen können allerdings sehr groß sein.

Die Sysplex Coupling Facility verwendet das Eager-Protokoll (auch als „force-at-commit“ bezeichnet). Datenkonflikte treten häufig auf, wenn existierende Anwendungen auf den Sysplex portiert werden.

11.4.2 Ein/Ausgabepuffer

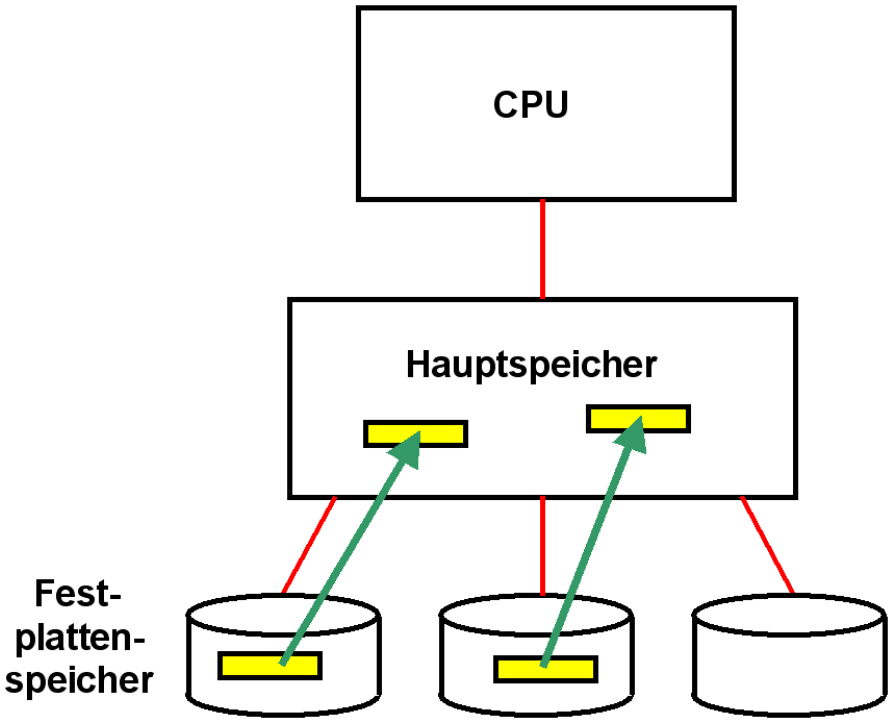


Abb. 11.4.2
Die Menge der Ein/Ausgabe Puffer wird zu einem Buffer Pool zusammengefasst.

In der Vergangenheit hat ein Anwendungsprogramm jeweils einzelne Datensätze (Records) vom Plattenspeicher in einen Ein/Ausgabepuffer (I/O Buffer) im Hauptspeicher gelesen und dort verarbeitet. Zur Leistungssteigerung hat man bald mehrere logische Records zu einem physischen Record zusammengefasst. Mit etwas Glück findet das Anwendungsprogramm beim nächsten Zugriff die Daten bereits im Ein/Ausgabepuffer und ein Plattenspeicherzugriff erübrigt sich.

In der Regel wird mit mehreren Dateien oder Datenbanken gleichzeitig gearbeitet, die alle ihren eigenen Ein/Ausgabepuffer benutzen. Die Menge der Ein/Ausgabepuffer wird als „Buffer Pool“ bezeichnet und vom Datenbank-System optimal verwaltet.

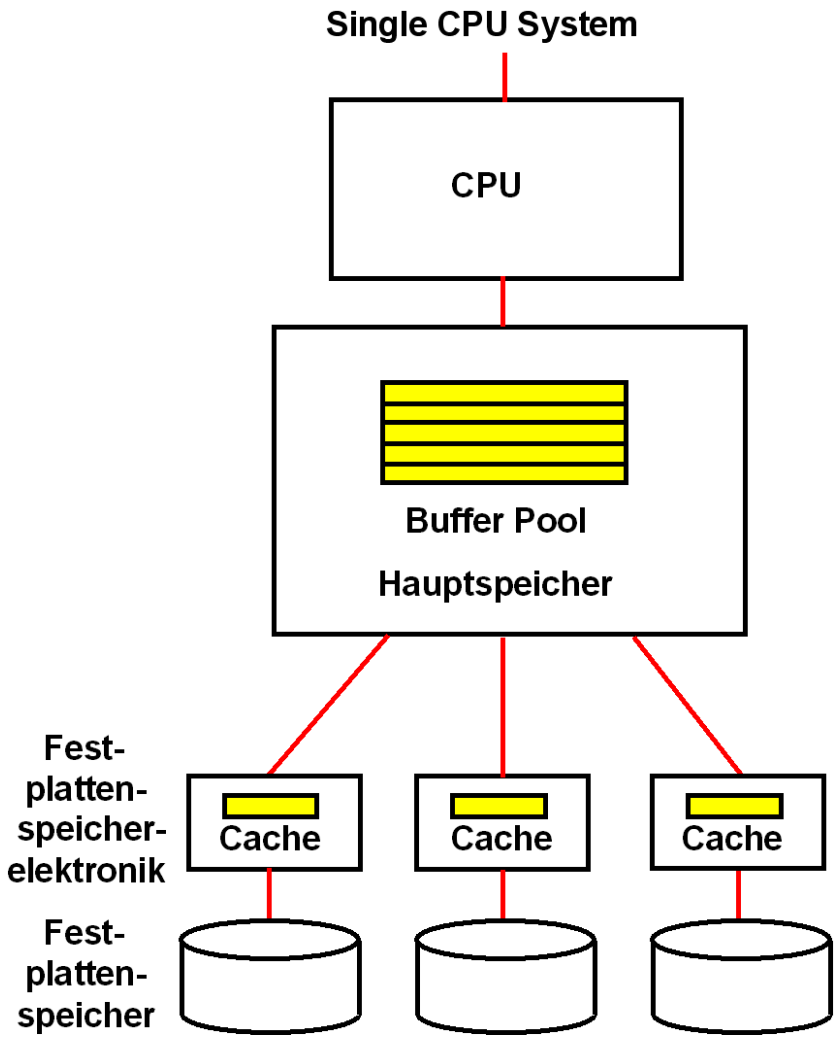


Abb. 11.4.3
Festplattenspeicher Cache und Hauptspeicher Buffer Pool

Der Buffer Pool stellt eine Art Plattenspeicher Cache im Hauptspeicher dar. Das Datenbank-System bemüht sich, den Speicherplatz im Buffer Pool optimal zu verwalten. (Unabhängig davon werden Daten zusätzlich in einem Plattenspeicher Cache gespeichert, der Bestandteil der Plattenspeicher Elektronik oder des Enterprise Storage Servers ist).

Der Buffer Pool besteht aus einzelnen Puffern (Buffers), die Datenbankobjekte oder Teile einer Datei aufnehmen.

In einem Cluster ist nicht auszuschließen, dass Datensätze oder Datenbank Records gleichzeitig in den Buffer Pools mehrerer Knoten (Systeme) abgespeichert werden.

Der Datentransfer zwischen dem Festplattenspeicher und dem Bufferpool erfolgt in 4 KByte-Blöcken (auch als „Slots“ bezeichnet).

11.4.3 Coupling Facility Cache Directory

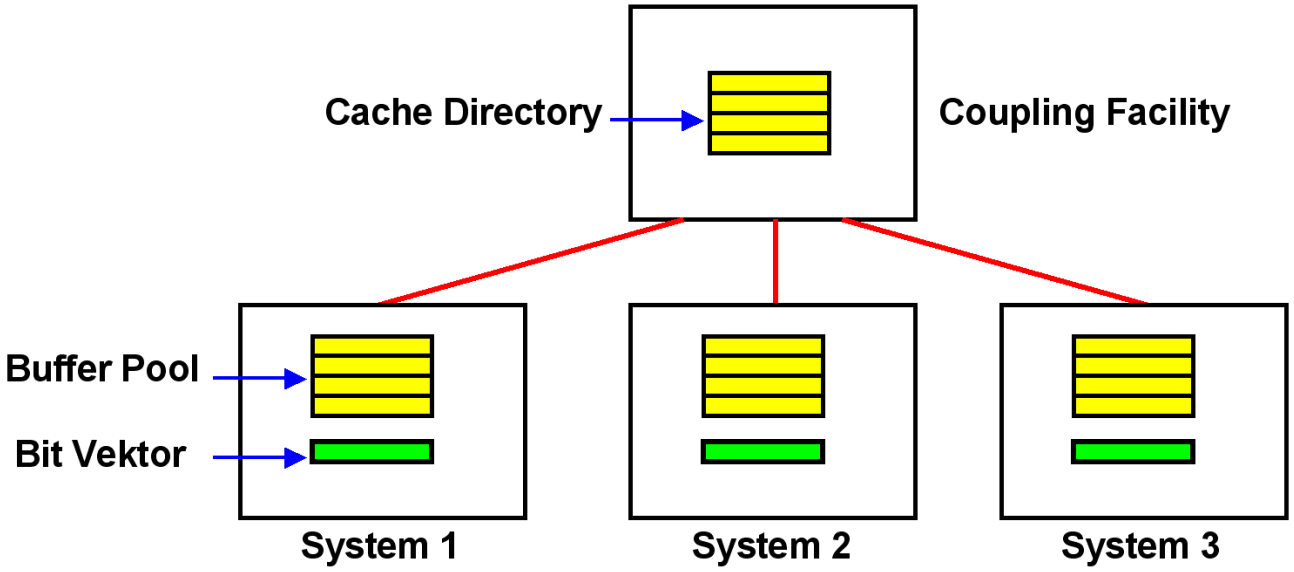


Abb. 11.4.
Cache Directory in der Coupling Facility

Der Buffer Pool in jedem System enthält Blöcke (Buffer), die möglicherweise gerade bearbeitet werden. Es kann sein, dass sich in zwei unterschiedlichen Systemen Buffer mit den gleichen Datenbankrekords befinden.

Die Coupling Facility unterhält ein "Cache Directory", in dem sich jeweils ein Eintrag für jeden Buffer in den angeschlossenen Systemen befindet. Analog zur Lock-Verwaltung befinden sich außerdem in jedem System Bit-Vektoren, die den Inhalt des Cache Directory teilweise replizieren.

Bei einer Änderung eines Eintrags im Cache Directory erfolgt ein automatisches Update der Bit-Vektoren in allen angeschlossenen Systemen.

Der lokale Buffer Pool im System 1 enthält Puffer (Blöcke) mit Records, die gerade bearbeitet werden. Solange die Transaktion nicht abgeschlossen ist, verhindert der Lock-Manager einen Zugriff durch ein anderes System (z.B. System 2).

Wenn die Transaktion abgeschlossen ist (commit), werden die Locks freigegeben. Die Puffer bleiben in System 1 erhalten; evtl. werden sie demnächst wieder gebraucht.

Greift System 2 jetzt auf einen Buffer mit dem gleichen Datenbankrekord zu, entsteht ein Kohärenzproblem. Die beiden Buffer in den Systemen 1 und 2 haben nicht den gleichen Inhalt.

Lösung: „Force-at-Commit". Bei Transaktionsabschluss erfolgt ein Update des Cache Directory durch System 1.

Die CF sendet hierzu eine „Cross-Invalidate" (CI) Nachricht an alle anderen betroffenen Systeme (und nur an die betroffenen Systeme)

Die Cross-Invalidate Nachricht ändert den lokalen State Vector innerhalb des Hauptspeichers eines jeden betroffenen Systems ab. Dies geschieht durch den Link Prozessor und verursacht keine CPU-Unterbrechung!

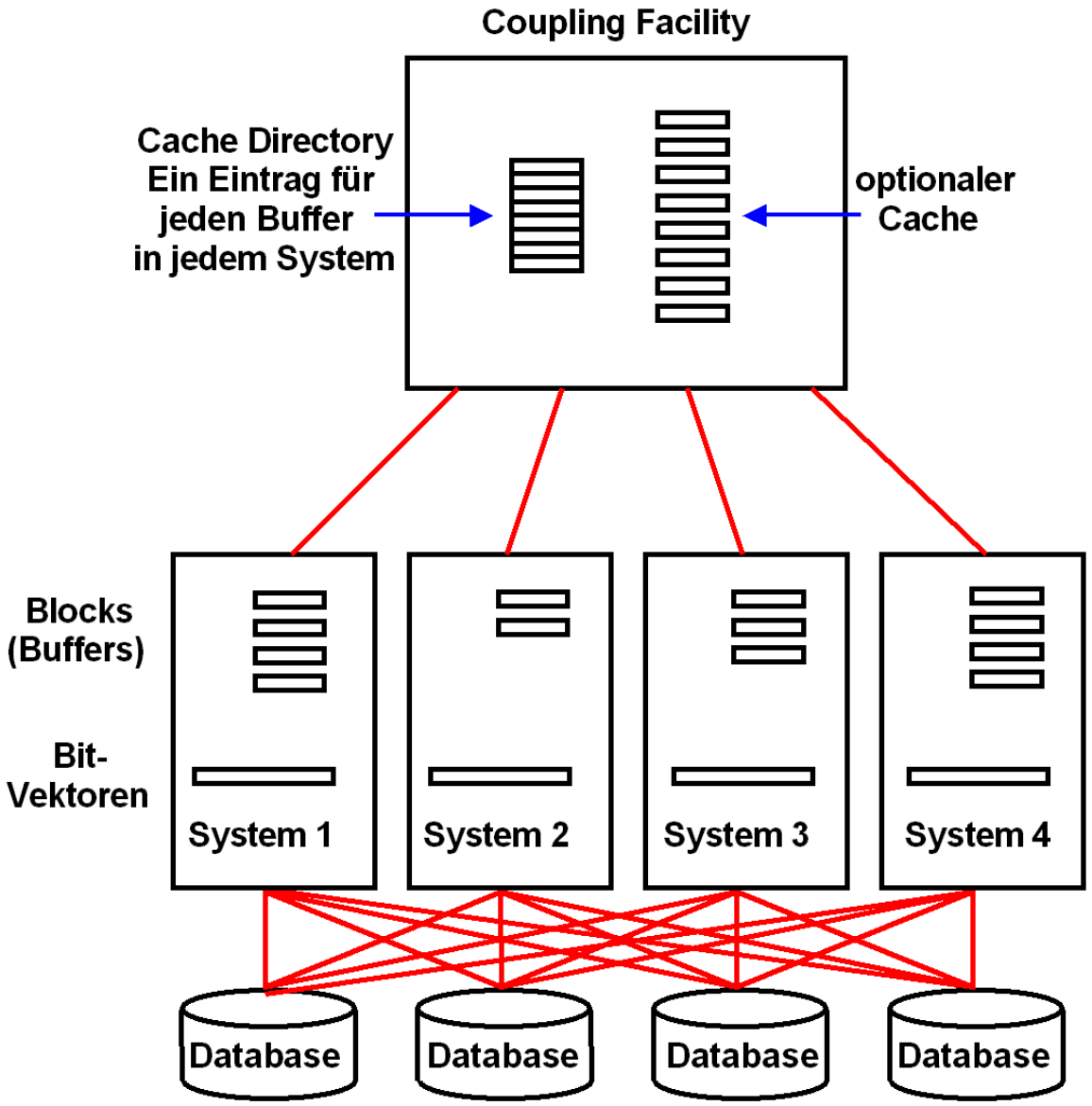


Abb. 11.4.4
Bit Vektoren kopieren teilweise die Information im CF Cache Directory

Das Cache Directory in der Coupling Facility enthält einen Eintrag für jeden Block (Buffer), der Teil eines Buffer Pools in einem der beteiligten Systeme ist.

a) System 1 Read from Disk
1. Load Block from Disk
2. Register with CF Directory
3. add Bit in Bit Vector

b) System 2 Read from Disk
1. Load Block from Disk
2. Register with CF Directory
3. add Bit in Bit Vector

einige Zeit später

c) System 1 Intend to write (to local Buffer)
1. Register with CF
2. CF invalidates all Bit Vectors
3. Write to local Buffer

d) System 2 Read from Buffer
1. Read
2. detects invalid Bit im localen Bit Vector
3. führt erforderliche Maßnahmen durch

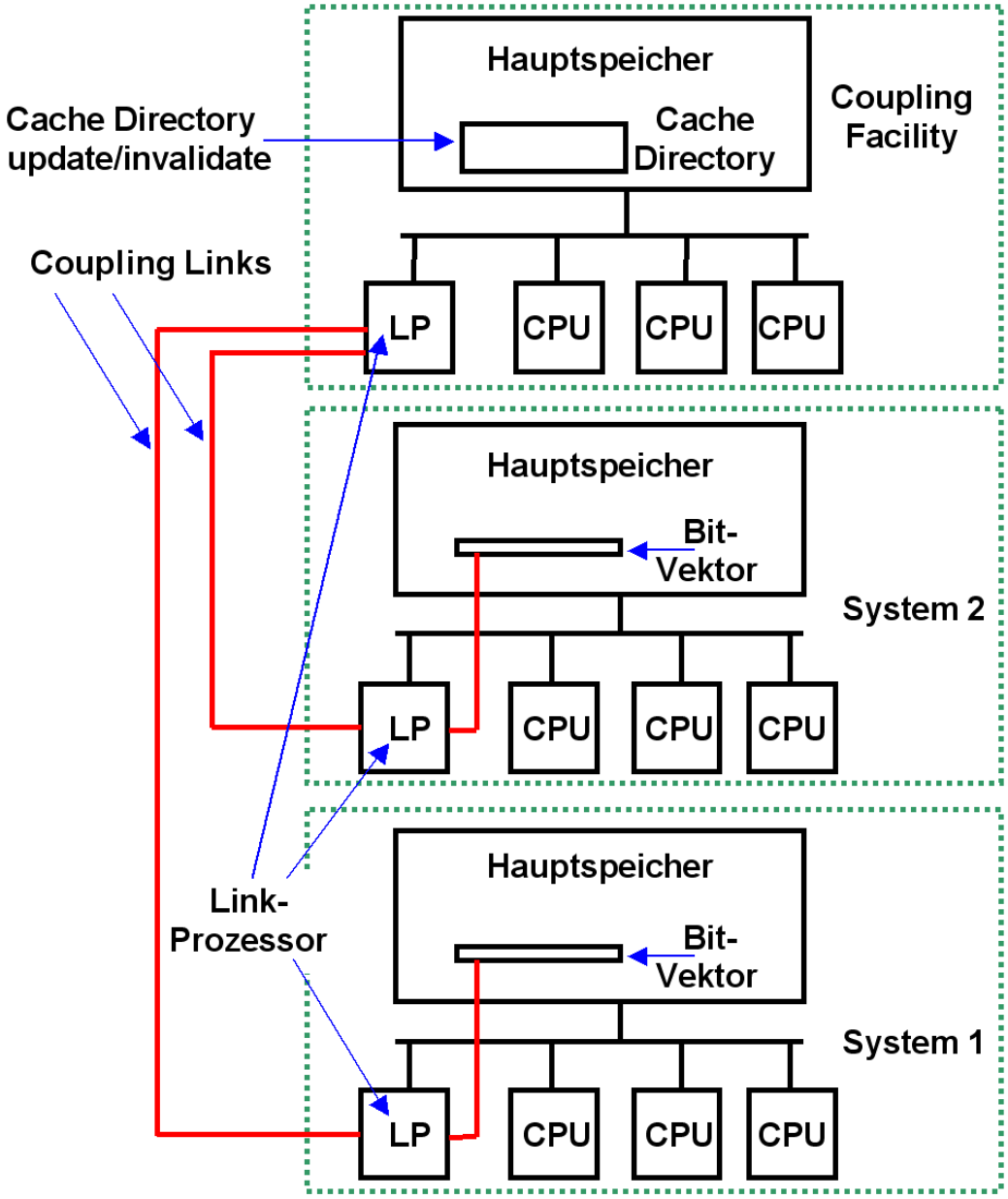


Abb. 11.4.5
Update der Bit Vektoren erfolgt unbemerkt von den beteiligten Systemen

Beim Force-at-Commit sendet die CF eine „Cross-Invalidate (CI) Nachricht an alle anderen Systeme.

Link-Prozessoren (LP) haben einen Direct Memory Access (DMA) zu dem Hauptspeicher.

Über die Link-Prozessoren der Coupling Facility und die Link-Prozessoren der Systeme können Bit-Vektoren im Hauptspeicher abgeändert werden, ohne dass der normale Programmablauf in den Systemen dadurch beeinflusst wird (kein Prozesswechsel). Dies verbessert das Leistungsverhalten, da jeder Prozesswechsel eine Pfadlänge von mehreren Tausend Maschinenbefehlen erfordert.

11.4.4 Coupling Facility Cache

Neben dem Cache Directory kann die Coupling Facility auch als Plattenspeicher-Cache genutzt werden. Hiervon machen einige, aber nicht alle Datenbanksysteme Gebrauch. DB2 und Adabas nutzen die Coupling Facility auch als Plattenspeicher-Cache, IMS jedoch nicht.

DB2, IMS und Adabas sind die wichtigsten unter z/OS eingesetzten Datenbanksysteme.

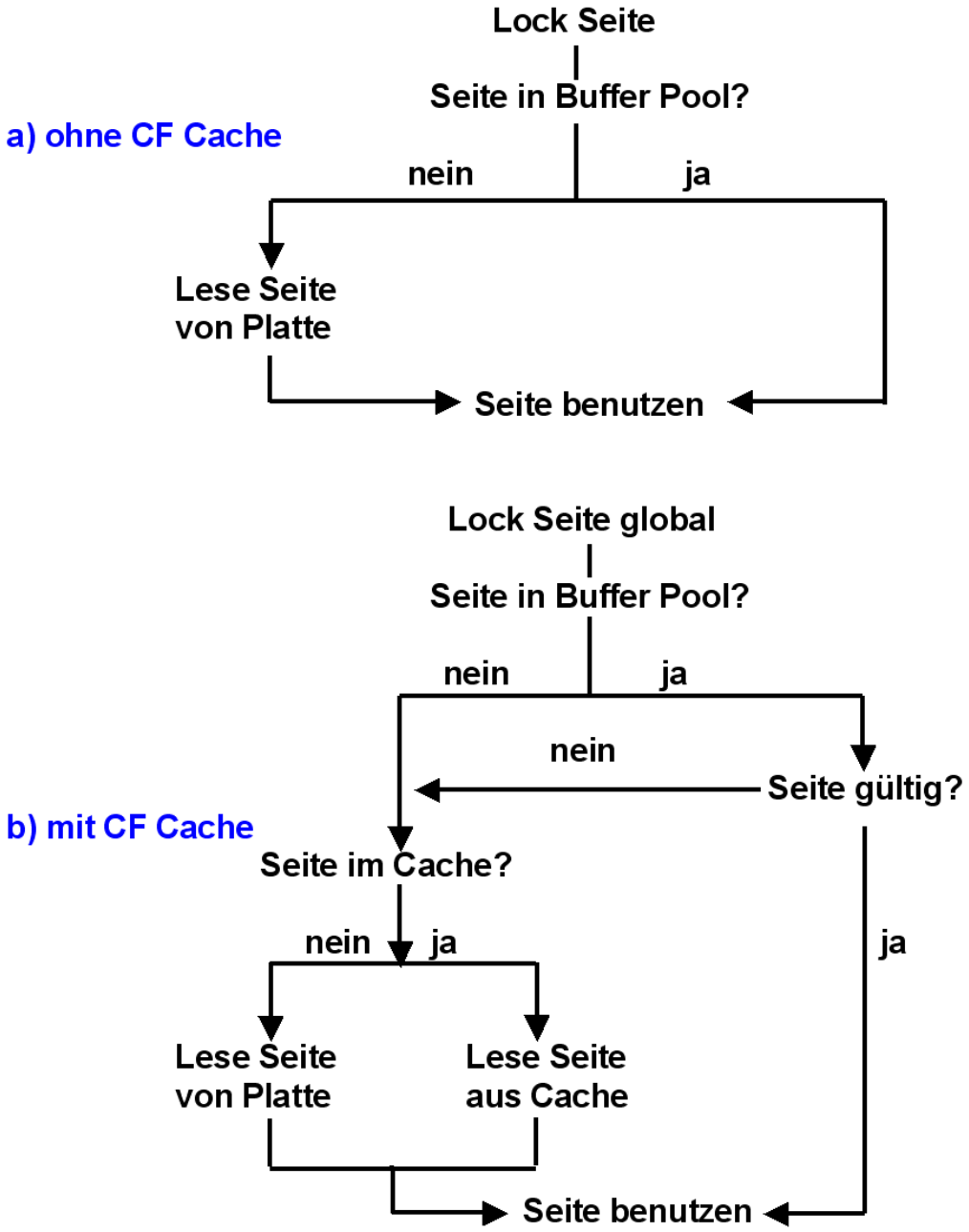


Abb. 11.4.6
CF Cache Algorithmen

Bei einem Plattenspeicherzugriff werden meistens 4096 Bytes große Blöcke von Daten transportiert (identisch mit der Rahmen Größe).

Ist kein Cache in der Coupling Facility vorhanden, erfolgt der Zugriff auf die folgende Weise:

1. **Lock auf die gewünschte Seite setzen.**
2. **Seite lesen, falls im Buffer Pool vorhanden**
3. **Falls nicht vorhanden, Seite vom Plattenspeicher einlesen.**
4. **Update des Cache Directories in der CF.**

Wird ein Coupling Facility Cache genutzt, wird die Seite aus dem Cache gelesen, falls vorhanden. Wenn nicht, wird die Seite vom Plattenspeicher eingelesen.

In diesem Fall schreibt DB2 zusätzlich die Seite in den CF Cache. Dies ist ein Store-in-Cache; nicht bei jedem Update der Seite wird diese auf dem Plattenspeicher geschrieben. Somit kann die CF Cache Version des Blockes jüngeren Datums sein als die Version auf dem Plattenspeicher.

11.4.5 Adabas

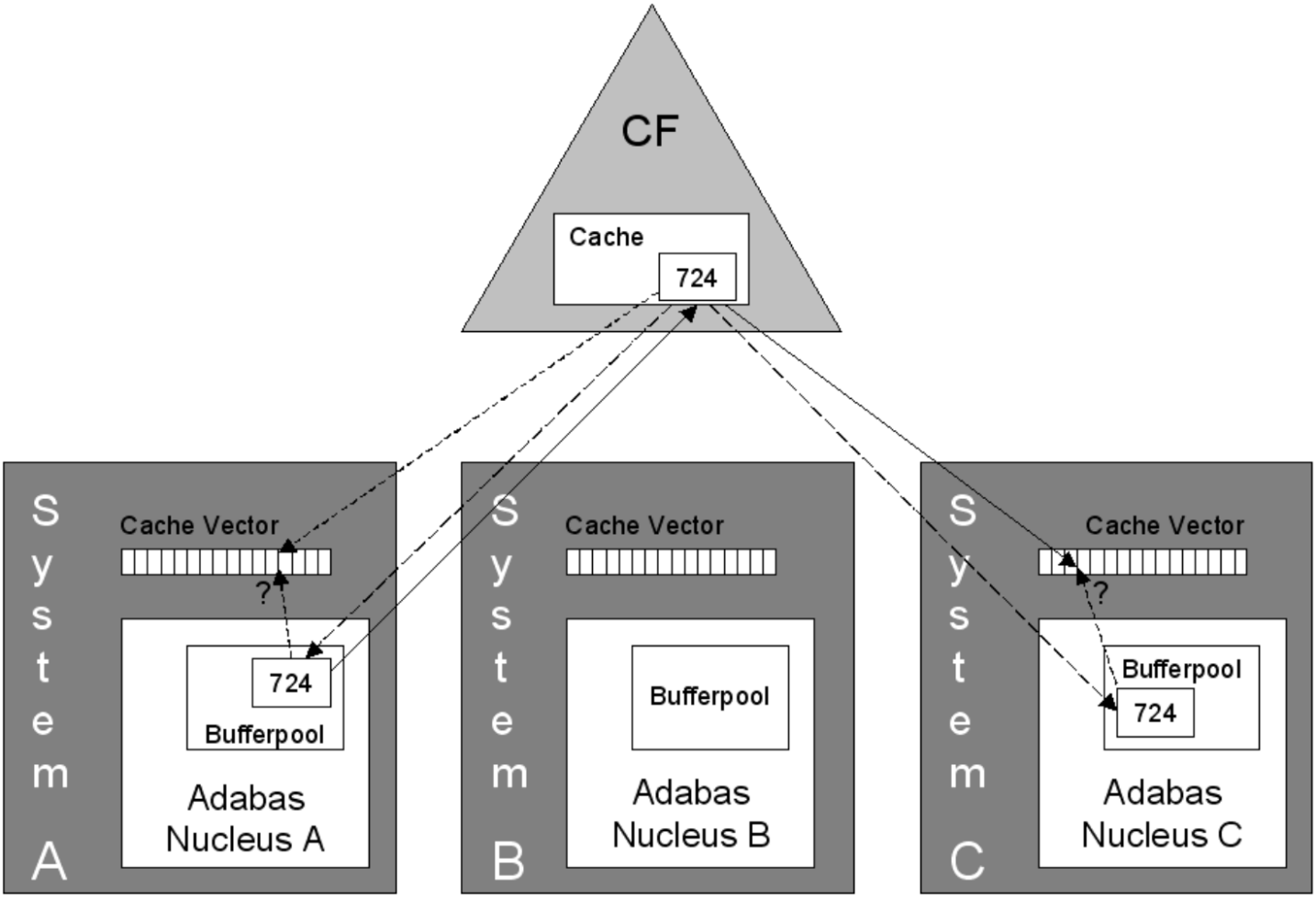


Abb. 11.4.7
Nutzung der CF Cache durch das ADABAS Datenbanksystem

Adabas (Adaptable Database System) wurde von der Software AG, Darmstadt, in 1971 eingeführt. Ursprünglich war ADABAS nur auf Mainframes verfügbar. Heute läuft ADABAS unter den z/OS, VSE, VM/CMS, Fujitsu's Mainframe Betriebssystem Facom, den Fujitsu/Siemens BS2000 Betriebssystemen sowie den Windows, Solaris, AIX, HP-UX, SUSE-Linux und Red Hat Linux Betriebssystemen.

Weltweit wird Adabas von etwa 3000 Installationen benutzt, darunter etwa 100 – 200 z/OS Installationen. Adabas ist ein Geheimtipp für neue Datenbankbenutzer; in Bezug auf Performance nimmt es auch heute noch eine Spitzenposition ein.

Adabas ist kein relationales Datenbanksystem (auch wenn die Firma Adabas behauptet, Adabas sei fast relational). Daten in der ADABAS-Datenbank sind hierarchisch in Files strukturiert; eine File entspricht in etwa einer SQL-Tabelle. Die Files bestehen aus Records; die Felder der Records entsprechen etwa den Spalten einer SQL-Tabelle. Es bestehen einige Ähnlichkeiten mit der IMS-Datenbank.

Als „Nucleus“ bezeichnet Adabas die Kernkomponente seines Datenbanksystems.

Wenn Nucleus A den Block 724 seines Buffer Pools ändert, wird die nicht mehr korrekte Kopie von Block 724 in dem Bufferpool von Nucleus C als ungültig markiert. Nucleus B hat keine Kopie von Block 724, und ist deshalb nicht betroffen.

Die Adabas Database der Software AG in Darmstadt nutzt ebenfalls die Coupling Facility als Plattenspeicher-Cache.

11.4.6 Cast Out des CF Caches

Problem: Die Coupling Facility ist nur mit den Knoten (Systemen) des Sysplex verbunden; sie hat keine direkte Verbindung mit den Plattenspeichern.

Frage: Wenn der Plattenspeicher-Cache in der Coupling Facility zu voll wird, wie werden Teile auf einen Plattenspeicher ausgelagert, um Platz zu schaffen?

Wird ein neuer Buffer in die CF Cache geschrieben, muss dafür Platz geschaffen werden und ein anderer CF Puffer auf den Plattenspeicher ausgelagert werden. Die Coupling Facility ist aber nur mit den Systemen (Knoten) verbunden. Sie hat keinen direkten Zugriff auf die Plattenspeicher.

DB2-Instanzen in den einzelnen Systemen unterhalten jeweils einen „Cast-Out“ Thread, der einen Buffer aus dem Cache lesen und auf den Plattenspeicher schreiben kann.

Die Cast-Out-Verantwortung wird nach dem Round-Robin-Algorithmus (oder einem anderen Algorithmus) den einzelnen Threads zugeordnet. Ein Cast-Out erfolgt jeweils für eine Gruppe von Seiten.

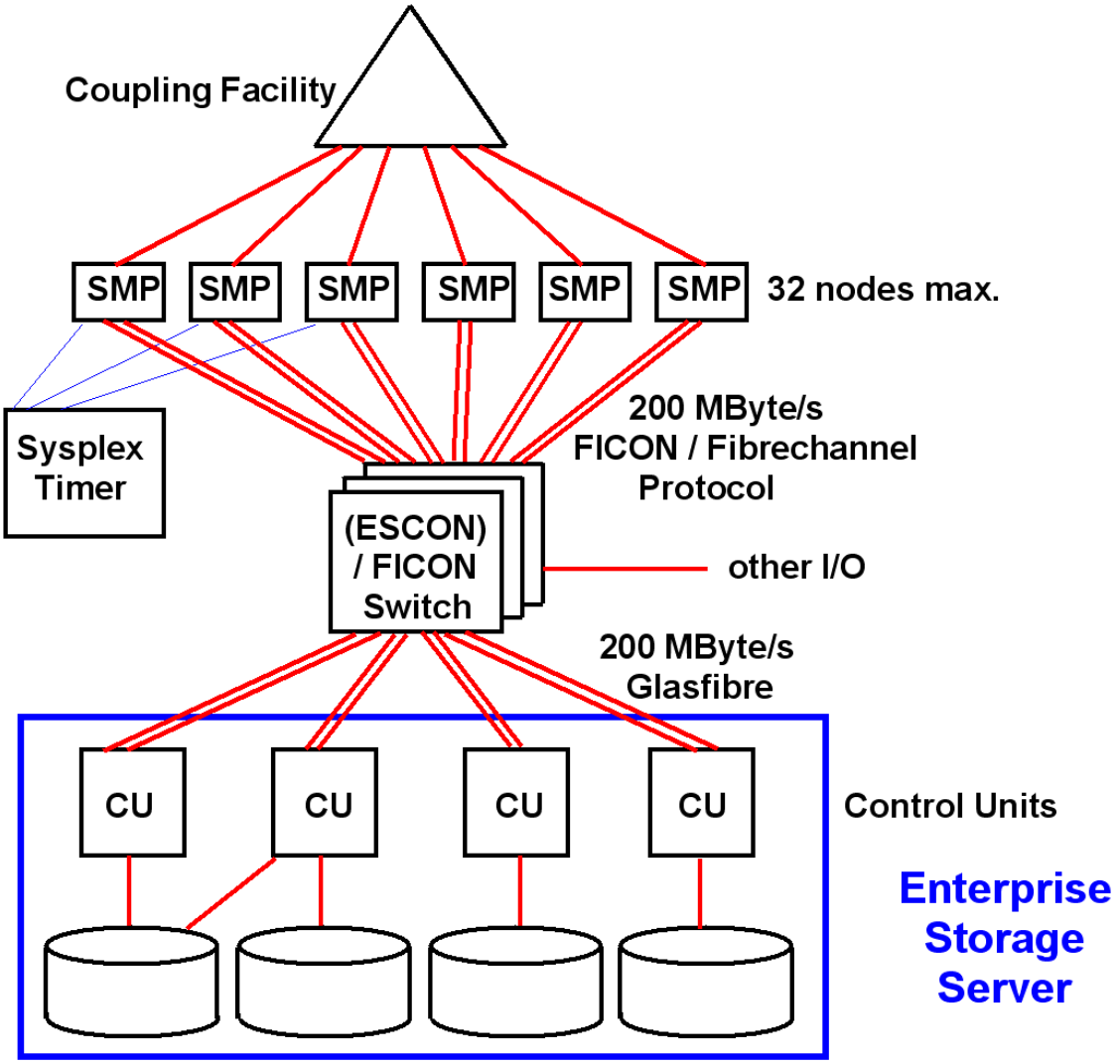


Abb. 11.4.8
Cast Out der CF Cache über das Coupling Link eines der beteiligten Systeme

Frage: Ist es nicht bedenklich, Daten nur in den Coupling Facility Cache zu schreiben, und nicht sofort auf den Plattenspeicher?

Ein Duplikat aller Daten befinden sich in den Buffer Pools der einzelnen Systeme. Im Fehlerfall kann hiermit ein Rebuild des CF Cache-Inhaltes stattfinden. Weiterhin verfügt eine Installation praktisch immer über 2 CFs, wobei die zweite CF alle Daten der ersten CF dupliziert und im Fehlerfall automatisch die Funktionen der ersten CF übernehmen kann.

11.4.7 CF Listen- / Queue-Strukturen

Neben dem Lock und dem Cache Management enthält die Coupling Facility Listen/Queue-Strukturen, die vor allem für eine zentrale Verwaltung aller angeschlossenen Systeme eingesetzt werden.

Beispiele hierfür sind:

- **Clusterweite RACF-Steuerung. Ein Sysplex-Cluster besteht aus mehreren z/OS-Instanzen. Im einfachsten Fall müsste sich ein Benutzer mit getrennten Passworten in jedes System einzeln einloggen. „Single Sign-On“ ist eine Einrichtung, mit der der Benutzer mit einem einzigen Sign On Zugriffsrechte auf alle Ressourcen eines Sysplex erhält. Die entsprechenden RACF-Benutzerprofile werden von der Coupling Facility zentral in einer Queue/List-Struktur verwaltet.**
- **Work Load Management (WLM)-Instanzen tauschen periodisch Statusinformationen aus, um Transaktionen dynamisch an unterbelastete Systeme weiter zu reichen.**
- **MQSeries: Queue-Sharing Group eines WebSphere MQ-Clusters**

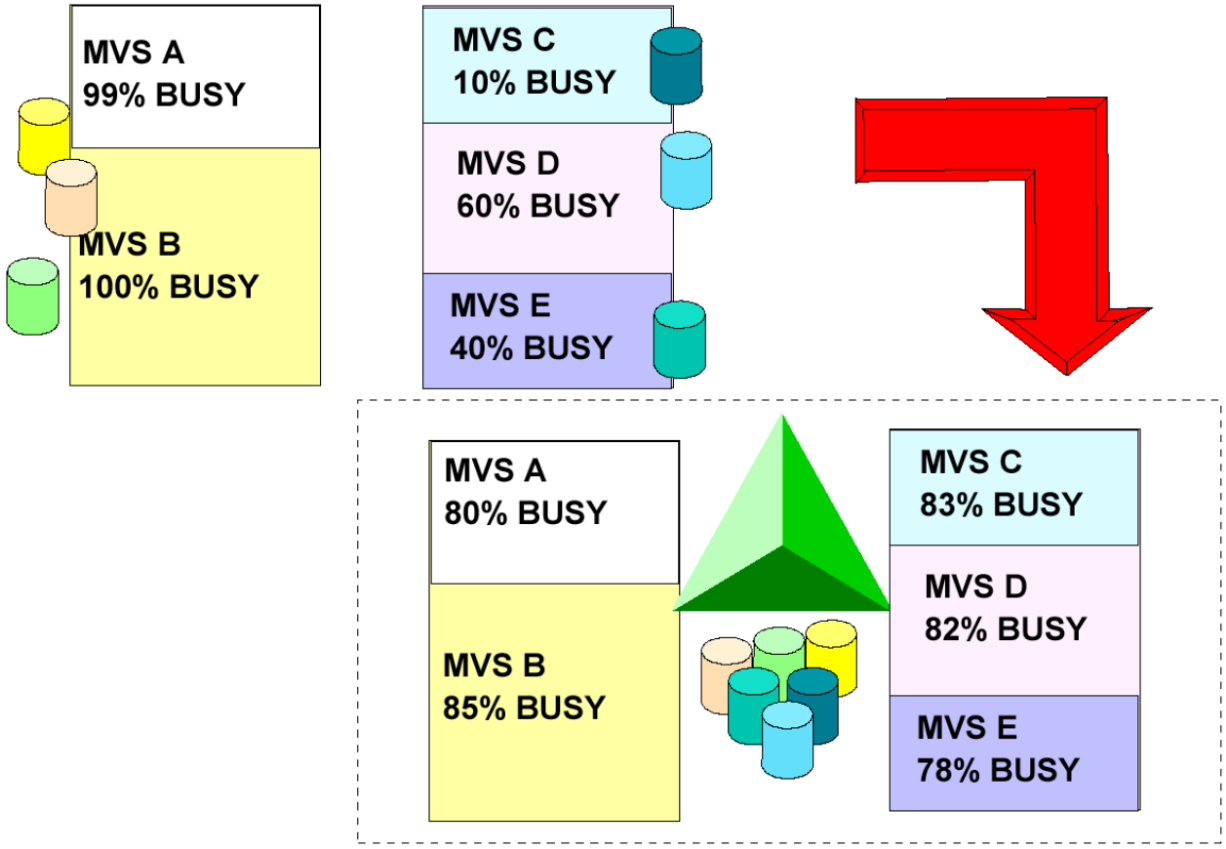


Abb. 11.4.9
Work Load Balancing

Für einen Zugriff auf die Queue/List-Strukturen bestehen drei Möglichkeiten:

- **LIFO Queue**
- **FIFO Queue**
- **Key Sequenced**

Key Sequenced bedeutet, dass auf ein bestimmtes Item in einer Queue/List-Struktur mittels eines Schlüssels zugegriffen werden kann.

In Abb. 11.4.9 sind 5 Systeme dargestellt: MVS A, MVS B, MVS C, MVS D und MVS E.

Es ist fast unausbleiblich, dass diese Systeme unterschiedlich ausgelastet sind.

Indem man die Rechner zu einem Sysplex zusammenfasst, kann eine Systemkomponente, der „Work Load Manager“ (WLM, siehe Kapitel 13) für eine gleichmäßige Auslastung aller 5 Systeme sorgen. Die entsprechenden Daten werden in der CF gespeichert.

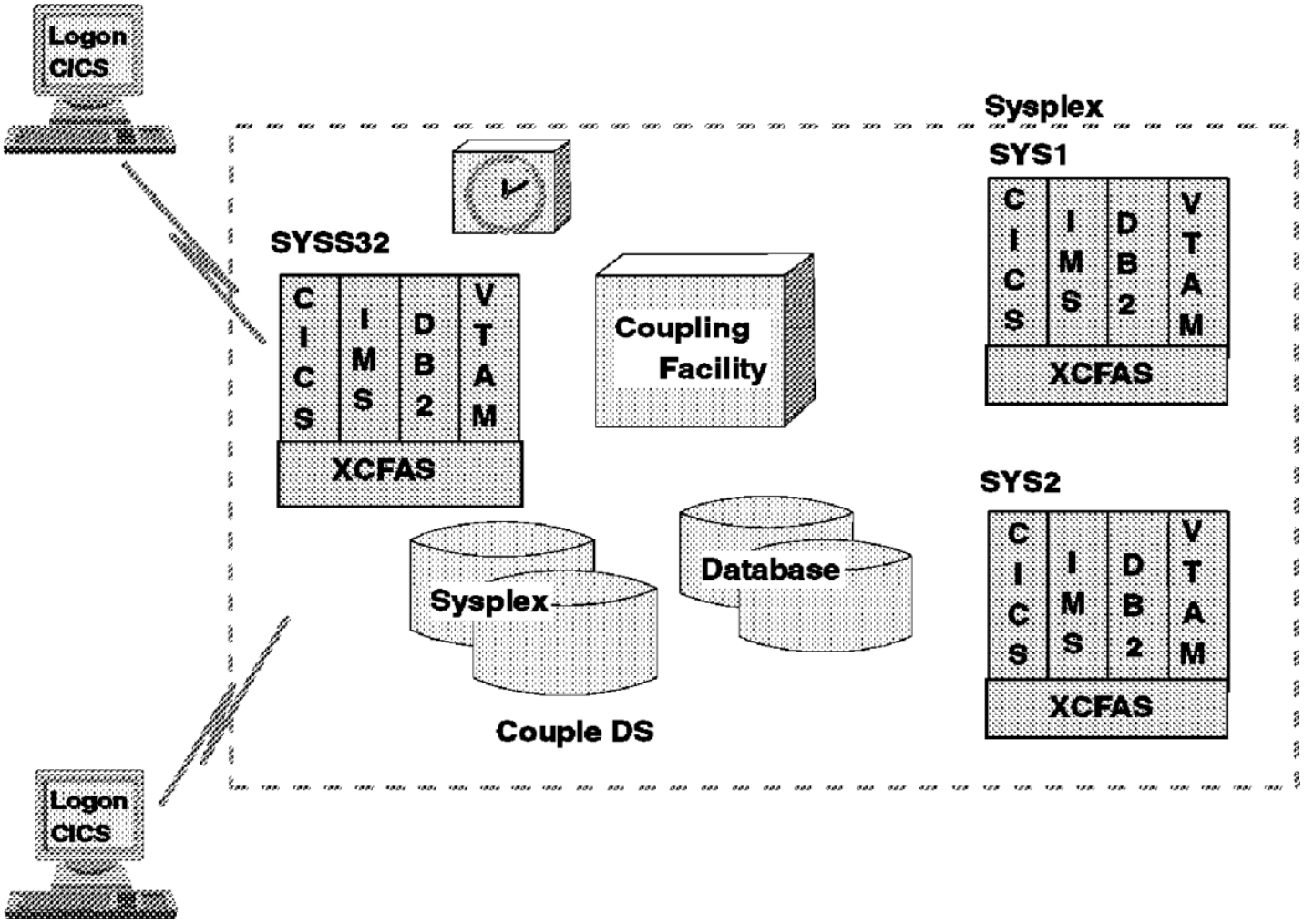


Abb. 11.4.10
Cross Coupling Facility Address Space

Angenommen seien mehrere Instanzen einer Anwendung oder eines Subsystems auf unterschiedlichen Knoten eines Sysplex, z.B. CICS oder WebSphere. Mit Hilfe von Cross Coupling Facility Address Spaces (XCFAS) können die Instanzen Statusinformationen austauschen oder miteinander kommunizieren.

Die gemeinsam genutzten Daten befinden sich als Listen- oder Queue-Strukturen auf der Coupling Facility. Der Zugriff auf diese Daten erfolgt mit Hilfe des Cross-System Extended Services (XES)-Protokolls, welches Zugriffs- und Verwaltungsdienste zur Verfügung stellt.

11.4.8 Sysplex-Performance

Wie sieht es mit dem Leistungsverhalten und der Skalierbarkeit eines Sysplex aus?

In den meisten Fällen ist es sehr schwierig, eine existierende Anwendung, die auf einer einzelnen CPU läuft auf einen Mehrfachrechner zu portieren. In vielen Fällen bringt bei 10 oder 30 CPUs jede weitere CPU keinen nennenswerten Leistungsgewinn mehr.

Der Sysplex und besonders die Coupling Facility bilden hier eine Ausnahme. Sie weisen hervorragende Skalierungseigenschaften auf. Dies wird in den beiden folgenden Abbildungen dargestellt.

Heute (2013) ist die Coupling Facility immer noch ein Mainframe-Alleinstellungsmerkmal. Wir erwarten, dass in Zukunft die Geschwindigkeit einer einzelnen CPU nur noch langsam steigen wird. Leistungssteigerungen unser Rechner sind daher in Zukunft in erster Linie durch den Einsatz von Mehrfachrechnern zu erwarten.

Es ist daher vorstellbar, dass eine Coupling Facility in Zukunft auch für die x86 Platform verfügbar sein wird – warten wir es ab.

Installation	Anzahl Systeme	% Sysplex Overhead
A	4	11 %
B	3	10
C	8	9
D	2	7
E	11	10
DB2 Warehouse Workload	2	13

Abb. 11.4.11
Sysplex Overhead

Die Sysplex Support Software (wenn installiert) erzeugt in jedem System zusätzlichen Overhead, selbst wenn der Sysplex nur aus einem einzigen System besteht. In jedem System wird zusätzliche CPU Kapazität benötigt, um den gleichen Durchsatz zu erreichen.

Die Sysplex Support Software wird nur dann installiert, wenn der Rechner als Bestandteil eines Sysplex genutzt wird.

Gezeigt ist der durch die Sysplex Support Software verursachte Overhead (zusätzlich benötigte CPU Zyklen). Die Messungen erfolgten auf den Sysplex Installationen von 5 großen Unternehmen, hier als A, B, C, D und E bezeichnet. Weiterhin erfolgten Messungen auf einer IBM internen DB2 Warehouse Installation.

Der Overhead bewegt sich zwischen 7 und 13 %.

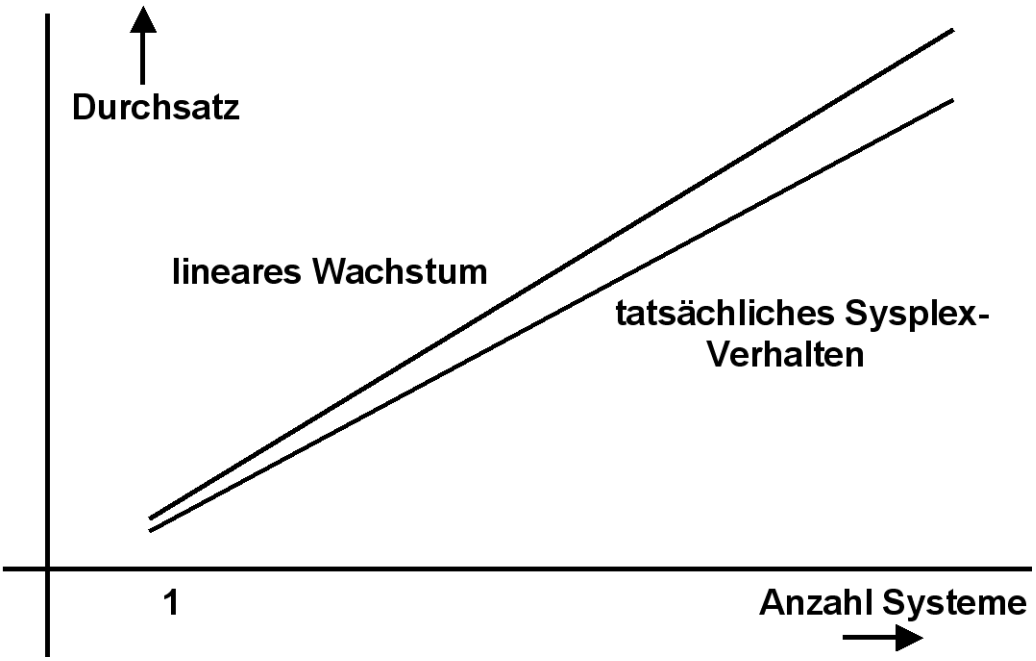


Abb. 11.4.12
Sysplex Skalierung

Mit nur einem einzigen System entsteht der erwähnte Sysplex Support Software Overhead. Mit einer wachsenden Anzahl von Systemen beobachten wir eine lineare Skalierung.

“The Parallel Sysplex environment can scale nearly linearly from 2 to 32 systems. The aggregate capacity of this configuration meets every processing requirement known today.”

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zconcepts/zconc_pllsysscale.htm

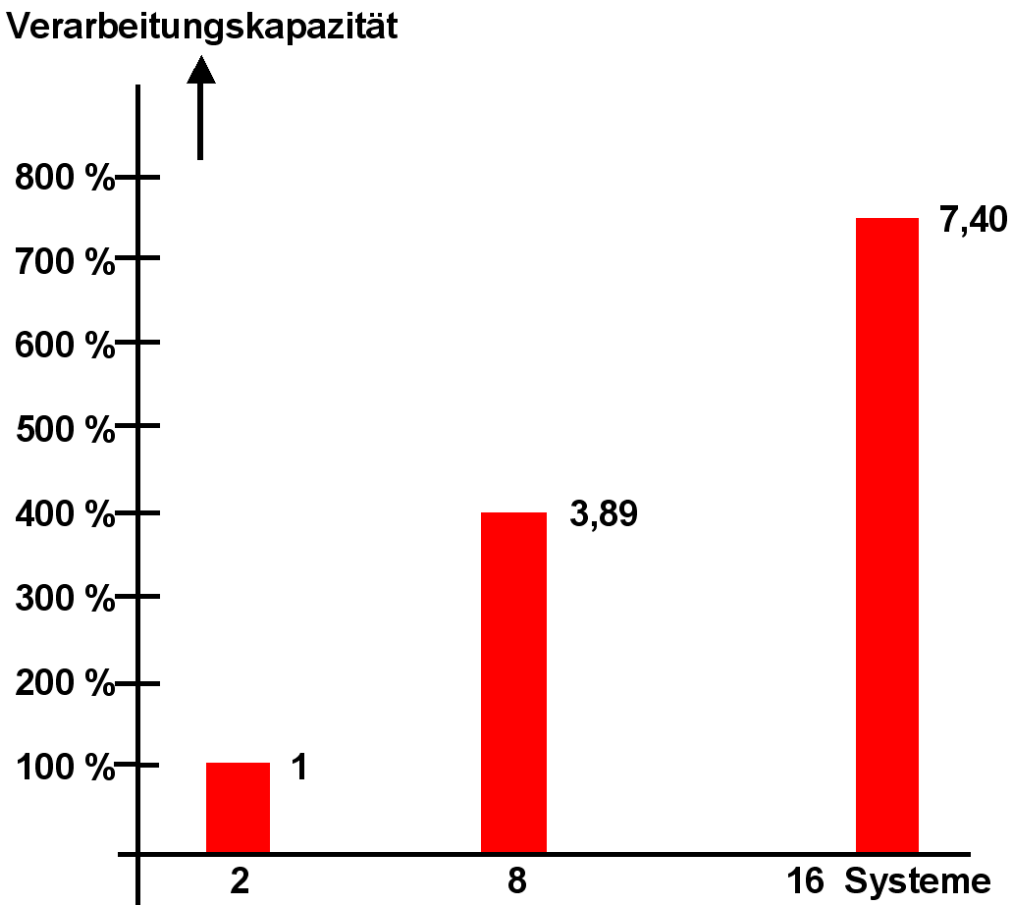


Abb. 11.4.13
Parallel Sysplex Leistungsverhalten

Abb. 11.4.13 zeigt Testergebnisse einer Installation bestehend aus CICS-Transaktionsserver, CICSplex System Manager und einer IMS Datenbank, mit einer Mischung von OLTP, Reservierung, Data Warehouse und Bankanwendungen.

Mit 2 Systemen, die mit einer CF verbunden sind, erreicht man eine Leistung von 100 %.

Mit 16 Systemen sollte man theoretisch eine Leistung von 800 % erreichen. Tatsächlich erreicht werden 740 %.

Jedes System kann z. B. 8 oder 16 CPUs enthalten.

Dies ist ein sehr guter Skalierungswert.

Literatur: Coupling Facility Performance: A Real World Perspective, IBM Redbook, March 2006, http://www.redbooks.ibm.com/redpapers/pdfs/redp4014.pdf

11.5 Weiterführende Information

11. 5.1 Literatur

Wilhelm G. Spruth, Erhard Rahm:
Sysplex-Cluster Technologien für Hochleistungs-Datenbanken.
Datenbank-Spektrum, Heft 3, 2002, S. 16-26.

download:
http://www.cedix.de/Publication/Mirror/Sysplex3.pdf

Sonderheft des IBM Journal of Research and Development, Vol. 36, No.4, July 1992 zum Thema Sysplex Hardware:

Sonderheft des IBM System Journal, Vol. 36, No.2, April 1997 mit folgenden Aufsätzen:

J. M. Nick, B. B. Moore, J.-Y. Chung, and N. S. Bowen:
S/390 cluster technology: Parallel Sysplex, p. 172.

N. S. Bowen, D. A. Elko, J. F. Isenberg, and G. W. Wang: A locking facility for parallel systems, p. 202.

G. M. King, D. M. Dias, and P. S. Yu: Cluster architectures and S/390 Parallel Sysplex scalability, p. 221.

J. W. Josten, C. Mohan, I. Narang, and J. Z. Teng: DB2's use of the coupling facility for data sharing ,p. 327.

T. Banks, K. E. Davies, and C. Moxey: The evolution of CICS/ESA in the sysplex environment, p. 352.

J. P. Strickland: VSAM record-level data sharing, p. 361.

11. 5.2 Videos und weitere Information

Ein technisches Superdome Video
http://www.youtube.com/watch?v=Ude6jL4MxO4

Eine gut gemachte Video Präsentation zum Thema Sysplex ist zu finden unter:
http://www.yourepeat.com/watch/?v=OGJBq_s-a9A&feature=youtube_gdata

Ein weiteres Video erläutert, wie DB2 die Coupling Facility nutzt um Data Sharing zu implementieren:
http://www.yourepeat.com/watch/?v=OGJBq_s-a9A&feature=youtube_gdata

Information über das Sysplex Leistungsverhalten

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.92.5684
oder
http://www-sr.informatik.uni-tuebingen.de/~bloching/papers/pdpta00.pdf

Eine schöne Zusammenfassung über Mainframe Kurzinformationen ist zu finden unter
http://idcp.marist.edu/enterprisesystemseducation/ztidbitz.html

Das Marist College im Staate New York (http://www.marist.edu/) ist eine der weltweit führenden Hochschulen auf dem Gebiet der akademischen Mainframe Ausbildung.

Die neue (2013) Sun-Oracle Produktlinie ist beschrieben in

http://www.theregister.co.uk/2013/03/26/oracle_sparc_t5_m5_server/
und
http://www.theregister.co.uk/2013/03/26/oracle_sparc_t5_m5_server/page3.html

12. Virtualisierung

12.1 Partitionierung

12.1.1 Skalierung eines Symmetrischen Multiprozessors

Ein Symmetrischer Multiprozessor (SMP) mit zwei statt einer CPU sollte idealer weise die zweifache Leistung haben. Dies ist aus mehreren Gründen nicht der Fall.

Die Gründe für den Leistungsabfall (mangelnde Skalierung) sind Zugriffskonflikte bei der Hardware und Zugriffskonflikte auf Komponenten des Überwachers. Zugriffskonflikte bei der Hardware sind weniger kritisch. Durch leistungsfähige interne Verbindungen wird eine hohe Bandbreite zwischen den einzelnen Prozessoren, dem Hauptspeicher und den I/O-Kanälen erreicht. Anders dagegen bei Zugriffskonflikte auf Komponenten des Überwachers. Besonders kritisch ist es, wenn mehrere CPUs eines SMP gleichzeitig einen Dienst des Betriebssystem-Kernels in Anspruch nehmen wollen, der nur seriell durchgeführt werden kann. Ein typisches Beispiel ist der Scheduler.

Zur Lösung wird der Kernel in Bereiche aufgeteilt, die mit Locks (Sperren) versehen werden, und die von mehreren CPUs unabhängig voneinander benutzt werden können. So kann z.B. der Kernel für eine CPU eine Fehlseitenunterbrechung bearbeiten, und parallel dazu für eine andere CPU eine I/O Operation durchführen. Damit ein gleichzeitiger Zugriff durch zwei CPUs möglich ist, werden die beiden Teile des Kernels durch Locks geschützt.

Gelegentlich muss eine CPU darauf warten. dass eine andere CPU eine Ressource des Kernels freigibt. Um diese Wahrscheinlichkeit zu verringern, kann man den Kernel in eine möglichst große Anzahl von Teilbereichen gliedern, die durch Locks vor dem gleichzeitigen Zugriff durch mehrere CPUs geschützt werden. Teilt man jedoch den Kernel in zu viele Bereiche auf, verbringt er einen Großteil seiner Verarbeitungszeit mit der Lockverwaltung. Durch Feintuning und Abstimmung der einzelnen Kernel-Komponenten kann die Skalierung von SMP Rechnern verbessert werden. Mit wachsender Anzahl der CPUs verstärkt sich das Problem.

Wir haben in den vergangenen Jahrzehnten Fortschritte für SMP Rechner mit zunächst 2, dann 4, später 8, dann 12, 16 usw. CPUs gemacht, indem Wege gefunden wurden, den Kernel möglichst optimal in mehr und mehr Teile zu gliedern, die durch Locks von der gleichzeitigen Nutzung durch zwei oder mehr CPUs geschützt werden können, ohne dass der Aufwand für die Lock Verwaltung zu groß wird.

Verfahren für die Aufteilung des Kernels wurden mühsam während der letzten Jahrzehnte entwickelt, und werden immer noch weiter entwickelt. Eine stetige, aber kleine Performance Verbesserung wird so jedes Jahr erreicht.

Wie groß die Beeinträchtigung ist, hängt sehr stark von der Art der laufenden Anwendungen ab. Anwendungen, die Dienste des Kernels nur selten in Anspruch nehmen, verursachen nur wenige Beeinträchtigungen. Ein Beispiel ist die Berechnung der Mandelbrot Menge (Apfelmännchen).

Bei den kritischen Transaktions- und Datenbankanwendungen skalieren die meisten SMP Betriebssysteme heute (2013) bis zu 12 – 16 CPUs. Als einzige Ausnahme skaliert z/OS bis zu 24 – 32 CPUs [1]. Der Vorsprung rührt daher, dass die z/OS Entwickler 15 – 20 Jahre vor dem Rest der IT Industrie begonnen haben, eine möglichst optimale Aufteilung des Kernels in durch Locks geschützte Bereiche zu erforschen.

Eine weitere Ausnahme sind Rechnerarchitekturen, die auf massive Parallelität ausgelegt sind, z.B. der Tilera64 (Parallelität auf Threading-Ebene, http://www.tilera.com/) oder GPGPUs (General-Purpose Graphics Processing Unit) von NVidia und AMD (Parallelität auf Datenebene). Diese Systeme skalieren – je nach eingesetztem Algorithmus – problemlos auf bis zu mehrere tausend Recheneinheiten. Sie nutzen den Vorteil aus, dass hier kein Überwacher existiert. Nachteilig ist, dass die Algorithmen für diese Architekturen geeignet sein müssen und auf diesen Rechnerarchitekturen normale Anwendungen – wenn überhaupt – nur sehr ineffizient laufen. Sie sind nicht für die Aufgaben von klassischen CPUs verwendbar.

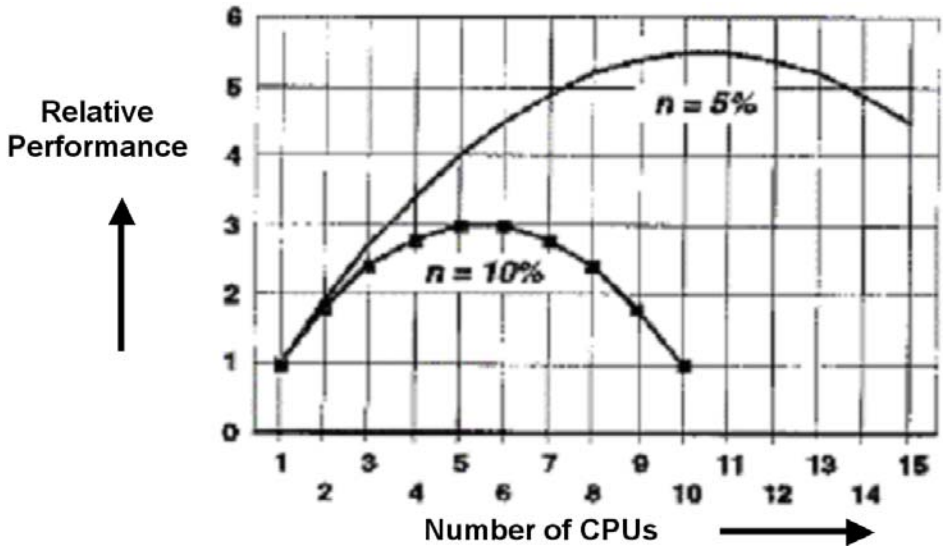


Abb. 12.1.1
Wachstum der Performance mit der Anzahl der CPUs

Die beiden dargestellten Kurven repräsentieren das typische Leistungsverhalten eines SMP als Funktion der Anzahl der eingesetzten CPUs. Die untere Kurve nimmt an, dass bei 2 CPUs jede CPU 10 % ihrer Leistung verliert; bei der oberen Kurve sind es 5 %. Mit wachender Anzahl von CPUs wird der Leistungsgewinn immer kleiner; ab einer Grenze wird der Leistungsgewinn negativ.

Dis bedeutet, dass SMPs nur mit einer begrenzen maximalen CPUs sinnvoll betrieben werden können. Diese Grenze ist sehr stark von der Art der Anwendungen abhängig.

Angenommen, ein Zweifach Prozessor leistet das Zweifache minus n % eines Einfach Prozessors. Für n = 10 % ist es kaum sinnvoll, mehr als 4 Prozessoren einzusetzen. Für n = 5 % sind es 8 Prozessoren.

Bei einem zEC12 Rechner mit z/OS ist n << 2 %. Es kann sinnvoll sein, einen SMP mit bis zu 32 Prozessoren einzusetzen. Meistens sind es weniger.

12.1.2 Non-Uniform Memory Architektur (NUMA)

Wenn wir also einen Rechner mit 101 – 256 CPUs betreiben, muss dieser in mehrere unabhängige SMPs partitioniert werden, wobei jede Partition einen SMP darstellt. Die mehrfachen Partitionen können als Cluster (lose gekoppelt) betrieben werden und werden durch ein Hochgeschwindigkeitsnetzwerk (z.B. ein Crossbar Switch) miteinander verbunden.

Die System Boards oder Cell Boards eines Sun 25k oder eines HP Superdome Rechners enthalten jeweils ihren eigenen Hauptspeicher. Im einfachsten Fall partitioniert man den Rechner so, dass jede Baugruppe (System Board oder Cell Board, Abschnitt 11.1.4) gleichzeitig ein SMP ist. Die CPUs einer Baugruppe greifen dann nur auf den Hauptspeicher der gleichen Baugruppe zu.

Wenn ein SMP aus mehr als einer Baugruppe besteht, muss eine SMP Betriebssystem Instanz auf den Hauptspeicher aller beteiligten Baugruppen zugreifen können.

Hierzu ist es erforderlich, dass eine CPU über den globalen Bus/Switch auf den Hauptspeicher einer fremden Baugruppe zugreift. Es ist damit möglich, dass zwei (oder mehr) Baugruppen einen einzigen SMP bilden. Ein Rechner mit dieser Einrichtung verfügt über eine „Non-uniform Memory Architektur“ (NUMA).

Dies geht aber nur bis zu der erwähnten Grenze von 12 – 16 CPUs. Ein SMP kann zwar aus mehr als einer Baugruppe bestehen, aber bestenfalls aus sehr wenigen Baugruppen. Deshalb besteht ein Rechner mit z.B. 16 Baugruppen aus mehreren SMPs, auch wenn ein SMP aus mehr als einer Baugruppe besteht. Die zu unterschiedlichen SMPs gehörigen Baugruppen werden durch Schalter voneinander getrennt (sog. „harte“ Partitionierung).

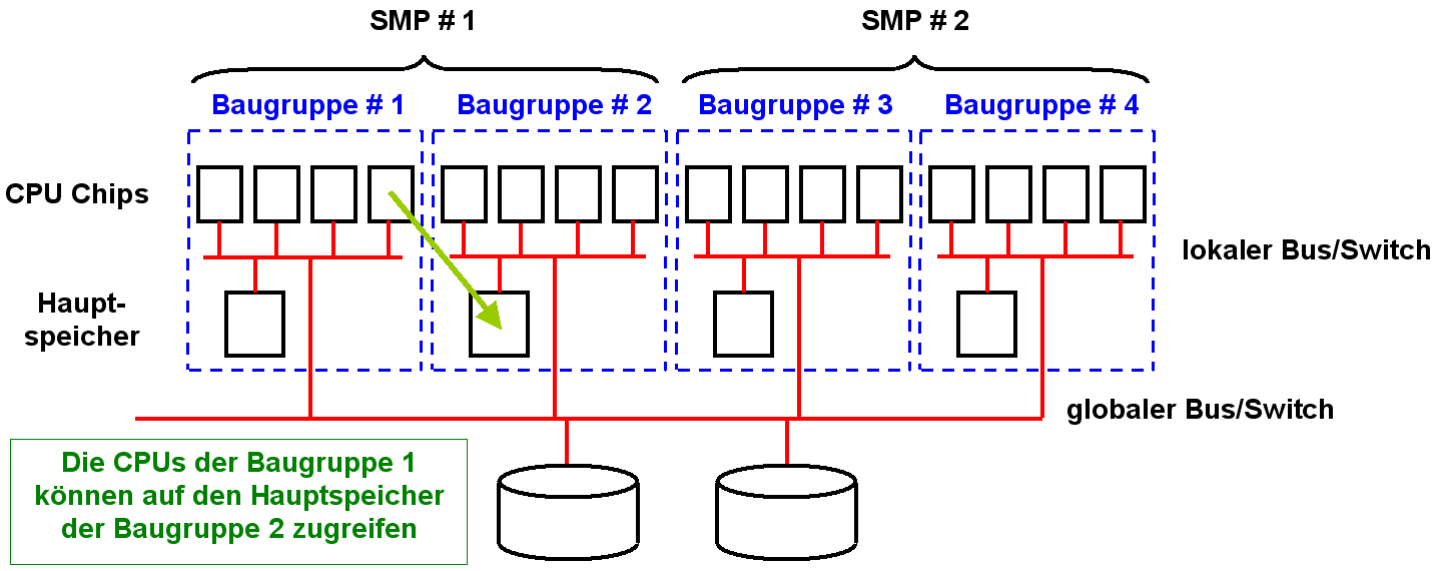


Abb. 12.1.2
Die CPUs der Baugruppe 1 können auf den Hauptspeicher der Baugruppe 2 zugreifen

In dem gezeigten Beispiel besteht der Rechner (System) aus 4 Baugruppen (z.B. „System“ oder „Cell“ Boards). Jede Baugruppe enthält 4 CPU Sockel und einem von allen CPU Sockeln gemeinsam genutzten Hauptspeicher. Jeweils 2 Baugruppen bilden einen SMP. Alle CPUs eines SMPs können jeweils auf den Hauptspeicher der anderen Baugruppe zugreifen.

Frage: Mit welchen realen Hauptspeicheradressen arbeiten die einzelnen Baugruppen?

Hierzu ist es notwendig, dass die 4 Hauptspeicher der 4 Baugruppen mit unterschiedlichen realen Adressen arbeiten.

Abb. 12.1.3 zeigt als Beispiel den Adressenraum von zwei SMP Rechnern bestehend aus jeweils 2 Baugruppen. Jede Baugruppe verfügt über einen Hauptspeicher von 4 GByte.

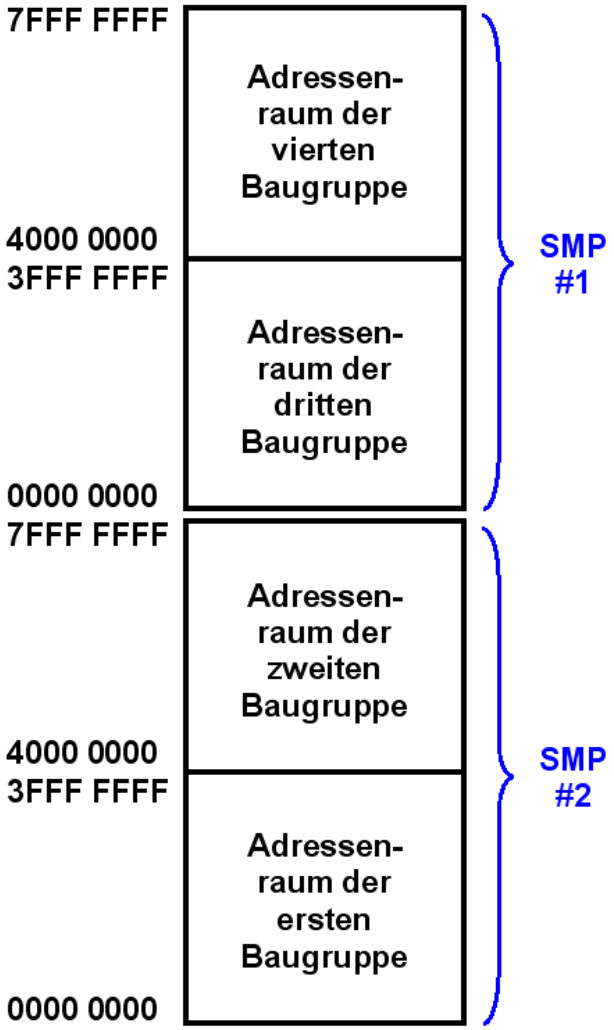


Abb. 12.1.3
NUMA Adressenraum

Der erste der 4 Baugruppen benutzt die Adressen Hex 0000 0000 bis 3FFF FFFF.

Der zweite der 4 Baugruppen benutzt die Adressen Hex 4000 0000 bis 7FFF FFFF.

Der dritte der 4 Baugruppen benutzt die Adressen Hex 0000 0000 bis 3FFF FFFF.

Der vierte der 4 Baugruppen benutzt die Adressen Hex 4000 0000 bis FFFF FFFF.

Da alle Prozesse in den beiden SMPs mit virtuellen Adressen arbeiten, ist diese Aufteilung der realen Adressen problemlos möglich.

Die Hauptspeicher DIMMs aller Baugruppen bilden einen gemeinsamen physischen Speicher. Da jede CPU auf die Speicher fremder Baugruppen zugreift, muss ein Verfahren für den Zugriff auf nicht-lokale Speicher existieren. Dies bewirkt die Dir-Einheit (Directory). Diese enthalten Informationen über alle auf den externen Baugruppen befindlichen Speichersegmente und deren Inhalte.

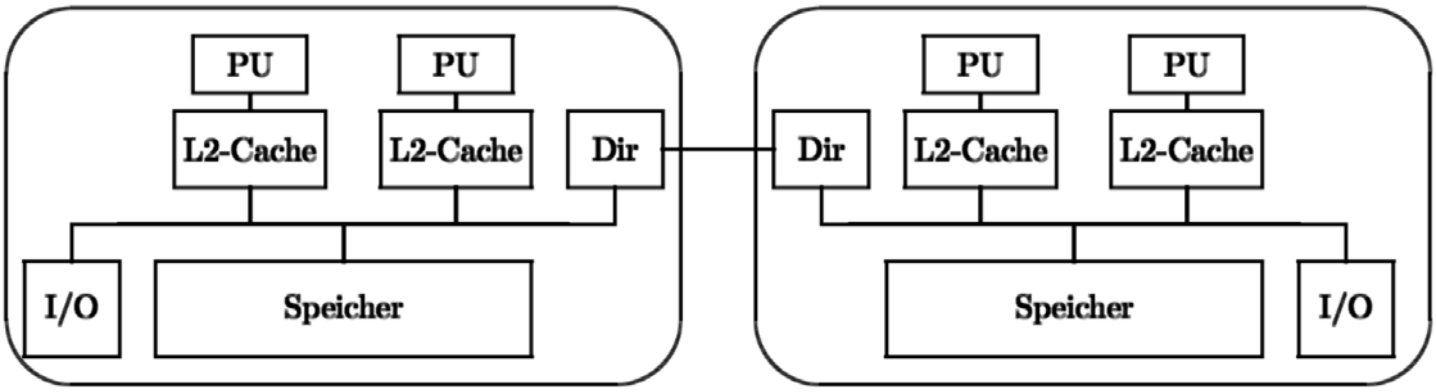


Abb. 12.1.4
Das Directory bildet Adressen beim Zugriff auf ein anderes System Board ab.

Bei der Partitionierung eines Rechners in mehrere SMPs kann ein SMP immer nur auf einen Teil des physischen Speichers zugreifen. Zusätzliche Einrichtungen teilen den physischen Speicher in mehrere logische Speicher für die einzelnen SMPs auf.

12.1.3 Harte Partitionierung

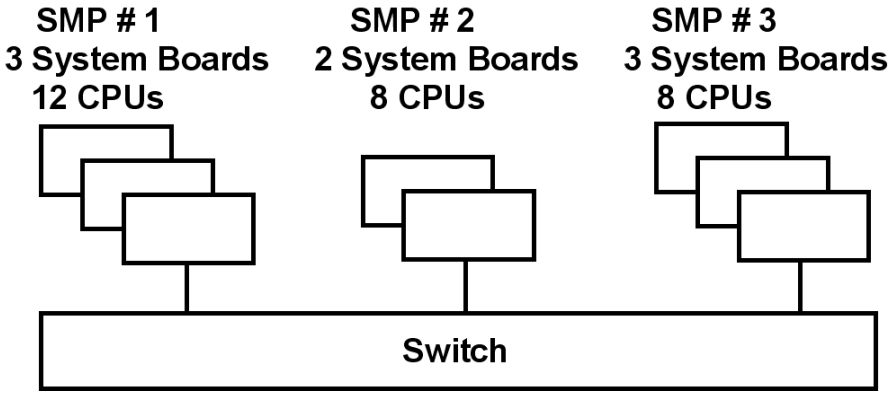


Abb. 12.1.5
Aufteilung eines Sun 15k Rechners mit 8 System Boards in 3 Harte Partitionen

Als Beispiel ist die Aufteilung eines Sun 25K (oder HP Superdome) Servers mit 8 Baugruppen (System Boards) mit je 4 single Core CPU Chips in mehrere parallel laufende SMPs gezeigt.

Die 8 System Boards werden in 3 Partitionen und 3 SMPS mit 3, 2 und 3 System Boards aufgeteilt. Jeder SMP hat ein eigenes Betriebssystem. Bei der „Harten Partitionierung“ erfolgt die Aufteilung statisch mit Hilfe von mechanischen oder elektronischen Schaltern.

Elektronische Schalter können während des laufenden Betriebes umkonfiguriert werden. Der System Administrator kann die Zuordnung der System Boards zu den einzelnen SMPs während des laufenden Betriebes ändern

Für Transaktionsanwendungen unter Unix sind kaum mehr als 3 System Boards (12 CPU Cores) pro SMP realistisch.

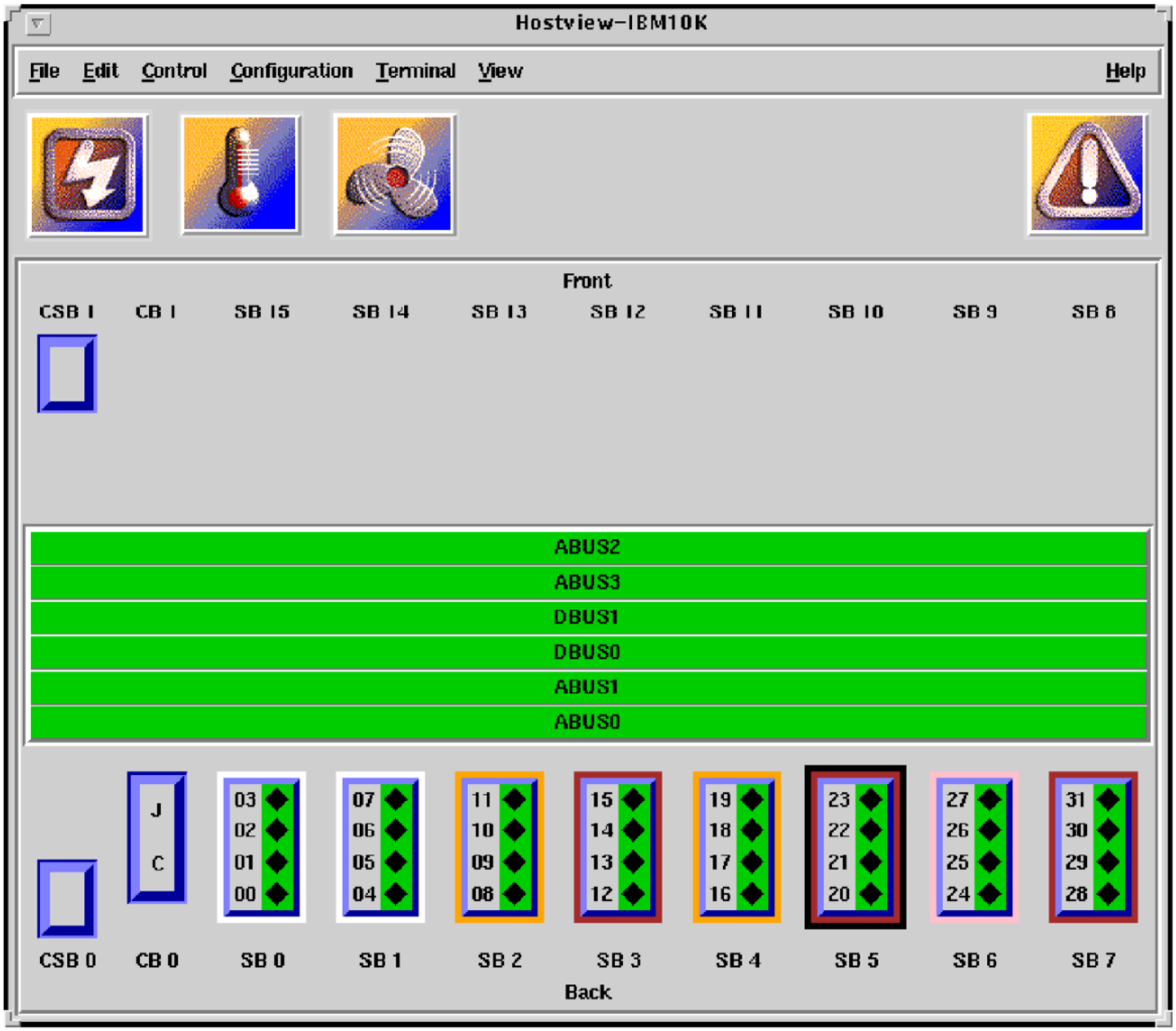


Abb. 12.1.6
Administrator Bildschirm eines Sun 15 K Rechners

Dargestellt sind 8 System Boards SB0…SB7. Sie werden in 3 SMPs partitioniert:

- **SMP1 besteht aus: SB0, SB1, SB6**
- **SMP2 besteht aus: SP2, SB4**
- **SMP3 besteht aus: SP3, SB5, SB7**

Angenommen, SMP2 ist überlastet, während SMP1 überschüssige Kapazität hat.

Der Administrator beschließt, SMP1 das System Board SB6 wegzunehmen und SMP2 zuzuordnen. Auf der in Abb. 12.1.6 dargestellten Konsole klickt der Administrator auf das SB6 Icon und zieht es mit der Maus herüber auf den SB2 Icon.

Die Granularität der Zuordnung auf die einzelnen SMPs (Partitionen) war ursprünglich in Einheiten von ganzen System Boards. Dies ist bei neueren Systemen verfeinert worden.

http://docs.oracle.com/cd/E19065-01/servers.12k/806-3509-13/806-3509-13.pdf, oder als Mirror http://www.cedix.de/VorlesMirror/Band1/SunFire15ka.pdf

12.1.4 Alternativen zur Harten Partitionierung

Die Harte Partitionierung (Hardware Partitioning) ist einfach und sauber zu implementieren, indem man System Boards oder Gruppen von System Boards durch elektronische Schalter elektrisch voneinander trennt. Die harten Partitionen verhalten sich dann wie getrennte physische Rechner. Über einen Switch können die einzelnen harten Partitionen miteinander kommunizieren.

Die Harte Partitionierung ist jedoch nicht sehr flexibel. Große Unix-Rechner wie z.B. HP Superdome und Sun 25K sowie den Nachfolgersystemen Sun M9000 und M5-32 bieten deshalb zusätzlich eine „Virtuelle Partitionierung“ an, auch als „Software Partitioning“ oder „Virtualisierung“ bezeichnet. Bei der virtuellen Partitionierung läuft jeder SMP in einer „virtuellen Maschine“.

Beide Alternativen haben Vor- und Nachteile. Der Benutzer entscheidet von Fall zu Fall, welche der beiden Alternativen er bei einer Neuinstallation einsetzten will. Bei den Sun und HP Servern hatte die virtuelle Partitionierung ursprünglich erhebliche Performanceprobleme. In den letzten Jahren wurden deutliche Verbesserungen erreicht..

Die Virtuelle Partitionierung mit Software ist in der Regel flexibler als die Hardware-Partitionierung. Allerdings ist durch den Einsatz von Software der Overhead größer, der für die Steuerung der Umgebung benötigt wird.

Da die Hardware-Umgebung virtuell abgebildet wird. kann auch mit Hardware gearbeitet werden, die physisch gar nicht vorhanden ist. Weiterhin werden die Ressourcen einer virtuellen Maschine nur dann benötig, wenn in der virtuellen Maschine Anwendungen laufen.

Anmerkung: Eine harte Partitionierung ist bei dem IBM Unix „System p“ und bei Mainframes nicht erforderlich und nicht verfügbar. An ihrer Stelle werden „Logische Partitionen“ (LPARs) eingesetzt, die auf anderen Pattformen bis heute nicht verfügbar sind. Siehe Abschnitt 12.3.

12.1.5 Emulator

Ein Emulator ist ein Software Paket, welches auf einem Rechner mit der Hardware-Architektur x (Host) einen Rechner mit der Hardware-Architektur y (Gast) emuliert. Jeder einzelne Maschinenbefehl eines Programms in der Architektur y wird durch den Emulator interpretiert.

Die Emulation ist sehr Performance-aufwändig. Ein Performance Verlust um einen Faktor 10 oder noch schlechter ist häufig anzutreffen. Mehrere emulierte Gast Rechner auf einem Host Rechner sind zwar möglich, aber wegen des Performanceverlustes nicht üblich.

Beispiele:

Eine Java Virtuelle Maschine emuliert eine Java Hardware Architektur auf einer x86, PowerPC oder System z Hardware. Die Java Hardware Architektur wurde seinerzeit mit dem Ziel entwickelt, den Performanceverlust möglichst klein zu halten (früher: Faustformel Faktor 3, heute unter optimalen Bedingungen ca. 20%).

Microsoft VirtualPC Typ 1 emuliert einen Intel/AMD Windows Rechner auf einem Apple MAC PowerPC Rechner mit einem (früher verfügbaren) PowerPC Mikroprozessor.

Bochs ist ein in C++ geschriebener Open Source Emulator, der einen Rechner mit der die Intel/AMD Architektur auf vielen anderen Plattformen emuliert, z.B. PowerPC.

Hercules und IBM zPDT emulieren einen System z Rechner mit dem z/OS Betriebssystem auf einem x86 Rechner.

Hercules ist ein Public Domain S/390 und System z Emulator, der für Linux, Windows (98, NT, 2000, and XP), Solaris, FreeBSD und Mac OS X Plattformen verfügbar ist. Moderne Mikroprozessoren sind schnell genug, so dass der Betrieb von z/OS unter Hercules auf einem PC für Experimentierzwecke oder für die Software Entwicklung für einen einzelnen Programmierer eine durchaus brauchbaren Performance ergibt.
Siehe http://www.hercules-390.org/.

Hercules ist eine einwandfrei funktionierende S/390 und System z Implementierung. Nicht verfügbar ist z/OS. Es würde zwar laufen, aber IBM vergibt keine Lizenzen für die Hercules Plattform. Verfügbar ist die letzte lizenzfreie Version MVS 3.8 aus dem Jahre 1981, die zwar kompatibel, aber eben doch sehr veraltet ist.

Allerdings verhält sich die Firma IBM zivilisierter als manche Unternehmen der Film- oder Musikindustrie bezüglich der Verfolgung von Hercules Lizenzverletzungen.

z Personal Development Tool (zPDT) ist ein offizielles IBM Produkt, vergleichbar zu Hercules. Es wird mit z/OS ausgeliefert, benötigt einen Dongle und läuft auf einer x86 Plattform. Der System z Emulator läuft unter Linux. Die Lizenzgebühren für zPDT sind jedoch recht hoch.

12.1.6 Virtuelle Maschinen

Eine virtuelle Maschine ist etwas Ähnliches wie eine emulierte Maschine. Der Unterschied ist: Auf einem Host Rechner mit der Hardware-Architektur x wird ein (oder mehrere) Gast Rechner der gleichen Architektur abgebildet. Die meisten Maschinenbefehle der virtuellen Maschine brauchen nicht interpretiert zu werden. Der Leistungsverlust einer virtuellen Maschine hält sich deshalb in erträglichen Grenzen (bis zu 50 % unter frühen VMWare Implementierungen, wenig mehr als 1 % unter z/OS PR/SM).

Beispiele für Hardware oder Betriebssysteme, die Virtuelle Maschinen unterstützen, sind:

VM/370, z/VM und der PR/SM Hypervisor für die System z und S/390 Hardware-Architektur.

Für den PowerPC Mikroprozessor existiert der LPAR Hypervisor.

XEN, Virtual Box, KVM, (alle Open Source), VMWare und Microsoft VirtualPC Typ 2 bilden mehrere Windows oder Linux Gast Maschinen auf einem Windows oder Linux Host ab. Oracle VM Server for Sparc ist eine Weiterentwicklung von XEN.

Intel Virtualization Technology für die Itanium Architecture (VT-i) sowie die x86 (Pentium) Architecture (VT-x) ist eine Hardware Erweiterung, welche die Virtualisierung durch Mechanismen in der Hardware verbessert.

Pacifica (AMD-V) für AMD Prozessoren ist eine zu VT-x nicht kompatible (und nach Ansicht mancher Experten überlegene) Alternative. I/O MMU Virtualization (AMD-Vi und -VT) erlaubt die direkte Anbindung von peripheren Geräten an die virtuellen Maschinen, dazu gehören auch DMA und Interrupt Remapping.

Paravirtualization wird von Xen und Denali implementiert.

Weiterführende Literatur: Joachim von Buttlar, Wilhelm G. Spruth:
Virtuelle Maschinen. zSeries und S/390 Partitionierung.
IFE - Informatik Forschung und Entwicklung, Heft 1/2004, Juli 2004.
http://www.cedix.de/Publication/Mirror/Partit50.pdf

Wie funktionieren virtuelle Maschinen ?

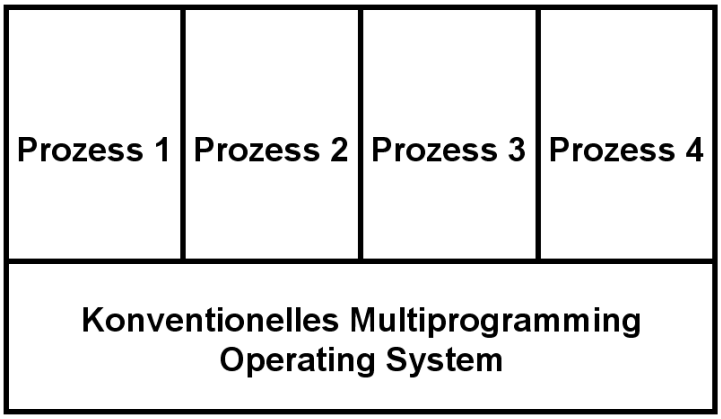


Abb. 12.1.7
Konventionelles Multiprogramming Operating System

Unter einem normalen multiprogrammierten Betriebssystem laufen zahlreiche Prozesse gleichzeitig ab. Jeder Prozess läuft in einem eigenen virtuellen Adressenraum.

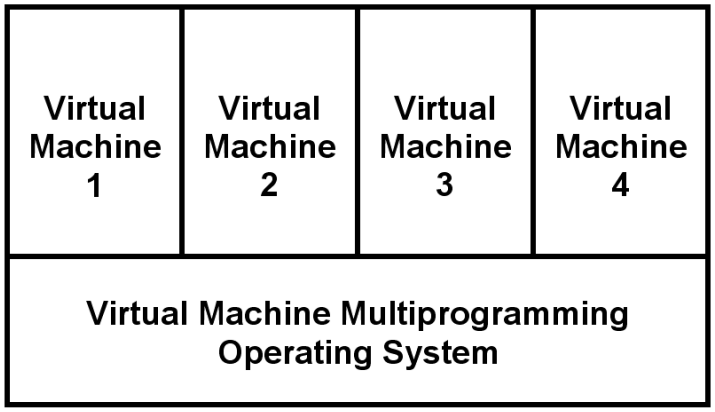


Abb. 12.1.8
Virtual Machine Multiprogramming Operating System

Unter einem Virtual Machine Operating System laufen spezielle Prozesse, von denen jeder eine virtuelle Maschine implementiert. Jede virtuelle Maschine hat ihr eigenes Betriebssystem, unter dem Anwendungen der virtuellen Maschine laufen. Auf zwei virtuellen Maschinen können unterschiedliche Betriebssysteme laufen. Eine virtuelle Maschine implementiert häufig einen SMP.

Das Virtual Machine Operating System wird häufig als Hypervisor oder als Host Betriebssystem bezeichnet. Analog bezeichnet man das emulierte Betriebssystem als das Gast-Betriebssystem, welches auf einer Gast-Maschine (virtuelle Maschine) läuft. Das Host-Betriebssystem verfügt über einen Host-Kernel (Überwacher) und das Gast-Betriebssystem verfügt über einen Gast-Kernel. Wir bezeichnen als Kernel denjenigen Teil des Betriebssystems, der im Kernel-Modus ausgeführt wird. Vielfach besteht der Hypervisor nur aus einem Host-Kernel. Es gibt aber Beispiele wie VMware, wo dies nicht der Fall ist, und zusätzliche Betriebssystem-Funktionen in Anspruch genommen werden.

Die Begriffe virtuelle Maschine und Gast Maschine sowie die Begriffe Hypervisor, Host Kernel und Virtual Machine Monitor (VMM) bedeuten jeweils das Gleiche und werden austauschbar verwendet.

12.1.7 Gleichzeitiger Betrieb mehrerer Betriebssysteme auf einem Rechner

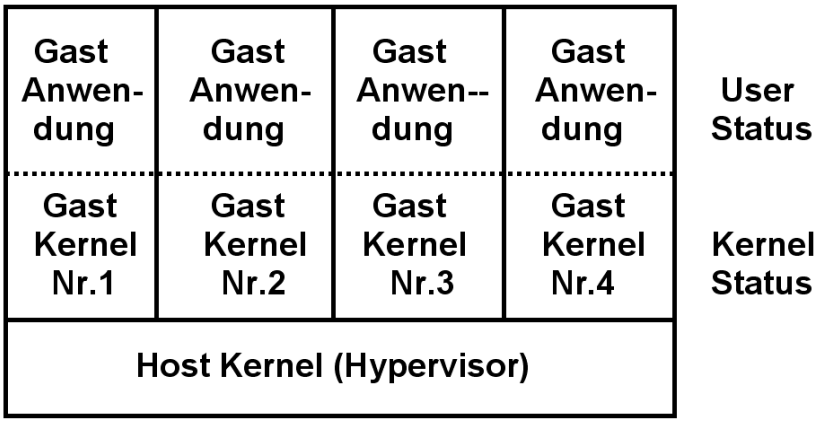


Abb. 12.1.9
Das Gast Betriebssystem besteht aus einem Gast Kernel und Gast Anwendungen

Der Ansatz, mehrere *Gast*-Betriebssysteme unter einem Host Betriebssystem zu betreiben, ermöglicht eine virtuelle Partitionierung im Gegensatz zu der bereits erwähnten harten Partitionierung, siehe Abb. 12.1.5 und 12.1.6. Beispielsweise kann ein Rechner mit 80 CPUs als 5 SMPs mit je 16 CPUs mittels der virtuellen Partitionierung betrieben werden.

Der Host-Kernel übernimmt die Kontrolle über den Rechner immer dann, wenn eine Gast-Maschine einen Maschinenbefehl auszuführen versucht, der das korrekte Verhalten des Hosts oder einer anderen Gast-Maschine beeinflussen würde. Derartige Maschinenbefehle werden als sensitive Befehle bezeichnet. Der Host-Kernel interpretiert diese Befehle für den Gast und stellt sicher, dass die Semantik erhalten bleibt. Wenn Gast und Host die gleiche Hardware Architektur verwenden, können im Gegensatz zur Emulation fremder Architekturen alle nicht-sensitiven Maschinenbefehle eines Prozesses direkt von der CPU ausgeführt werden. Ein Prozess führt hauptsächlich nicht-sensitive Befehle aus; daher ist der Leistungsabfall nur mäßig, wenn er als Gast betrieben wird. Im Gegensatz dazu ist eine vollständige Emulation sehr aufwendig.

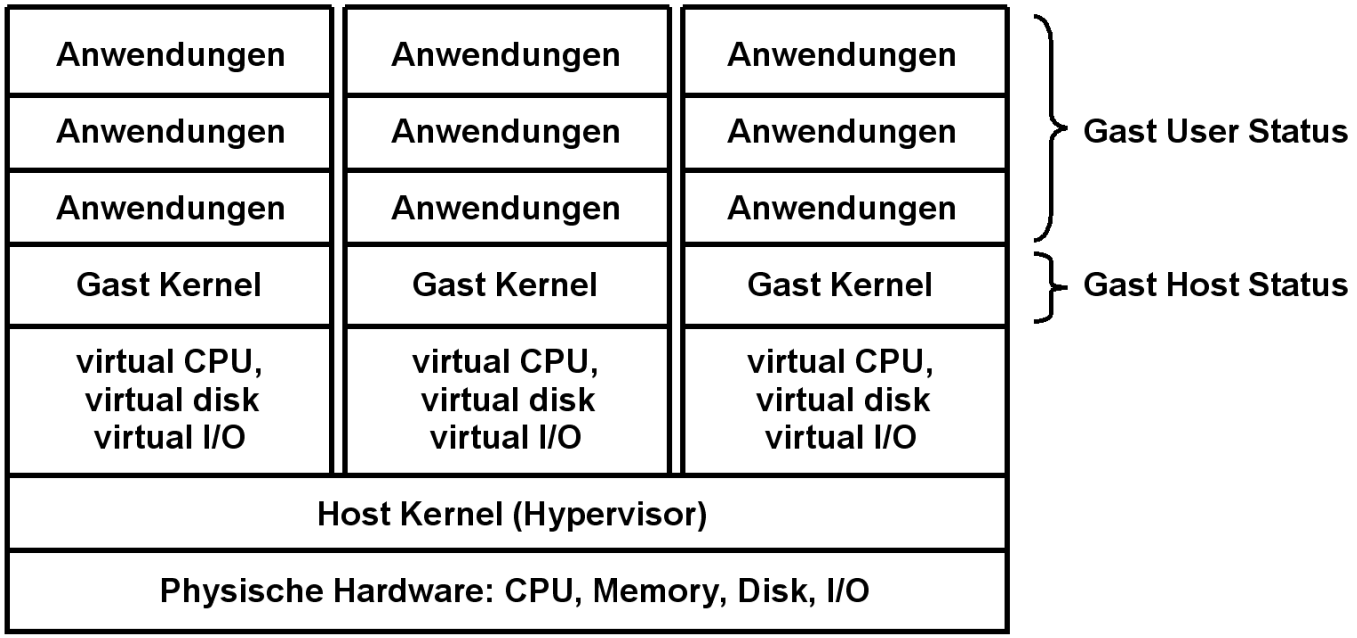


Abb. 12.1.10
Virtualisierung von CPUs und I/O Geräten

Es ist möglich, jedem Gast System eigene I/O Geräte, wie Plattenspeicher oder Drucker, fest zuzuordnen. Vielfach werden aber auch die I/O Geräte virtualisiert. Beispielsweise können mehrere virtuelle Drucker auf einen realen Drucker abgebildet werden. Virtuelle Plattenspeicher (Minidisks) können als Dateien auf einem realen Plattenspeicher abgebildet werden.

12.1.8 Alternativen für die Implementierung der Virtualisierung

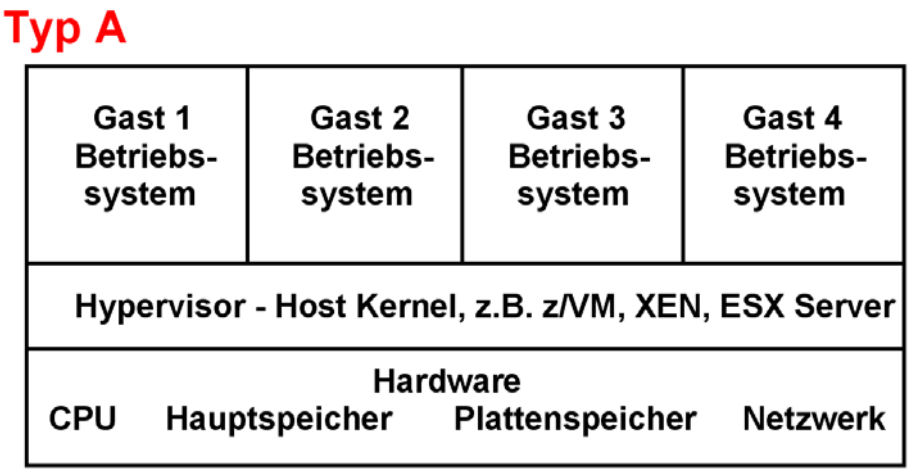


Abb. 12.1.11
Typ A ist in Enterprise Systemen gebräuchlich

Es existieren zwei Alternativen für die Implementierung.

Typ A ist die bisher beschriebene Konfiguration. Sie wird auf allen größeren Servern eingesetzt, besonders auch auf den Mainframes.

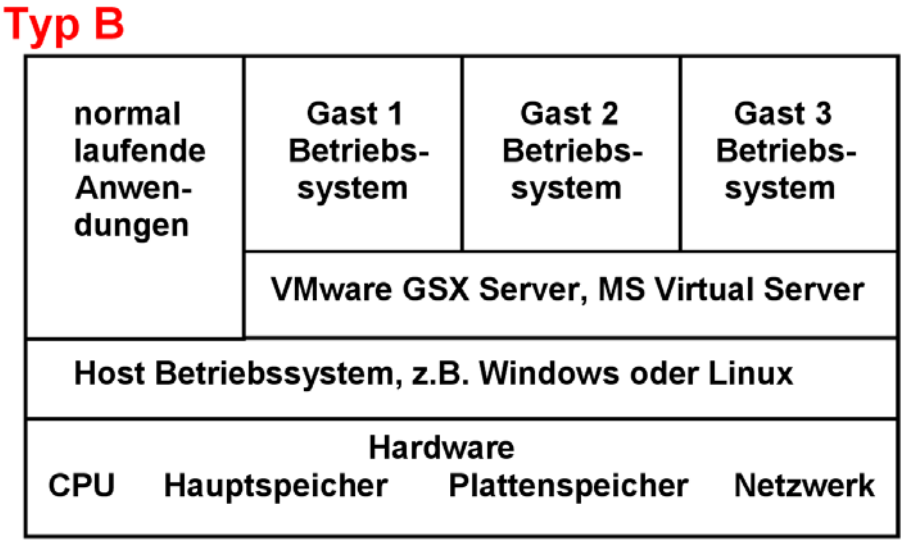


Abb. 12.1.12
Typ B ist auf dem PC gebräuchlich

Für den PC existiert als zusätzlich Alternative Typ B. Der Host Kernel läuft als Anwendung unter einem regulären PC Betriebssystem, z.B. Windows oder Linux. Er hat nicht alle Eigenschaften eines selbständigen Betriebssystem Kernels, sondern nutzt stattdessen teilweise Komponenten des darunterliegenden Windows oder Linux Betriebssystems.

Der Vorteil von Typ B ist, dass Anwendungen auch nicht virtualisiert mit besserem Leistungsverhalten laufen können. Als Nachteil laufen virtuelle Maschinen bei Typ B langsamer als bei Typ A.

Große Unix Server und Mainframes benutzen ausschließlich Alternative Typ A.

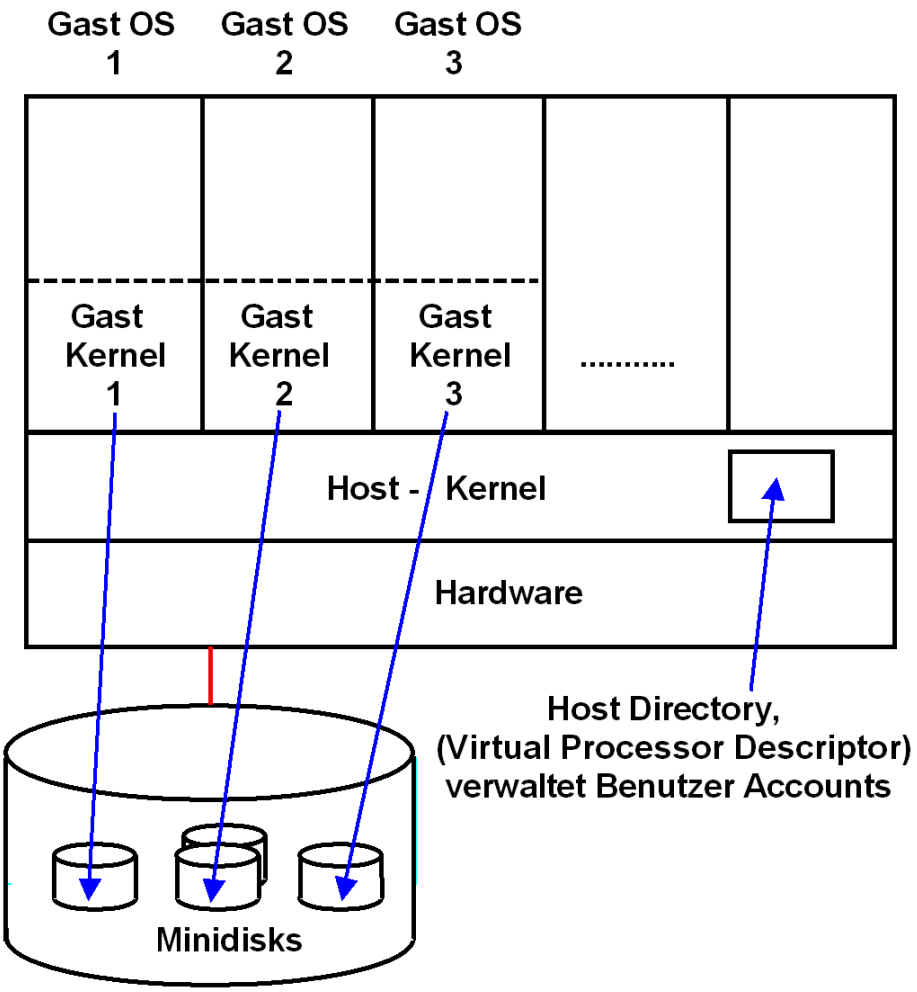


Abb. 12.1.13
In großen Systemen ordnet man den Gast Maschinen getrennte Festplattenspeicher zu

Im einfachsten Fall werden dem Gast Betriebssystemen Ressourcen wie

- **CPU-Zeit,**
- **Aufteilung auf mehrere CPUs in einem Mehrfachrechner,**
- **Hauptspeicher,**
- **Plattenspeicher**
- **Ein-/Ausgabe-Geräte und -Anschlüsse**

fest zugeordnet. Alternativ, z.B. Mainframes, ist auch eine dynamische Zuordnung möglich.

Der Host Kernel enthält ein „Host Directory“, welches die Konfiguration der virtuellen Maschinen verwaltet.

Bei kleinen Systemen mit nur einem einzigen physischen Plattenspeicher können mehrere Platten der virtuellen Systeme als „Minidisks“ auf dem physischen Plattenspeicher emuliert und den einzelnen virtuellen Maschinen zugeordnet werden.

Alle Gast Maschinen laufen in einem eigenen virtuellen Adressenraum

Der Host Kernel Zeitscheiben-Scheduler übergibt die Kontrolle über die CPU(s) einer Gastmaschine.

Der Kernel der Gast Maschine läuft im User Mode (Problem Mode). Wenn das Programm des Gastbetriebssystems versucht, einen privilegierten Maschinenbefehl auszuführen, führt dies zu einer Programmunterbrechung.

Die Programmunterbrechungsroutine des Host Kernels interpretiert den privilegierten Maschinenbefehl der Gastmaschine soweit als erforderlich und übergibt die Kontrolle zurück an den Kernel des Gastbetriebssystems.

Auf einem Mainframe kann das z/VM Betriebssystem die Rolle des Host Betriebssystems übernehmen. Auf der x86 Platform ist VMWare ein führender Vertreter.

12.1.9 Ausführung einer READ I/O-Operation

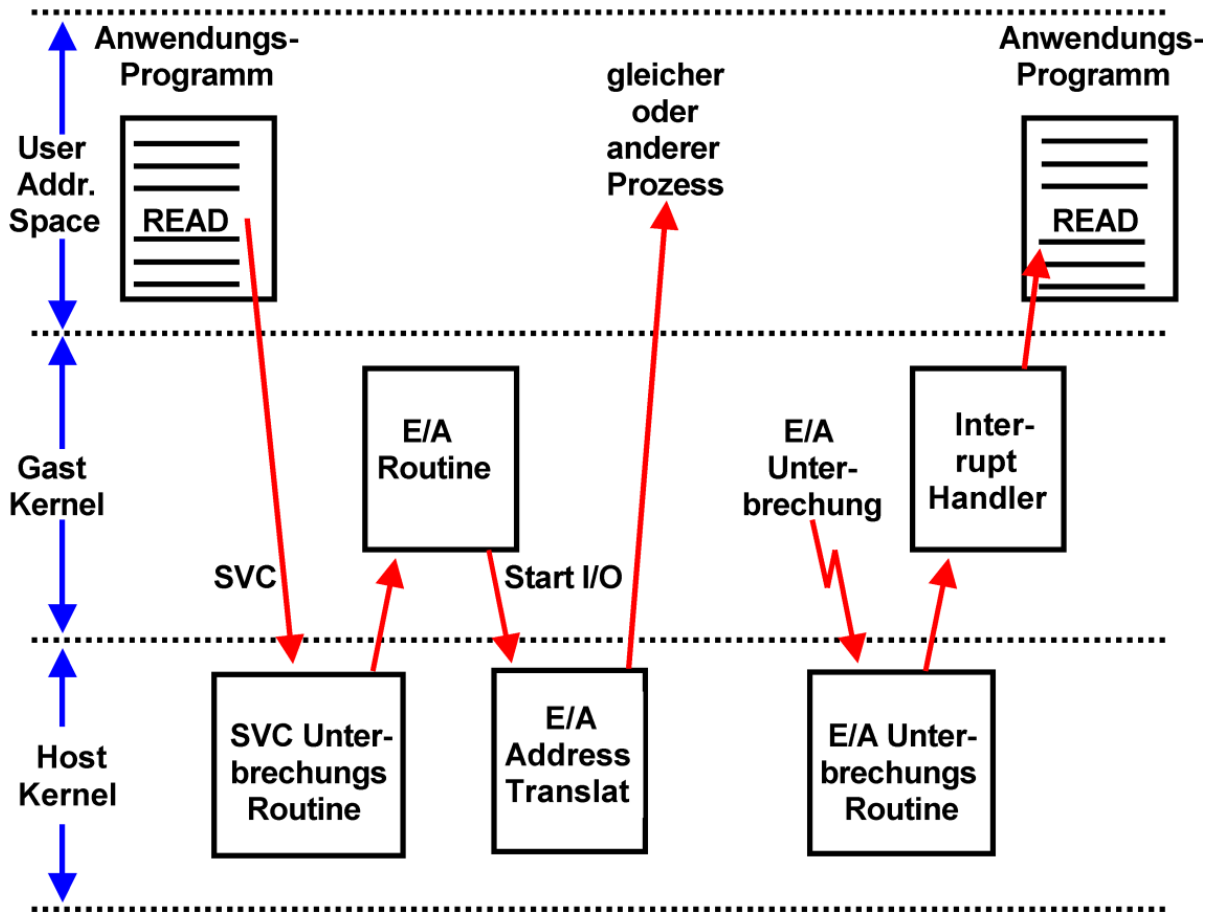


Abb. 12.1.14
Der Host Kernel führt Funktionen stellvertretend für den Gast Kernel aus

Die Ausführung einer (Plattenspeicher) READ-Operation in einem Anwendungsprogramm führt zum Aufruf einer Library Routine, welche mittels eines SVC (Supervisor Call) Maschinenbefehl in eine Routine des Kernels verzweigt. Diese führt die I/O-Operation stellvertretend für den Anwendungsprozess aus.

Eine virtuelle Maschine läuft als Prozess unter dem Host Kernel im User-Status, einschließlich des Kernels des Gastbetriebssystems. Erfolgt die READ-Operation in einer virtuellen Gast-Maschine, so bewirkt der SVC Befehl einen Aufruf nicht des Gast Kernels sondern des Host Kernels, da nur dieser im Kernel Status läuft. Nur dieser ist in der Lage, den eigentlichen I/O Driver auszuführen.

Auf der anderen Seite verwaltet der Gast Kernel die I/O Buffer oder die I/O-Steuerblöcke (z.B. DCB, ECB, UCB unter z/OS). Der Host Kernel delegiert deshalb die Erstellung der I/O Anforderung an den Gast Kernel.

Die so erstellte I/O-Anforderung arbeitet aber mit den virtuellen Adressen des Gast Kernels. Der Gast Kernel übergibt deshalb die entgültige Fertigstellung an den Host Kernel (über einen SVC Aufruf), der die I/O-Anforderung vervollständigt und an die Festplatten-Steuereinheit weitergibt.

Der Abschluss der I/O-Operation wird mit einer I/O-Unterbrechung dem Host Kernel übergeben. Dieser übergibt die Unterbrechung an den entsprechenden Unterbrechungshandler des Gastkernels. Letzterer benachrichtigt die Anwendung, welche die READ-Operation auslöste, von deren erfolgreichen Abschluss.

12.2 Host/Gast Status

12.2.1 z/VM Betriebssystem

Für Mainframe Rechner existiert seit 1972 ein spezielles Betriebssystem VM/370, welches heute fast ausschließlich für Virtualisierungsaufgaben eingesetzt wird.

Ein Vorläufer wurde 1965 unter der Bezeichnung CP/40 im IBM Scientific Center in Cambridge/Mass. entwickelt. Eine spezielle Version wurde 1967 für das Rechnermodell S/360-67 für den Vertrieb freigegeben.

Seitdem wurden zahlreiche Verbesserungen und Erweiterungen eingeführt. Von besonderer Bedeutung war die Entwicklung der „Interpretive Execution Facility“ (IEF) im Jahre 1981. Die heutige Bezeichnung des Betriebssystems ist z/VM.

Unter CP/67 und VM/370 laufen alle Anwendungen grundsätzlich auf virtuellen Maschinen. Der Host Kernel (Hypervisor) wird als CP (Control Programm) bezeichnet und läuft im Kernelstatus.

http://www.cedix.de/VorlesMirror/Band2/multicians.pdf

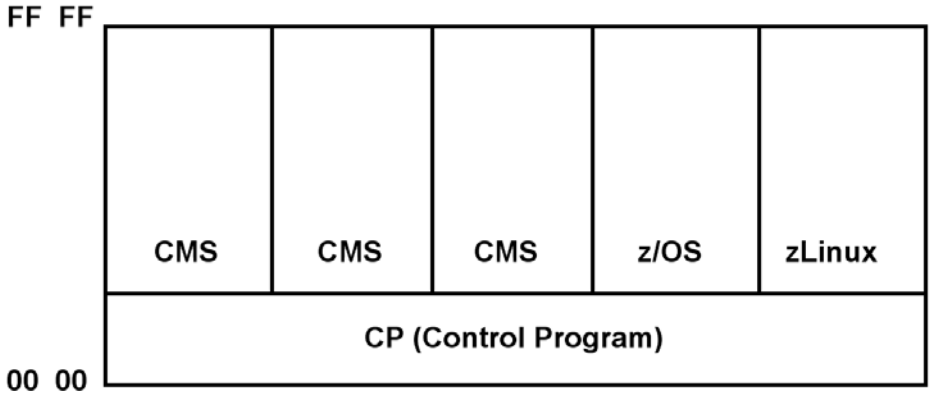


Abb. 12.2.1
Gast Betriebssysteme unter z/VM

Für CP/67 und VM/370 wurde ein eigenes Gast Betriebssystem CMS (Conversational Monitor System) entwickelt. Dieses und alle anderen Gast Betriebssysteme (einschließlich ihrer Kernel Funktionen) laufen im Problemstatus. Privilegierte Maschinenbefehle (z.B. E/A) werden von CP abgefangen und interpretativ abgearbeitet.

CMS ist ein besonders für die Software Entwicklung ausgelegtes Einzelplatz-Betriebssystem. Für 1000 gleichzeitige CMS Benutzer werden 1000 CMS-Instanzen angelegt.

VM/370 und z/VM erreichen eine uneingeschränkte S/390 und System z Kompatibilität für alle Gast-Betriebssysteme. Der Performance Verlust ist relativ gering (< 5%).

Plattenspeicherplatz wird allen Gastbetriebssystemen in der Form virtueller „Minidisks“ statisch zugeordnet. Hauptspeicherplatz wird dynamisch verwaltet.

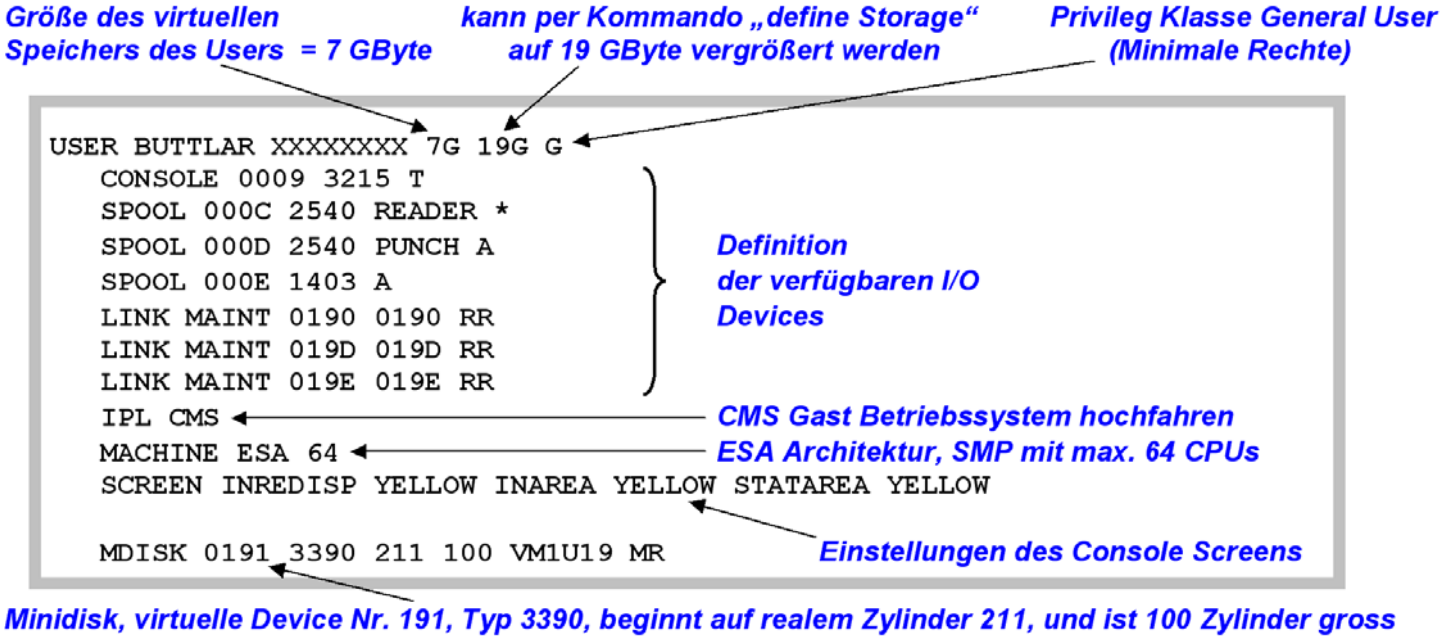


Abb. 12.2.2
z/VM Welcome Screen

Gezeigt ist der Welcome Screen, mit dem sich ein CMS User in z/VM einloggt. Die Größe des virtuellen Speichers wird für jeden Benutzer eingeschränkt, um den Verwaltungsaufwand klein zu halten. Reader und Punch sind eine Erinnerung an die früher gebräuchlichen Lochkartenleser und –stanzer. Mit dem IPL (Initial Program Load) Kommando wird das CMS Gast Betriebssystem in den virtuellen Adressenraum dieses Benutzers geladen. Es steht eine Minidisk mit der Device Nr. 191 zur Verfügung. Sie ist auf einem realen Plattenspeicher vom Typ IBM 3390 abgespeichert, beginnt auf Zylinder 211 und ist 100 Zylinder groß.

12.2.2 Virtuelle Adressumsetzung

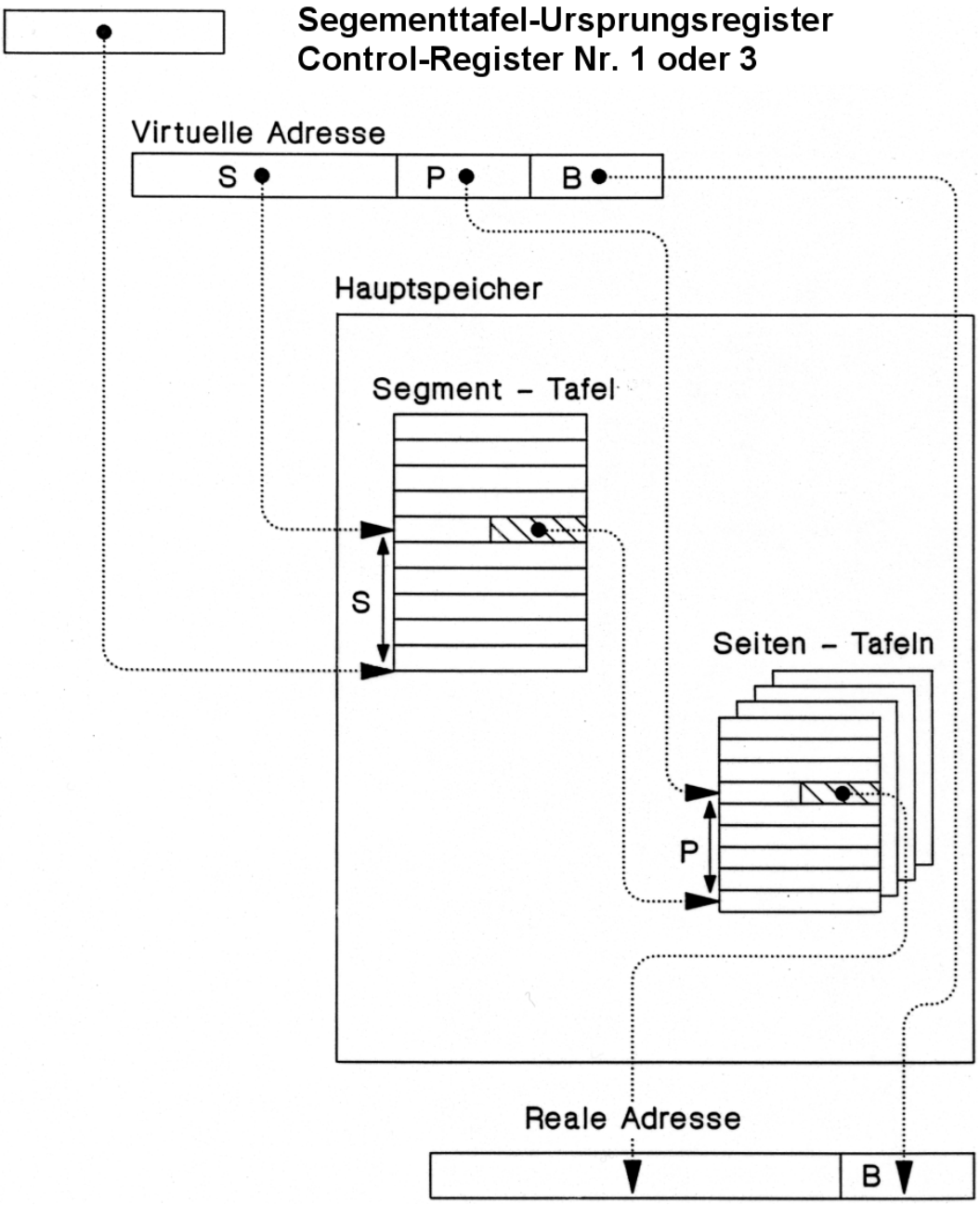


Abb. 12.2.3
Adressumsetzung Pentium, S/390

Die Adressumsetzung erfolgt durch zwei Arten von Tabellen (Segment- und Seitentabelle), die im Kernel Bereich des Hauptspeichers untergebracht sind (Siehe auch Band 1, Abschnitt 2.2.4). Die Anfangsadresse der Segmenttabelle steht in einem Control Register der Zentraleinheit, z.B. CR Nr. 1 bei der S/390 Architektur.

Die Adressumsetzung erfolgt bei jedem Hauptspeicherzugriff durch Hardware mit Unterstützung durch die Segmenttabelle und eine Seitentabelle im Kernel-Bereich. Sie kann durch den Programmierer nicht beeinflusst werden.

(Zur Leistungsverbesserung werden die derzeitig benutzten Adressen in einem Adressumsetzpuffer untergebracht.)

Problem: Das Betriebssystem einer Gastmaschine, die mit Virtueller Adressumsetzung konfiguriert ist, besteht darauf, ihre eigene Adressumsetzung durchzuführen.

12.2.3 Virtuelle Maschinen mit virtueller Adressumsetzung

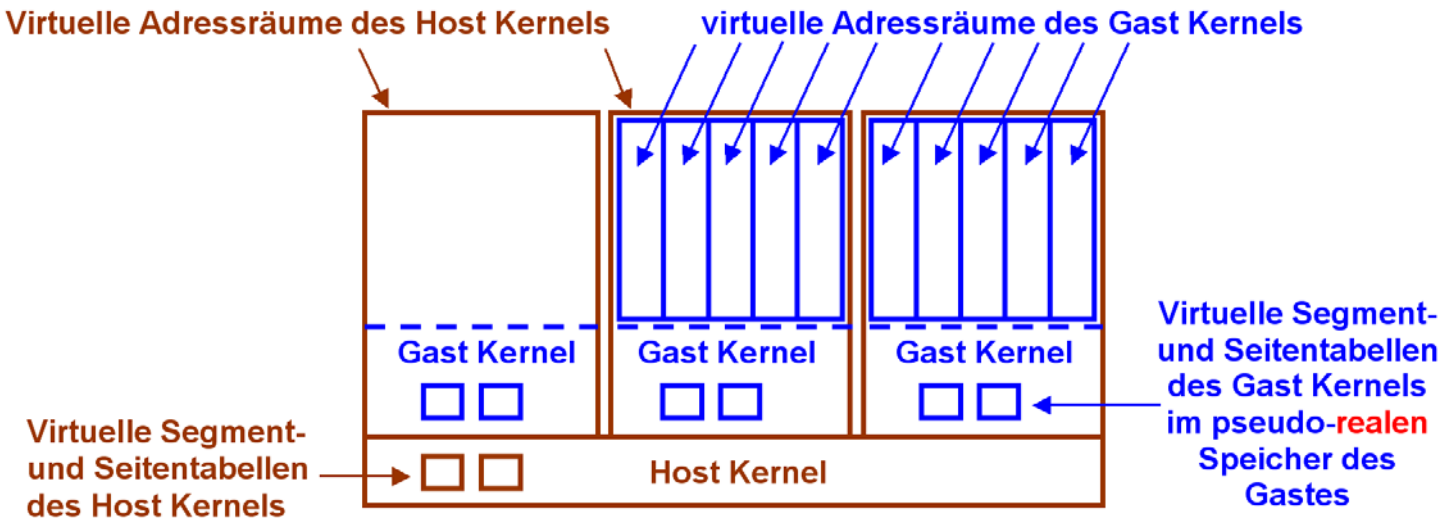


Abb. 12.2.4
Gast Betriebssysteme enthalten eine eigene virtuelle Adressumsetzung

Angenommen, wir laden ein Gast Betriebssystem in einen (und nur einen) virtuellen Adressenraum (Address Space) des Host Betriebssystems.

Bei der Einführung von VM/370 war CMS das einzige verfügbare Gast-Betriebssystem. CMS ist ein single User Betriebssystem und verwendet keine virtuelle Adressumsetzung. Das erleichtert die Implementierung.

Praktisch alle modernen Betriebssysteme verwenden eine virtuelle Adressumsetzung. Ein solches Gast Betriebssystem glaubt, der ihm zur Verfügung gestellte virtuellen Adressenraum des Host Betriebssystems ist ein realer Speicher – hier als pseudo-realer Gastspeicher bezeichnet. Es richtet in diesem realen Gastspeicher mehrfache virtuelle Adressenräume (virtuelle Gastspeicher) ein und will seine eigenen Prozesse in diesen virtuellen Adressräumen laufen lassen. Wir brauchen eine Virtualisierung der virtuellen Adressenräume des Gastbetriebssystems!!!

12.2.4 Problem der Adressübersetzung für die Virtuelle Maschine

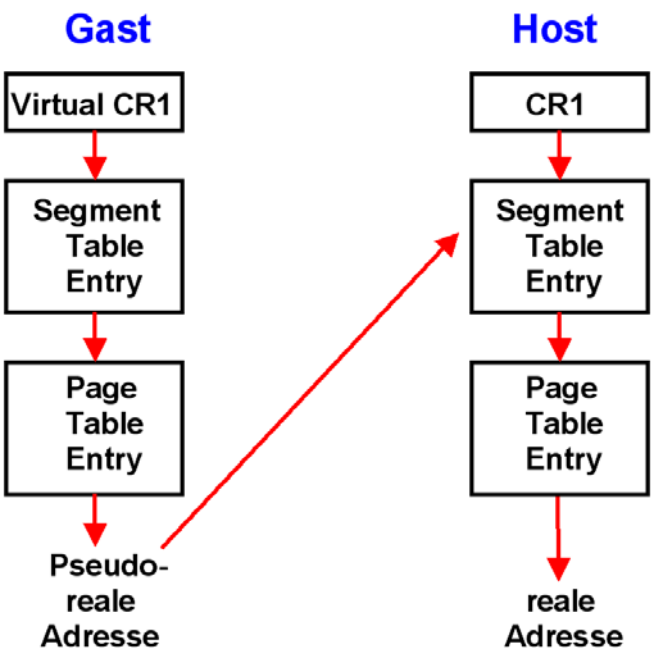


Abb. 12.2.5
Adressumsetzung wie es eigentlich laufen sollte

Das Gastbetriebssystem nimmt bei jedem Hauptspeicherzugriff eine virtuelle→ reale Adressumsetzung vor. Es unterhält innerhalb des Gast Kernel Bereiches hierfür seine eigenen Segment- und Seitentabellen. Als Ergebnis der Adressumsetzung entsteht eine Rahmenadresse im pseudo-realen Gastspeicher.

Die Rahmenadresse im pseudo-realen Gastspeicher ist aus Sicht des Host Kernels aber eine virtuelle Seitenadresse innerhalb des virtuellen Adressenraums, den der Host Kernel der Gast Maschine als pseudo-realen Speicher zur Verfügung stellt. Bei jedem Hauptspeicherzugriff benutzt der Host Kernel das Ergebnis der Adressumsetzung des Gast Kernels, um die Rahmenadresse im pseudo-realen Gastspeicher in eine echte reale Adresse zu übersetzen. Der Host Kernel verwendet für diese Adressumsetzung seine eigenen Segment- und Seitentabellen.

Da die Adressumsetzung per Hardware erfolgt, kann die doppelte Adressumsetzung nicht funktionieren!

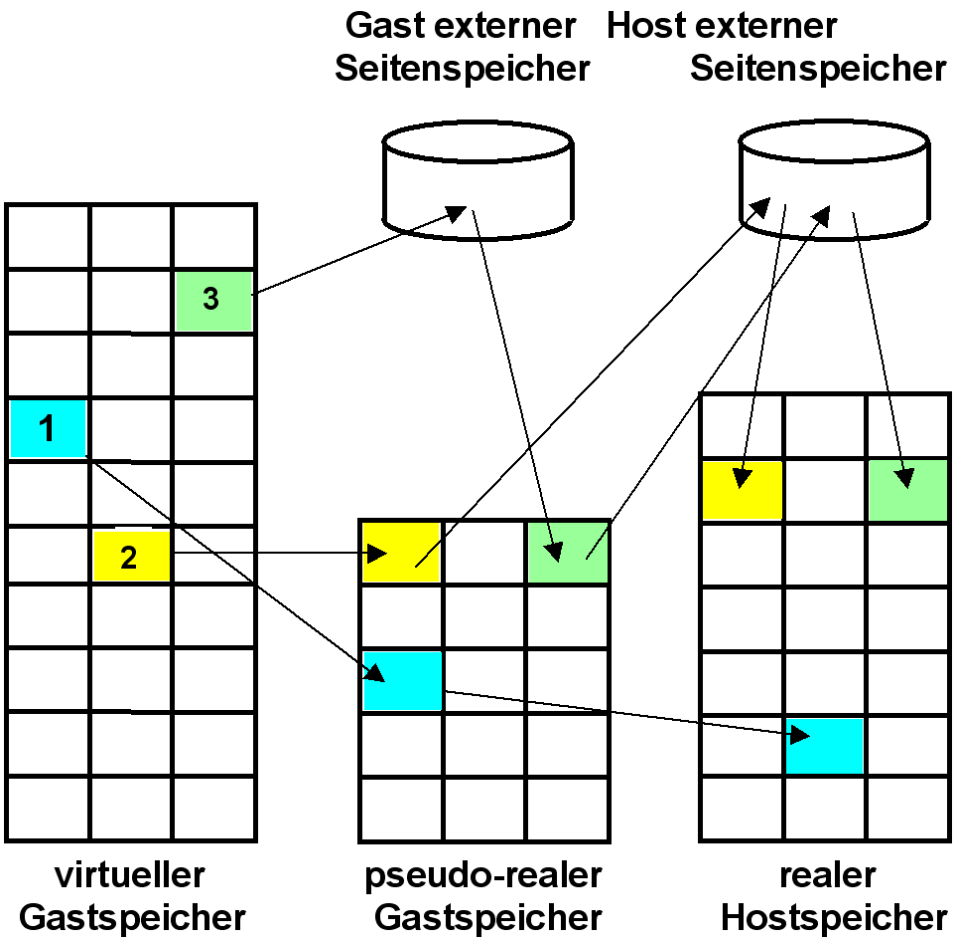


Abb. 12.2.6
Adressumsetzung zwischen Gast-Kernel und Host-Kernel

Wie erfolgt die Adressumsetzung der Seiten des virtuellen Gastspeichers in Rahmen des realen Hauptspeichers des Host System? Je nach dem Zustand der Seiten existieren drei alternative Möglichkeiten:

1. **Der gewünschte Rahmen befindet sich im (pseudo-)realen Speicher des Gastes und auch im realen Speicher des Rechners selbst (realer Hostspeicher).**

2. **Der gewünschte Rahmen befindet sich im pseudo-realen Speicher des Gastes aber nicht im realen Hostspeicher. Der Host-Kernel löst eine Fehlseitenunterbrechung aus und lädt den Rahmen in den realen Hauptspeicher.**

3. **Der gewünschte Rahmen befindet sich nicht im pseudo-realen Speicher des Gastes. Der Gast-Kernel löst eine Fehlseitenunterbrechung aus, die vom Host-Kernel abgefangen wird. Dieser lädt den Rahmen in den realen Hauptspeicher und gleichzeitig auch in den realen Gastspeicher.**

Die virtuelle Adressumsetzung mit Hilfe von Segment- und Seitentafeln erfolgt durch Hardware und kann durch Software nicht beeinflusst werden. Das hier geschilderte Verfahren kann deshalb nicht funktionieren.

Die Anfangsadresse der Segmenttafel steht im Control Register 1 (Mainframe) oder Control Register 3 (x86 Architektur). Für die Gast Maschine ist dies ein virtuelles Control Register, welches den Pointer für die Lokation der Segmenttabelle im Gast-Kernel Bereich enthält. Die Information in dem virtuellen Control Register steht aber nicht in dem physischen Control Register, sondern wird vom Host Kernel irgendwo gespeichert. Mit diesem virtuellen Control Register kann somit die Adressumsetzungs-Hardware nichts anfangen.

Erforderlich wäre eine Änderung der Hardware-Architektur, welche die hier beschrieben doppelte Adressumsetzung ermöglicht. Dies war jedoch in der ursprünglichen x86 (Pentium) Architektur nicht vorgesehen.

Zur Lösung des Problems führte man die sogenannten „Shadow Tables“ ein. Shadow Tables duplizieren teilweise die Einträge in den Segment- und Seitentabellen des Gasts und des Hosts, und müssen mit diesen im laufenden Betrieb synchron gehalten werden.

12.2.5 Shadow Page Tables unter VM/370 und VMware

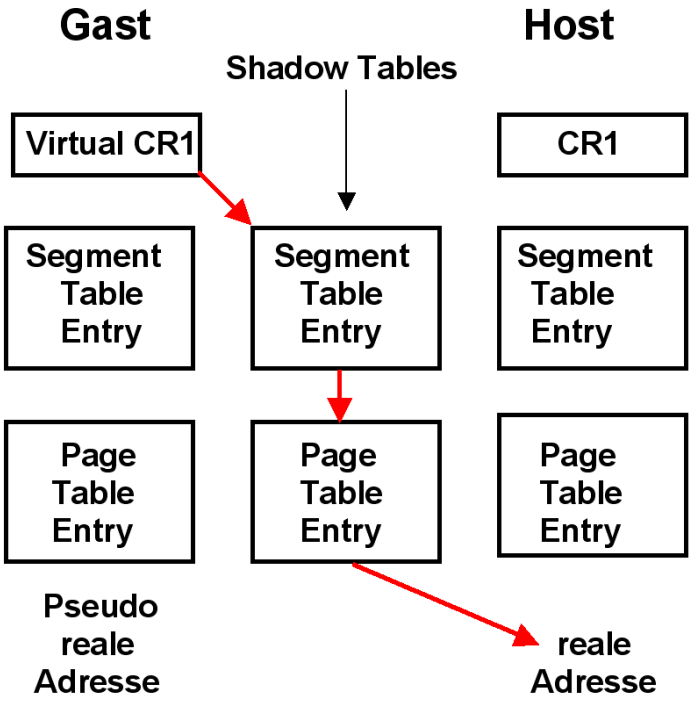


Abb. 12.2.7
Shadow Page Tables an Stelle einer zweistufigen Adressumsetzung

VM/370 und VMware erstellen anhand der Gast- und der eigenen Segment- und Seitentabellen sogenannte *Shadow Tables*, die direkt die virtuellen Adressen des Gastes auf reale Adressen des Hosts abbilden. Diese Tabellen liegen im Speicherbereich des Host-Kernels.

Wenn immer eine Gast Maschine (in Wirklichkeit ein Prozess des Host Kernels) die Verfügungsgewalt über die CPU bekommt, muss der Host Kernel die Shadow Tables mit neu berechneten korrekten Werten füllen. Bei jeder Änderung in den Segment- und Seitentabellen einträgen des Gastes werden diese Änderungen in den Shadow Tables nachvollzogen. Der Vorteil des Verfahrens ist, es funktioniert. Nachteilig ist, es kostet viel Performance.

12.2.6 Sensitive Maschinenbefehle

Es muss verhindert werden, dass bei der Ausführung eines Maschinenbefehls innerhalb einer virtuellen Maschine das Verhalten einer anderen virtuellen Maschine beeinflusst wird.

Die Ausführung von nicht-sensitiven Maschinenbefehlen beeinflusst nicht das Verhalten einer anderen virtuellen Maschine.

Die Ausführung von sensitiven Maschinenbefehlen kann das Verhalten einer anderen virtuellen Maschine beeinflussen.

Einfachste Implementierung: Alle sensitiven Maschinenbefehle sind gleichzeitig auch privilegierte Maschinenbefehle und können nur vom Host-Kernel ausgeführt werden. So wurde VM/370 in 1971 implementiert.

Die Firma VMWare portierte VM/370 auf die x86-Plattform und lieferte ihr erstes Produkt in 1999 aus. Unglücklicherweise war die x86-Architektur hierfür wesentlich schlechter als die damalige S/370 Architektur geeignet:

Many models of Intel's machines allow user code to read registers and get the value that the privileged code put there instead of the value that the privileged code wishes the user code to see.

G.J. Popek, R.P. Goldberg: Formal Requirements for Virtualizable Third Generation Architectures. Comm. ACM, Vol. 17, Nr. 7, Juli 1974, S. 412-421.
http://www.cap-lore.com/CP.html

Im Vergleich zu VM/370 sind die VMware ESX und GSX Server benachteiligt, weil einige kritische Eigenschaften in der x86-Architektur fehlen. Für den Betrieb von Gast-Maschinen ist es erforderlich, dass alle Maschinenbefehle, welche den privilegierten Maschinenstatus abändern oder auch nur lesen, nur im Kernel Status ausgeführt werden können.

Wenn ein Gast in ein Control Register schreibt, muss der Host Kernel diesen Maschinenbefehl abfangen, damit nicht das reale Kontrollregister des Hosts verändert wird. Der Host Kernel wird jetzt nur die Effekte der Instruktion für diesen Gast simulieren. Liest der Gast anschließend diese Kontrollregister wieder aus, so muss diese Instruktion ebenfalls abgefangen werden, damit der Gast wieder den Wert sieht, den er vorher in das Register geschrieben hat (und nicht etwa den realen Wert des Kontrollregisters, der nur für den Host sichtbar ist).

Da die x86-Architektur diese Bedingung ursprünglich nicht erfüllte, war es nicht möglich, wie unter VM/370, alle Maschinenbefehle einfach im User Mode auszuführen, und auf Programmunterbrechungen zu vertrauen wenn auf privilegierten Maschinenstatus Information zugegriffen wird.

VMware's ESX Server überschreibt hierzu dynamisch Teile des Gast-Kernels und schiebt Unterbrechungsbedingungen dort ein, wo eine Intervention des Host-Kernels erforderlich ist. Als Folge hiervon tritt ein deutlicher Leistungsverlust auf, besonders bei Ein-/Ausgabe-Operationen. Manche Funktionen sind nicht vorhanden oder können nicht genutzt werden.

Kompatibilitätsprobleme treten auf; es kann sein, dass bestimmte Anwendungen nicht lauffähig sind.

12.2.7 Paravirtualization

Hierbei ist die Betriebssystem-Architektur der virtuellen Maschine nicht vollständig mit der Host Betriebssystem-Architektur identisch. Die Benutzerschnittstelle (Application Binary Interface, ABI) ist die gleiche. Der Gast-Kernel unterstellt jedoch Abweichungen zu der x86-Architektur. Dies verbessert das Leistungsverhalten, erfordert aber Änderungen des Gast-Kernels. Hiervon ist nur ein sehr kleiner Teil des Kernel-Codes betroffen. Beispielsweise existiert ein funktionsfähiger Linux-Port (XenoLinux), dessen ABI mit dem eines nicht-virtualisierten Linux identisch ist.

Ein ähnlicher Ansatz wird von *Denali verfolgt*. Als Gast-Betriebssystem dient *Ilwaco*, eine speziell an den Denali Hypervisor angepasste BSD Version. Denali unterstützt nicht das vollständige x86 ABI. Es wurde für Netzwerk-Anwendungen entwickelt und unterstellt Einzelbenutzer-Anwendungen. Mehrfache virtuelle Adressräume sind unter Ilwaco nicht möglich.

12.2.8 Die Entwicklung der Virtuellen Maschinen

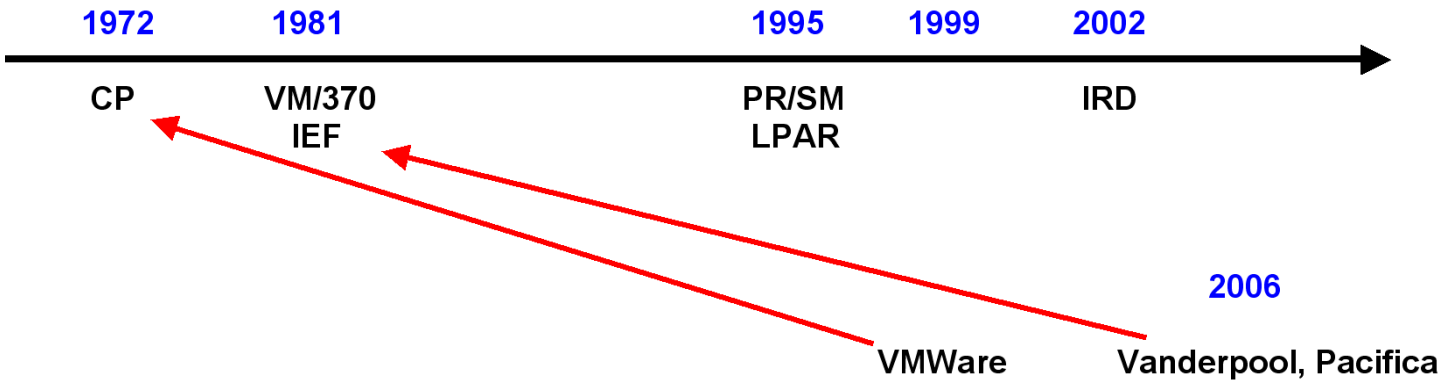


Abb. 12.2.8
Die Entwicklung der Virtualisierung

Das VM/370 Betriebssystem mit dem CP Hypervisor und dem CMS Gast-Betriebssystem ist seit 1972 verfügbar. 1995 gründeten einige VM/370 Entwickler die Firma VMware. Seit 1999 ist VMWare für die x86 Plattform verfügbar.

Bereits 1981 hatte IBM für alle S/370 Rechner eine Erweiterung der Hardware Architektur eingeführt, die als „Interpretive Execution Facility“ (IEF) bezeichnet wurde, und zu einer erheblichen Leistungsverbesserung der virtuellen Maschinen führte. Eine vergleichbare Architekturerweiterung wurde von Intel und AMD im Jahre 2006 für die x86 (Pentium) Architektur unter dem Namen Vanderpool bzw. Pacifica eingeführt.

VT-x (Virtualization Technology for x86) ist eine alternative Bezeichnung für Vanderpool. Eine ähnliche Einrichtung für Itanium hat den Namen VT-i. AMD Virtualization, abgekürzt AMD-V ist eine alternative Bezeichnung für Pacifica.

Bereits vorher im Jahre 1995 hatte die S/390 Architektur eine weitere grundlegende Verbesserung unter dem Namen „Logische Partition“ (LPAR) eingeführt. Dies wurde 2002 durch eine nochmalige Verbesserung, den „Intelligent Resource Director“ (IRD) ergänzt. Der PowerPC führte LPARs 2002 ein.

Experten gehen davon aus, dass x86 weitere 10 Jahre benötigen wird, um eine mit LPAR und IRD vergleichbare Funktionalität zu erreichen.

12.2.9 Host- und Gast Modus

	Überwacher Status	Problem Status
Host Modus	+	—
Gast Modus	+	+

Abb. 12.2.9
Mögliche Zustände beim Einsatz des Host/Gast Modus

Beim heutigen Ansatz unterscheidet jeder Rechner zwischen dem Überwacher Status und dem Problem Status. Bestimmte sog. „Privilegierte Maschinenbefehle“ können im Überwacher Status, aber nicht im Problem Status ausgeführt werden.

Jeder Rechner unterscheidet zwischen Überwacher Status (supervisor state) und Problem Status (Problem state). Das unterschiedliche Verhalten wird durch ein Bit im Status Register der CPU (Program Status Word, PSW) definiert.

Zur Lösung von Virtualisierungs-Schwierigkeiten führte man zusätzlich einen Host- bzw. Gast Modus als Architekturerweiterung ein. Mainframes bezeichnen diese Erweiterung als „Interpretive Execution Facility“ (IEF). Die entsprechenden Bezeichnungen sind VT-x (Vanderpool) bei Intel bzw. AMD-V (Pacifica) bei AMD. Obwohl die drei Ansätze sehr ähnlich sind, unterscheiden sie sich in einigen Details. Besonders VT-x und AMD-V sind nicht miteinander kompatibel.

Interpretive Execution Facility, Vanderpool und Pacifica führten eine Unterscheidung Host Modus/Gast/Modus mit Hilfe eines zweiten Bits im Statusregister ein. Damit sind die Kombinationen Host Modus/Überwacher Status. Gast Modus/Überwacher Status und Gast Modus/Problem Status möglich. Der Hypervisor läuft grundsätzlich im Host Modus; die Kombination Host Modus/Problem Status hat in Bezug auf virtuelle Maschinen keine Bedeutung.

Das Problem der sensitiven Maschinenbefehle kann nun dadurch gelöst werden, dass die Hardware sich im Gast Modus anders verhält als im Host Modus. Auch kann die Gast Maschine in ihrem Kernel Modus solche privilegierten Maschinenbefehle direkt selbst ausführen, die nicht sensitiv sind – sie braucht hierfür nicht mehr die Unterstützung des Host Kernels.

Vanderpool bezeichnet den Host Überwacher Status, in dem der Host Kernel (Virtual Machine Monitor, VMM) ausgeführt wird, als VMX Root Operation. Jeder Gast läuft in VMX non-Root Operation.

Der Aufruf eines Gastes erfolgt durch einen VM Entry Befehl (VM Run bei AMD VT, SIE bei VM/370). Die Rückkehr zum VMM erfolgt durch einen VM Exit Befehl.

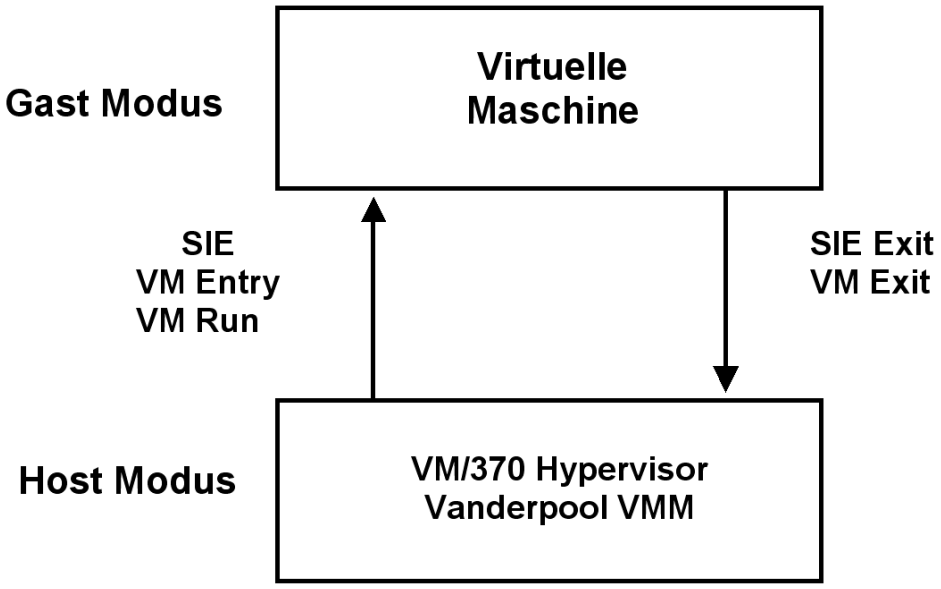


Abb. 12.2.10
Aufruf der Gastmaschine durch den Host

Im Gastmodus können privilegierte, aber nicht sensitive Maschinenbefehle vom Kernel des Gast-Betriebssystems abgearbeitet werden. Das Problem der sensitiven nicht-privilegierten x86 Architektur der x86 Architektur wird durch zusätzliche Vanderpool- bzw. Pacifica-Hardware gelöst, die diese Befehle bei ihrer Dekodierung identifiziert und zwecks Ausführung an den VMM übergibt.

Der Kernel des VM/370 Betriebssystems ist gleichzeitig der Hypervisor. Der Rechner läuft immer im Virtualisierungsmodus.

Der Vanderpool Virtual Maschine Monitor (VMM) ist eine Erweiterung zu einem vorhandenen Betriebssystem-Kernel, z.B. dem Windows XP-Kernel. Beide zusammen bilden den Hypervisor. Anwendungen können deshalb wahlweise im Virtualisierungsmodus unter einem Gastbetriebssystem oder nicht virtualisiert unmittelbar unter dem Host Betriebssystem laufen.

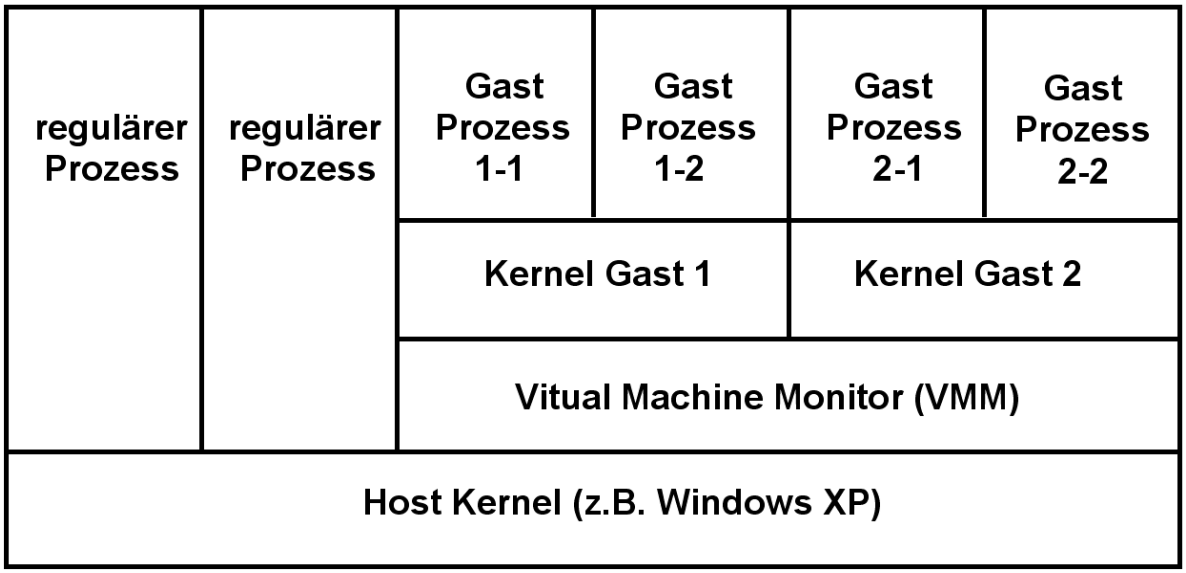


Abb. 12.2.11
Vanderpool oder Pacifica für den PC

Der Rechner befindet sich entweder im Virtualisierungsmodus oder auch nicht. Im Virtualisierungsmodus ist er in der Lage, „Virtual Machine Execution“ (VMX)-Operationen durchzuführen und mit Gast-Maschinen zu arbeiten. Ein VMXON Befehl versetzt den Rechner in den Virtualisierungsmodus und aktiviert den Virtual Machine Monitor; mit VMXOFF wird der Virtualisierungsmodus wieder verlassen.

Existierende Virtual Machine Monitore wie VMware, KVM, XEN, Virtual PC, Virtual Box usw. nutzen die Vanderpool bzw. Pacifica Technologie. Damit entfallen die meisten bisherigen x86-Probleme.

12.2.10 Shadow Page Tables

Mit der Einführung der Interpretive Execution Facility, Vanderpool bzw. Pacifica kann die Rechner-Hardware für die Bedürfnisse der Virtuellen Maschinen angepasst und erweitert werden. Spezifisch sind damit die Einführung einer doppelten Adressumsetzung und die Eliminierung von Shadow Page Tables möglich.

Hiermit werden die häufigen Updates der Shadow Page Tables eliminiert. Nachteilig sind die zusätzlichen Zugriffe auf Segment- und Pagetables im Hauptspeicher. Der Aufwand hierfür ist in der Regel aber nicht so groß wie der Aufwand für die Shadow Page Table Updates. Der Grund ist, die am häufigsten benutzten Einträge in den Segment- und Seitentabellen werden in einem als DLAT (Directory Look Aside Table) oder TLB (Translation Lookaside Buffer) bezeichneten Cache Speicher abgelegt. Wenn man diesen ausreichend dimensioniert (Tausende von Einträgen), ist der Leistungsabfall vernachlässigbar.

Ende 2007 führte AMD für seine x86-Prozessoren eine neue Virtualisierungstechnologie Rapid Virtualization Indexing (RVI) – früher als Nested Page Tables bezeichnet – ein. Das Pendant Extended Page Tables wurde ein Jahr später von Intel mit den Nehalem-Prozessoren (erste Core i-Generation, i9xx) eingeführt. Mit RVI wird der virtuelle Adressraum eines Gastes direkt auf einen realen Adressraum des Hosts abgebildet. Dabei wird die Shadow Page Table durch eine Nested Page Table ersetzt, die vom Prozessor in Hardware unterstützt wird. Dadurch ist auch keine doppelte Adressumsetzung mehr notwendig. Eine Untersuchung von VMWare ergab, dass mit RVI ein Performancegewinn von bis zu 42% im Gegensatz zur Verwendung von softwarebasierten Shadow Page Tables erzielt werden kann. Red Hat hat mit Verwendung von RVI eine Performanceverdopplung für OLTP-Benchmarks (Online Transaction Processing) beobachtet.

Weitere Informationen: AMD-V Nested Paging, White Paper, 2008:
http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf

www.cedix.de/VorlesMirror/Band2/AMDshadow.pdf

12.3 Logical Partition

12.3.1 Firmware

Wenn Sie einen Access Point für Ihr privates WLAN kaufen, erhalten Sie eine Box, die u.a. einen Mikroprozessor mit umfangreichem Code (häufig in EPROMs gespeichert) enthält. Das Betriebssystem ist in vielen Fällen eine Linux-Variante.

Als Benutzer können sie ein derartiges Gerät konfigurieren (z.B. eine IP Adresse eingeben). Sie können es jedoch nicht programmieren oder eigenen Code installieren.

Ein derartiges Gerät könnte ganz in Hardware implementiert sein, ohne dass Sie den Unterschied wahrnehmen würden. Mikroprozessor-Code mit derartigen Eigenschaften (vom Benutzer nicht programmierbar, änderbar oder einsehbar) wird als Firmware bezeichnet. Auch ein Administrator kann in der Regel Firmware nicht einsehen oder modifizieren, ggf. aber Konfigurierungs-Maßnahmen durchführen. Beispielsweise kann ein Internet-Router einen in Firmware implementierten Browser enthalten, den ein Administrator benutzen kann, um Einstellungen vorzunehmen.

System z benutzt neben den CPUs weitere, unsichtbare Prozessoren, welche den Funktionsablauf steuern. Diese Prozessoren werden – im Gegensatz zu den CPUs – als „System Assist Prozessoren“ (SAPs) bezeichnet. Beispielsweise kann ein zEC12 Rechner 101 CPUs und 16 SAPs enthalten.

Auf SAPs können keine Anwendungen laufen, und ein Benutzer kann keine Programme für SAPs schreiben. SAPs werden z.B. für die Ein/Ausgabe Steuerung, für Fehler-Behandlungsroutinen – und für Virtualisierungszwecke – benutzt.

Der von den SAPs ausgeführte Code wird als Firmware bezeichnet. (Der Coupling Facility Control Code (CFCC) ist ebenfalls Firmware). Firmware wird ähnlich wie Microcode als Bestandteil der Hardware betrachtet. Der Unterschied zwischen Firmware und Microcode besteht darin, dass Firmware reguläre System z-Maschinenbefehle unterstützt und ausführt. Der Firmware-Code befindet sich in einem getrennten Speicher, der als Hardware System Area (HSA) bezeichnet wird. Firmware wird von IBM auch als „Licensed Internal Code“ (LIC) bezeichnet. Itanium (und DEC Alpha) verwendet statt LIC die Bezeichnung PAL (Privileged Architecture Library). Itanium enthielt ursprünglich einen in PAL implementierten x86 Emulator.

Heutige Implementierungen verwenden einen einzigen physischen Speicher, der in zwei getrennte logische Speicher aufgeteilt wird:

- **Einen realen Speicher, der das Betriebssystem aufnimmt und im dem die virtuellen Speicher der Benutzerprozesse abgebildet werden.**
- **Eine Hardware System Area (HSA), die Firmware aufnimmt.**

Die von den SAPs und ihrem HSA Code ausgeführte Ein/Ausgabe-Steuerung wird als Channel Subsystem bezeichnet. Es bildet das virtuelle I/O-Subsystem, mit dem der Betriebssystem-Kernel glaubt zu arbeiten, auf die reale I/O-Struktur ab. Unabhängig von System- und Benutzercode sind damit umfangreiche Optimierungen der Plattenspeicherzugriffe möglich.

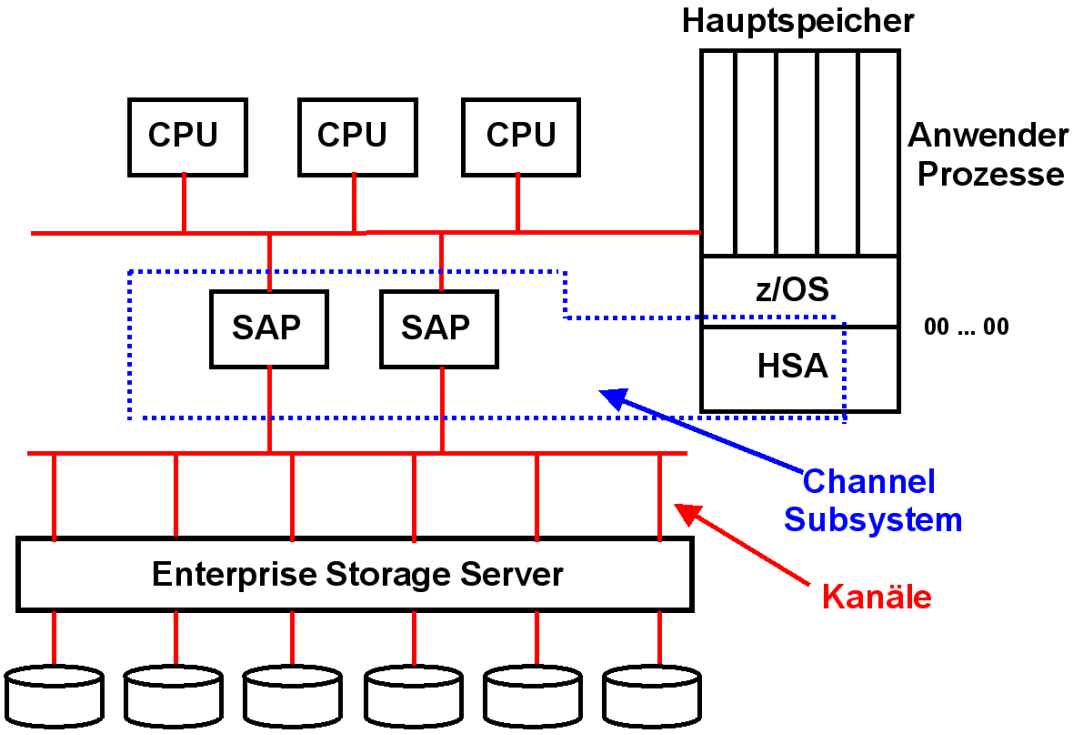


Abb. 12.3.1
Nutzung der HSA durch SAPs

Die HSA (Hardware System Area) ist ein Teil des System z physischen Hauptspeichers. Sie liegt außerhalb des Adressenraums, auf den die CPUs zugreifen können. Der zEC12 Rechner hat eine 32 GByte große HSA.

Der in der HSA untergebrachte Code wird ähnlich wie der in der Coupling Facility laufende Code als Firmware bezeichnet. Er wird von SAPs ausgeführt.

In der HSA ist Firmware Code für die I/O Ablaufsteuerung, dazugehörige Datenstrukturen sowie Code für umfangreiche Diagnostische und Fehlerbehandlungsfunktionen. SAPs führen z.B. Channel Subsystem Code aus oder implementieren Diagnostische oder Fehlerbehandlungs-Aufgaben. Auch der PR/SM Hypervisor Code wird von SAPs ausgeführt.

12.3.2 Logical Partition und PR/SM

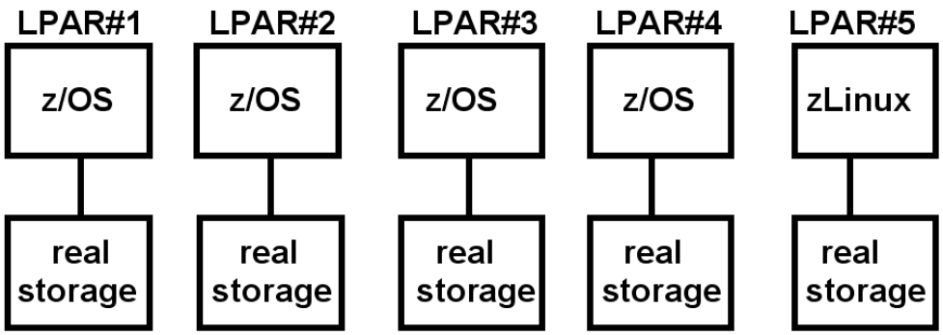


Abb. 12.3.2
LPARs laufen – im Gegensatz zu virtuellen Maschinen – in realen Adressräumen

Bei den virtuellen Maschinen wird das Gast-Betriebssystem mit seinen Anwendungen in einem virtuellen Adressenraum des Host-Kernels abgebildet. Nach erfolgter virtueller Adressumsetzung teilen sich alle Gast-Systeme einen einzigen realen Adressenraum. Die Adressumsetzung bedingt Shadow Tables oder eine doppelte Adressumsetzung wie im Fall von Pacifica und der z/VM Interpretive Execution Facility. Eine direkte Zuordnung des realen Adressraumes zu einem Gast wird auch mit AMDs RVI / Intels EPT auf x86-Systemen durchgeführt.

Die System z „PR/SM“ (Processor Resource/System Manager) Einrichtung verwendet ein anderes, als LPAR (Logical Partition) bezeichnetes Verfahren. PR/SM (ausgesprochen pri`sm) hat die Funktion eines Hypervisors (Host Kernel). PR/SM ist als Firmware implementiert, die in der HSA untergebracht ist und von SAPs ausgeführt wird. Für jedes Gastbetriebssystem wird eine eigene LPAR eingerichtet. Jede LPAR verfügt über einen eigenen getrennten realen Speicher. Die PR/SM Hardware stellt mehrfache (bis zu 60) LPARs mit getrennten realen Speicher zur Verfügung.

Da jede LPAR über einen eigenen realen Speicher verfügt, sind Shadow Tables oder eine doppelte Adressumsetzung nicht erforderlich.

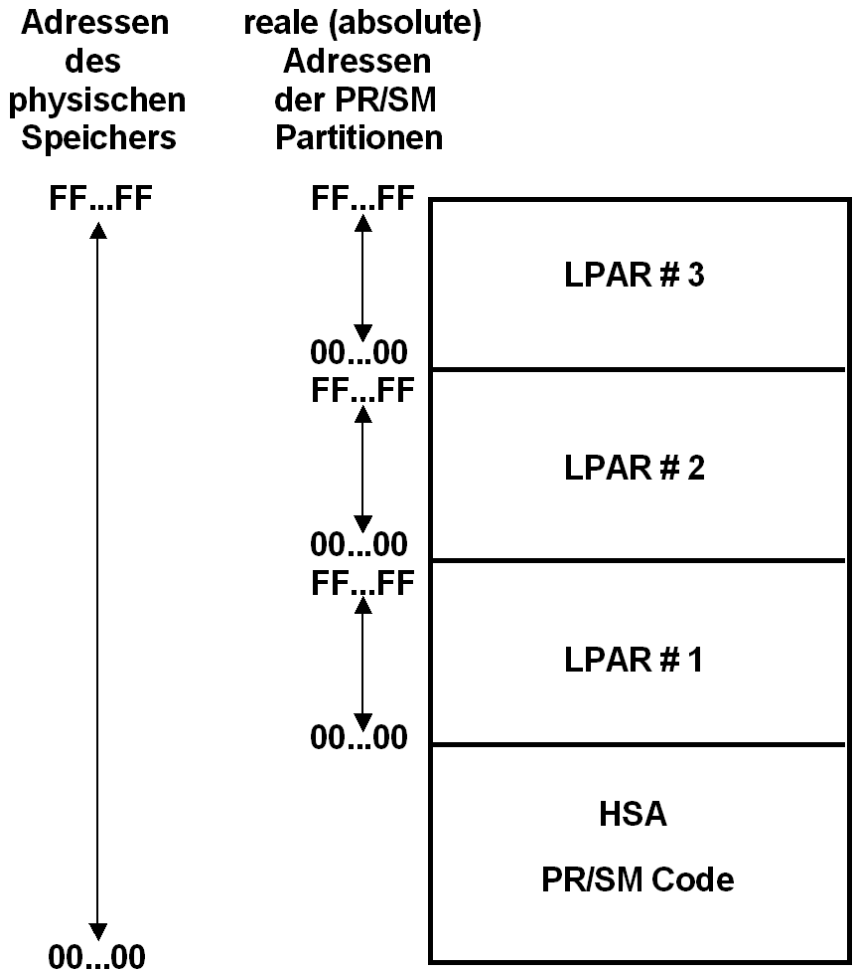


Abb. 12.3.3
Aufteilung des physischen Speichers in mehrere reale Speicher

Es wäre denkbar, jeder LPAR einen getrennten physischen Hauptspeicher zuzuordnen. System z Rechner verwenden statt dessen einen einzigen physischen Hauptspeicher, der mittels spezieller Logik in einzelnen Partitionen aufgeteilt wird. Jede LPAR erhält eine eigene Hauptspeicher-Partition, deren kontinuierlicher Adressenbereich mit der Adresse 00 .. 00 anfängt.

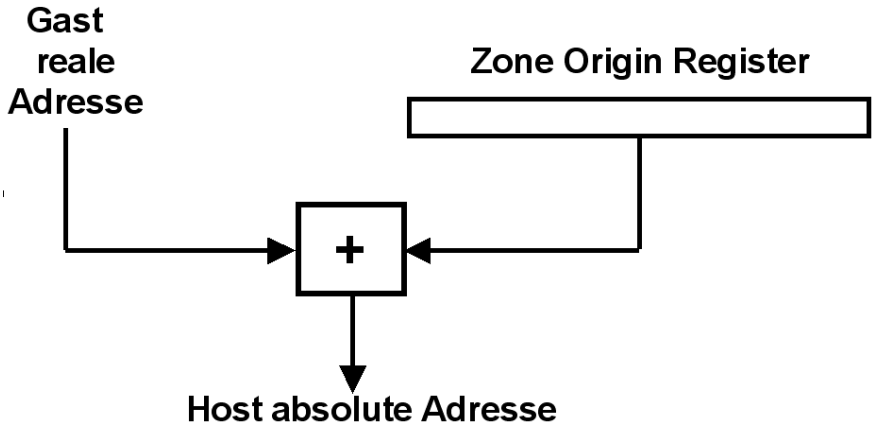


Abb. 12.3.4
Umsetzung der Gast Adressen in physische Adressen

Jede CPU hat ein LPAR Relocate Register, als LPAR Zone Origin Register bezeichnet. Hier ist die Anfangsadresse des Bereiches im physischen Speicher enthalten, der dieser LPAR zugeordnet ist.

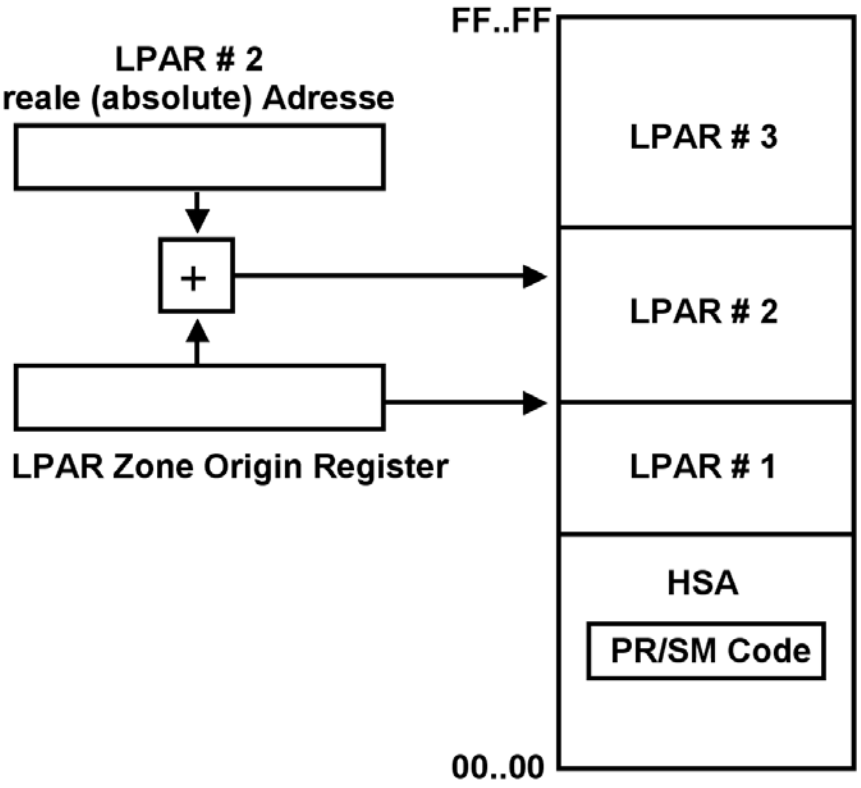


Abb. 12.3.5
LPAR Speicherplatz-Verwaltung

Zusätzlich existiert ein Zone Limit Register, welches die maximale reale Hauptspeicher-Adresse definiert, die dieser LPAR zugeordnet ist.

12.3.3 Aufteilung eines Großrechners in mehrere SMPs

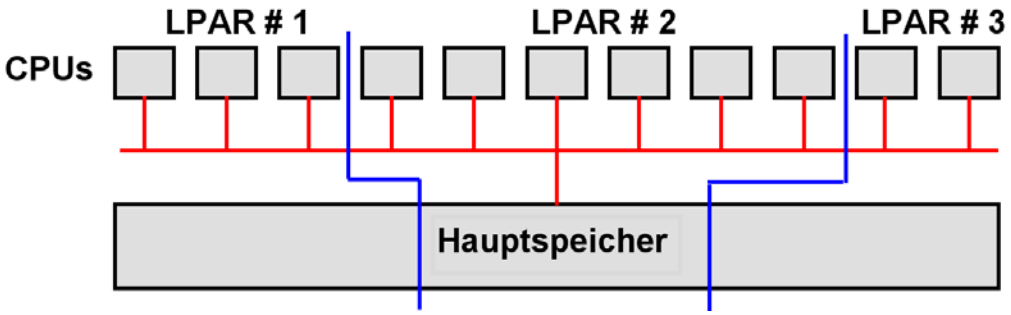


Abb. 12.3.6
Die Aufteilung des Hauptspeichers kann konfiguriert werden

z/OS unterstützt symmetrische Multiprozessoren (SMP) mit bis zu 101 CPUs. Realistisch sind bis zu 24 – 32 CPUs. Bei Unix, Linux und Windows Betriebssystemen liegt die Grenze für Transaktions- und Datenbankanwendungen eher bei 12-16 CPUs.

Moderne Großrechner (Systeme) verfügen über wesentlich mehr CPUs. Sie werden deshalb in mehrere SMPs aufgeteilt, die über einen zentralen Switch miteinander kommunizieren.

Die CPUs werden im einfachsten Fall auf die LPARs aufgeteilt und diesen fest zugeordnet.

Der Systemadministrator kann den gesamten Hauptspeicher in unterschiedlichen Größen auf die einzelnen LPARs aufteilen.

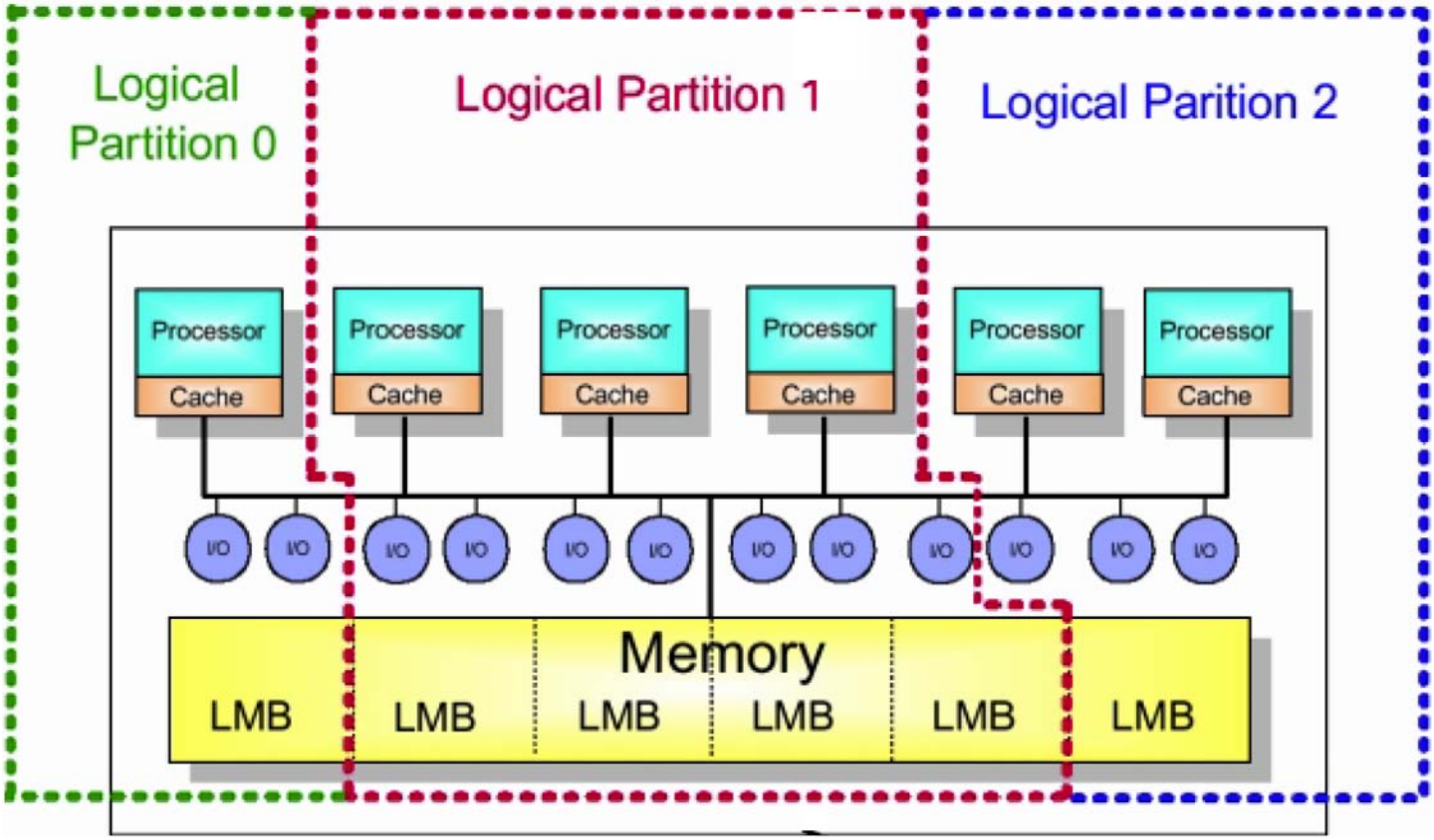


Abb. 12.3.7
Hauptspeicher Zuordnung in logischen Memory Blöcken

Die Aufteilung des physischen Speichers auf die einzelnen LPARs erfolgt (im einfachsten Fall) statisch, und wird bei der erstmaligen Inbetriebnahme des Systems festgelegt.

Der vorhandene physische Hauptspeicher besteht aus 64 MByte großen „Logischen Memory Blöcken“ (LMB). Die feste Aufteilung des vorhandenen physischen Hauptspeichers erfolgt in LMB Einheiten.

Die vom Channel Subsystem verwalteten I/O Anschlüsse (Kanäle) und die damit verbundenen Platten- und Magnetbandspeicher werden im einfachsten Fall den einzelnen LPARs fest zugeordnet. Das Gleiche gilt z.B. auch für Ethernet Anschlüsse. Hat ein Rechner z.B. 6 Ethernet Adapter, so kann man der ersten LPAR vier Adapter,der zweiten LPAR null Adapter und der dritten LPAR zwei Adapter zuordnen.

12.3.4 zLinux auf dem Mainframe

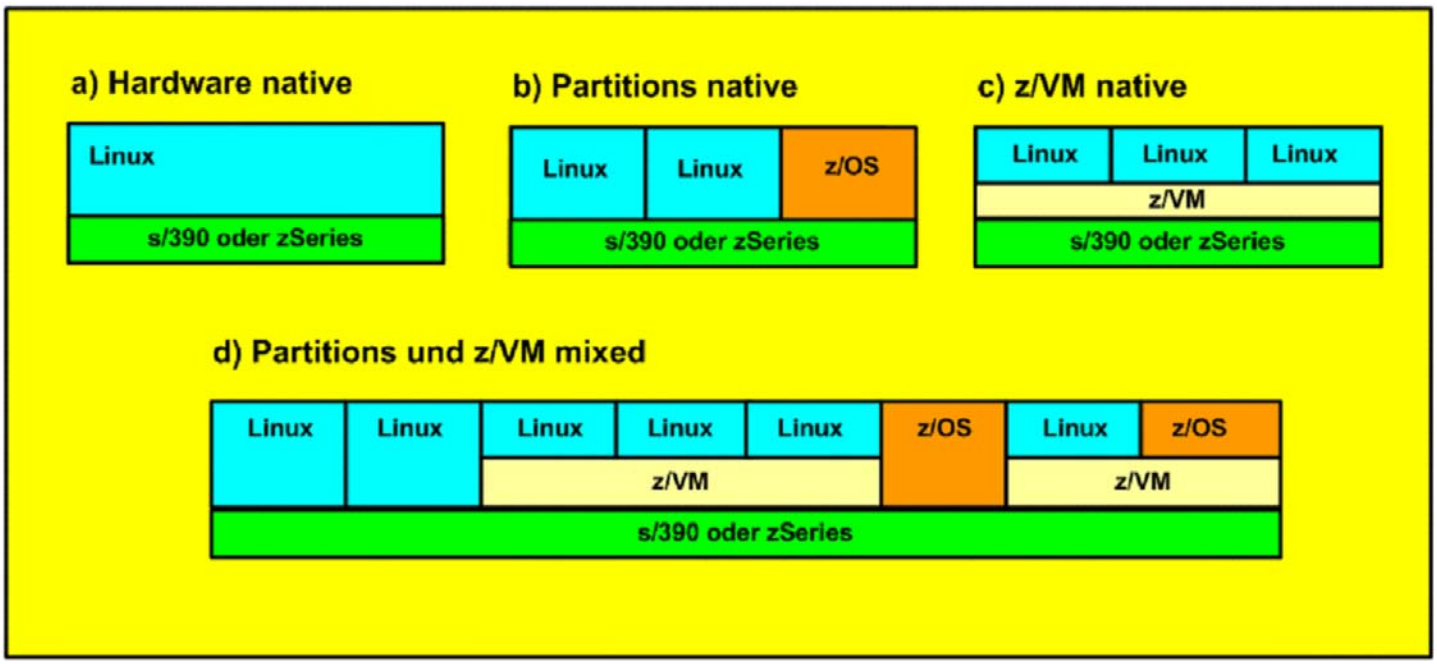


Abb. 12.3.8
In den LPARs können unterschiedliche Betriebssysteme laufen

zLinux ist ein populäres Mainframe Betriebssystem. Es ist ein normaler Port auf die System z Platform, und verhält sich dort genauso wie ein Linux auf der x86 oder PowerPC Hardware. Dies bedeutet auch, viele System z Funktionen werden nicht benutzt. Suse und Red Hat vertreiben zLinux Distributionen.

zLinux wird aber fast immer zusätzlich zu z/OS eingesetzt, und zwar entweder in einer z/VM virtuellen Maschine, oder in einer eigenen LPAR, oder beides. LPARs können parallel zu virtuellen Maschinen unter z/VM eingesetzt werden. Abb. 12.3.8 zeigt unterschiedliche Alternativen.

Die Möglichkeiten a) und c) sind denkbar, werden aber in der Praxis sehr selten eingesetzt. Am häufigsten findet man die Varianten b) und d) .

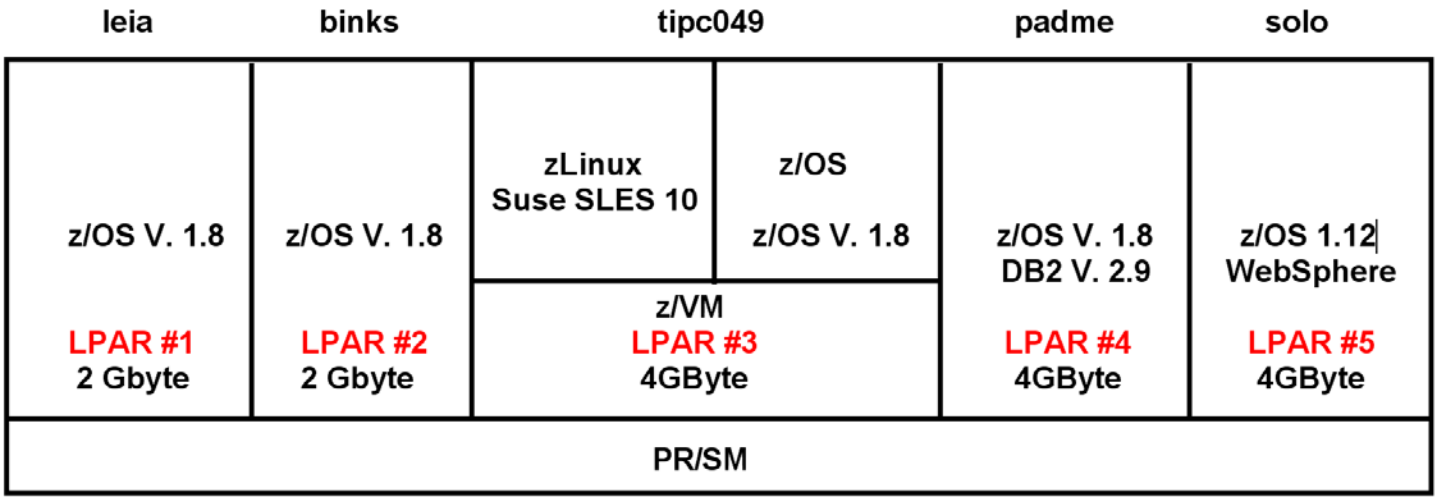


Abb. 12.3.9
Konfiguration des Rechners jedi.informatik.uni-leipzig.de

Ein Beispiel ist die Konfiguration des Mainframe Rechners des Lehrstuhls Technische Informatik an der Uni Leipzig. Wir betreiben 5 LPARs. Jede LPAR hat einen eigenen Ethernet Adapter für Anschluss an das Internet:

LPAR # 1 - 139.18.4.30	**leia.informatik.uni-leipzig.de**	**z/OS 1.8**
LPAR # 2 - 139.18.4.34	**binks.informatik.uni-leipzig.de**	**z/OS 1.5**
LPAR # 3 - 139.18.4.49	**tipc049.informatik.uni-leipzig.de**	**zVM mit zLinux und z/OS**
LPAR # 4 - 139.18.4.35	**padme.informatik.uni-leipzig.de**	**z/OS 1.8**
LPAR # 5 - 139.18.4.33	**Diplomarbeiten und Forschungsprojekte**	**z/OS 1.12**

Wir haben bisher keine Notwendigkeit gesehen, eine eigene LPAR für zLinux einzurichten. Auf dem Mainframe Rechner an der Uni Tübingen existiert eine eigene zLinux LPAR.

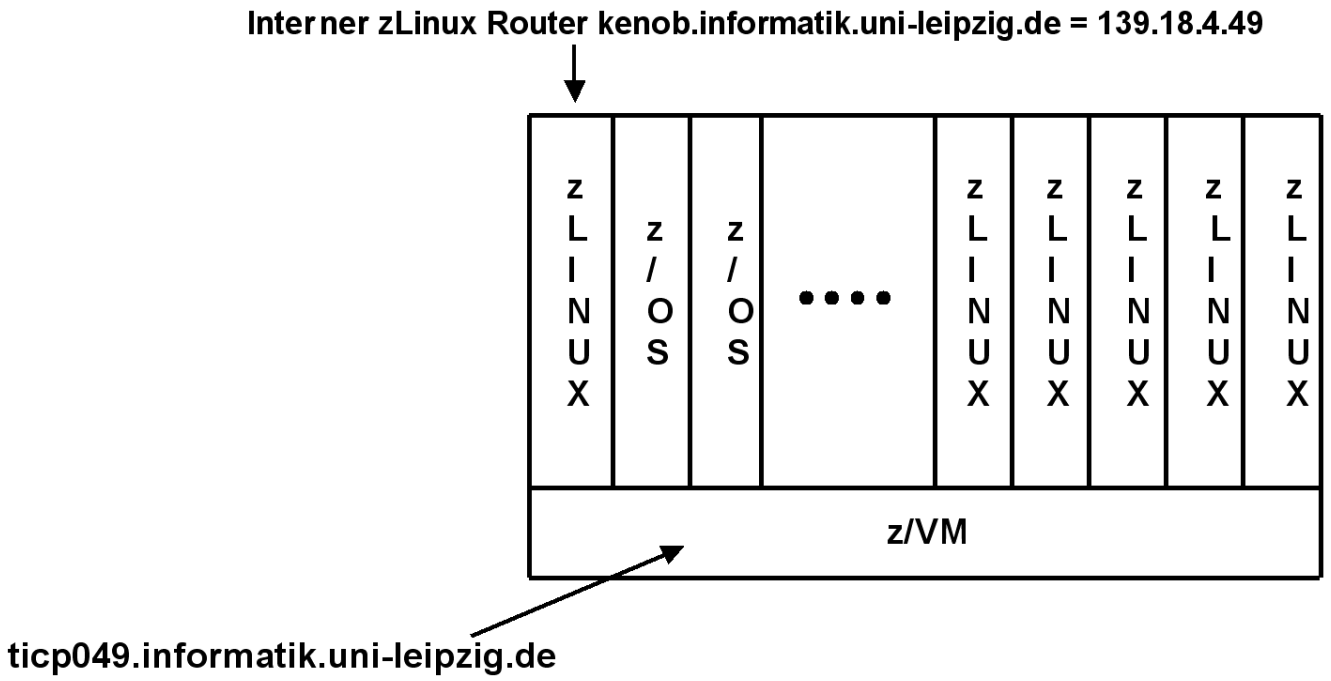


Abb. 12.3.10
kenob.informatik.uni-leipzig.de

Auf der LPAR # 3 - 139.18.4.49 laufen unter z/VM mehrere Gast Maschinen. Auf der ersten Gastmaschine läuft zLinux. Dieses dient als Router für alle weiteren Gastmaschinen.

Unter z/VM Gast sind etwa 20 virtuelle Maschinen installiert, teils z/OS, teils SLES v10 (Suse zLinux Enterprise Edition). Diese sind an ein internes Netz 10.x.x.x angeschlossen, welches der zLinux Router unterhält. Für einen Zugriff ruft man z/VM auf, und benutzt dort das „Dial VM“ Kommando, um sich mit einer der virtuellen Maschinen zu verbinden.

12.3.5 Zusammenfassung

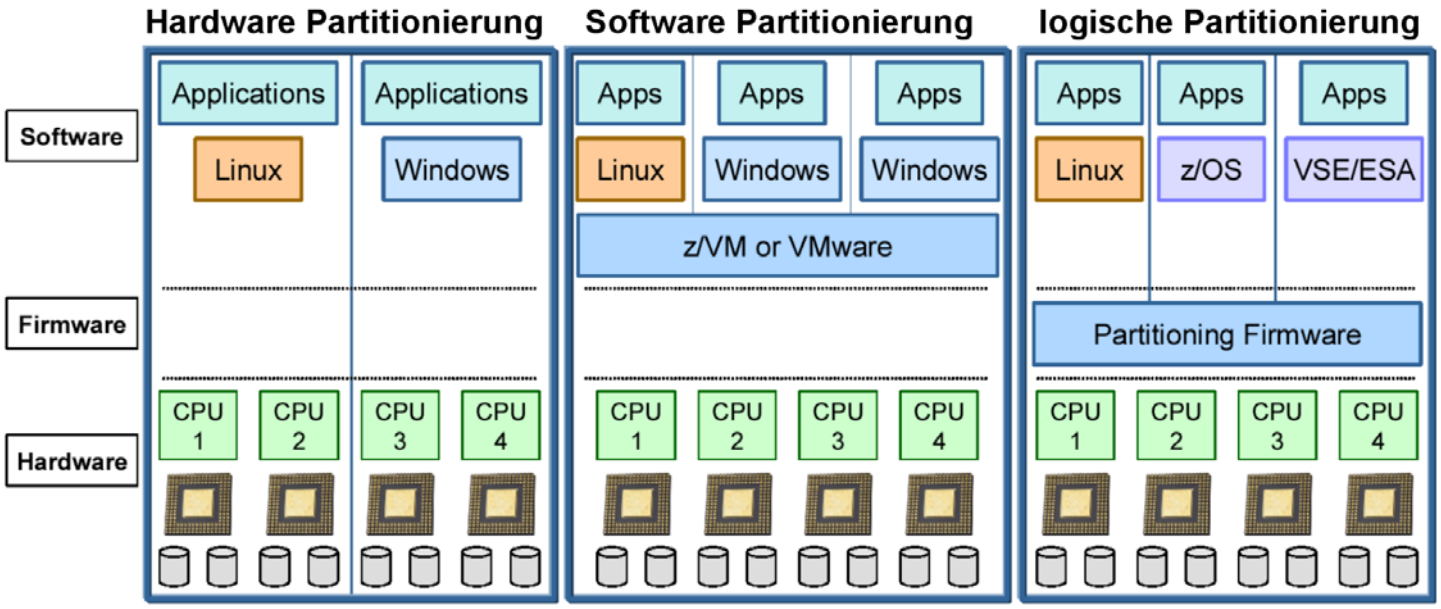


Abb. 12.3.11
Unterschiedliche Formen der Partitionierung

Hardware Partitioning ist unter anderem für die Großrechner der Firmen Sun und Hewlett Packard verfügbar.

Software Partitioning (auch als Virtual Partitioning bezeichnet) arbeitet mit einem Hypervisor (Host Kernel) und benötigt im Prinzip keine zusätzliche Hardware Unterstützung (Ausnahme Interpretive Execution Facility, Vanderpool und Pacifica).

Logical Partitioning benutzt Firmware für die Implementierung der Hypervisor Funktionalität. Weiterhin sind die virtuellen Maschinen nicht mehr in virtuellen Adressräumen des Host Kernels untergebracht.

Kommentar

Jurassic Park mit Zukunft

Wenn schon als Techno-Dinosaurier verspottet, dann kann man seine Mainframe-Produkte auch gleich nach den Weltherrschern von einst benennen – meint IBM und wählt mit Raptor und T-Rex besonders furchteinflößende Vertreter. Big Blue kann sich das kecke Spiel mit dem eigenen Image leisten: Immer deutlicher wird, dass die Architektur der Zeit nicht hinterher hinkt, sondern ihr sogar voraus eilt. Bei der logischen Partitionierung etwa geben Experten ihr einen zehnjährigen Entwicklungsvorsprung, und der Workload-Manager, der Ressourcen ziel- und situationsabhängig nutzt, fällt bei Standard-Servern viel primitiver aus. Eben solche Techniken sind aber die Voraussetzung für das On-Demand-Computing – die Nutzung von Rechenleistung ganz nach Bedarf. Neidische Blicke auf die Privilegierten, die sich einen T-Rex leisten können, sind aber unnötig: Stets ist der Open-Systems-Tross durch die technologischen Breschen, welche die Rechner-Dinos schlugen, nachgefolgt. Frank-Michael Kieß

12. Bei der logischen Partitionierung geben Experten ihr (System z) einen zehnjährigen Entwicklungsvorspr ung

Abb. 12.3.12
Computer Zeitung
19.5.2003, S. 1

12.4 Intelligent Resource Director

12.4.1 Größe des realen Speichers einer LPAR

Der Intelligent Resource Director (IRD) ist eine Erweiterung der LPAR Technologie. Die Erweiterungen bestehen im Wesentlichen aus 4 Einzelkomponenten.

- **Dynamische Verwaltung des realen Speichers**
- **LPAR CPU Management**
- **Dynamisches Channel Path Management**
- **Channel Subsystem Priority Queuing**

Das Betriebssystem einer LPAR nimmt an, dass sein realer Speicher über einen kontinuierlichen linearen Adressenraum, beginnend mit der Adresse 0x000000, verfügt.

Wird ein Mainframe Rechner neu installiert und konfiguriert, so wird entschieden, wie viele LPARs vorhanden sein sollen, und über wie viel realer Speicher jede LPAR verfügen soll. All diese Parameter werden in einer Konfigurationsdatei abgespeichert und bleiben in einfachen Fällen (z.B. auf unserem eigenen Mainframe Rechner an der Uni Leipzig) während der Lebensdauer unverändert.

Dies bedeutet, dass der LPAR x im physischen Speicher die Adressen von aaaa bis bbbb zugeteilt werden. Der reale Speicher der LPAR verfügt über den linearen Adressenraum von 0x000000 bis 0x00[bbbb-aaaa]. Dieser reale Adressenraum wird linear in den physischen Adressenraum von aaaa bis bbbb abgebildet. Das z/OS Betriebssystem (und auch das Windows Betriebssystem) unterstellt, dass sich bestimmte Teile des Kernels auf bestimmten realen Adressen befinden.

Will man den realen Speicher einer LPAR auf Kosten einer anderen LPAR vergrößern, ist es notwendig, alle LPARs herunterzufahren, die Konfigurationsdatei zu ändern, und die LPARs wieder hochzufahren. Es ist jedoch wünschenswert, dass dies während der z.B. 8 Jahre dauernden Lebensdauer des Rechners nie geschehen muss.

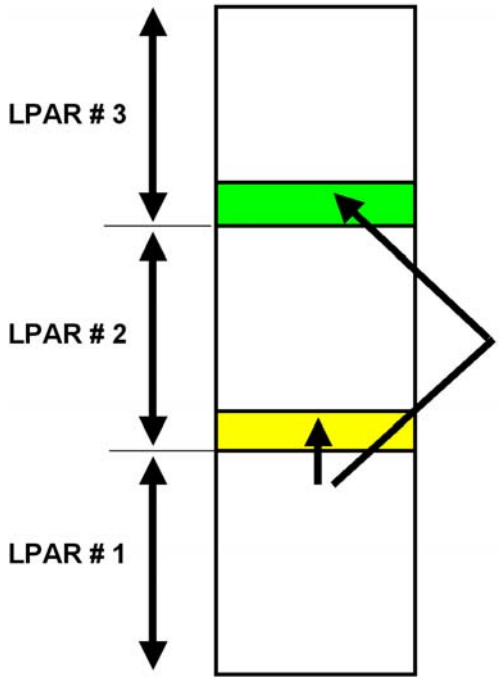


Abb. 12.4.1
Änderung der Größe des realen Speichers einer LPAR

Die mehrfachen realen Speicher der einzelnen LPARs werden in einem einzigen physischen Speicher abgebildet. Jeder LPAR wird ein zusammenhängender (contiguous) Adressenbereich in dem physischen Speicher zugeordnet.

Wenn sich im Laufe eines Tages die Auslastung der LPARs ändert, würde man gerne einer LPAR zusätzlichen physischen Speicher auf Kosten einer anderen LPAR zuordnen.

In dem gezeigten Beispiel wäre es möglich, dass LPAR # 1 auf Kosten der benachbarten LPAR # 2 Speicherplatz erhält. Es wäre nicht möglich, dass LPAR # 1 auf Kosten der nicht benachbarten LPAR # 3 Speicherplatz erhält, weil jede LPAR glaubt, einen eigenen physischen Speicher mit einen linearen Adressenraum mit kontinuierlichen Adressen zu besitzen.

z/OS Anwendungen verwenden virtuelle Adressen, die mit Hilfe von Segment- und Seitentabellen der normalen virtuellen Adressumsetzung in reale Adressen umgesetzt werden. Die Aufteilung des physischen Speichers auf die einzelnen LPARs erfolgt in 64 MByte großen „Logischen Memory Blöcken“ (LMB).

Bei der als IRD (Intelligent Resource Director) bezeichneten Erweiterung von PR/SM kann mit einem zusätzlichen Mechanismus (der ähnlich wie die virtuelle Speicherplatzverwaltung arbeitet) der Platz, der den einzelnen LPARs zugeteilt ist, dynamisch (während des laufenden Betriebes) in LMB Blöcken von je 64 MByte vergrößert und verkleinert werden.

Hierbei ist nicht mehr sichergestellt, dass sich die Menge der LMBs, die einer LPAR zugeteilt sind, in einem zusammenhängenden linearen Teil des physischen Speichers befindet. Damit für die LPARs die Illusion eines kontinuierlichen linearen Adressenraums gewährt bleibt, wird ein Ansatz ähnlich der virtuellen Speicherplatzverwaltung benutzt. Dabei wird wie bei den Seitentabellen der kontinuierliche (contiguous) reale Adressenraum einer LPAR in eine diskontinuierliche Menge von 64 MByte Blöcken innerhalb des physischen Speichers mit Hilfe einer Block Tabelle umgesetzt.

Die Größe des den einzelnen LPARs zugeordneten physischen Speicherbereiches kann mit Hilfe einer weiteren Adressenumsetzung dynamisch variiert werden:

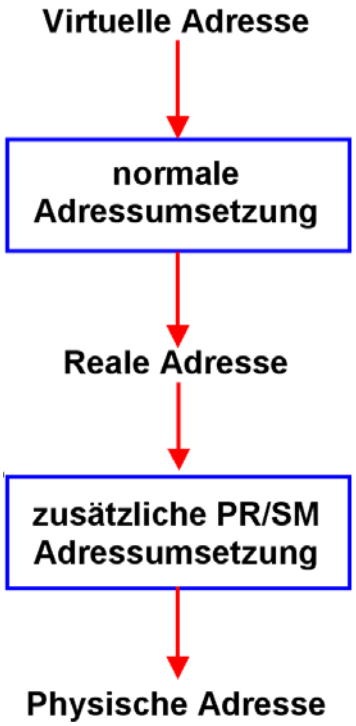


Abb. 12.4.2
Die PR/SM Adressumsetzung erfolgt zusätzlich zu der normalen Adressumsetzung

Die Seiten und Rahmen der zusätzlichen PR/SM Adressumsetzung haben eine Größe von 64 MByte, gegenüber 4 KByte bei der normalen virtuellen Adressumsetzung.

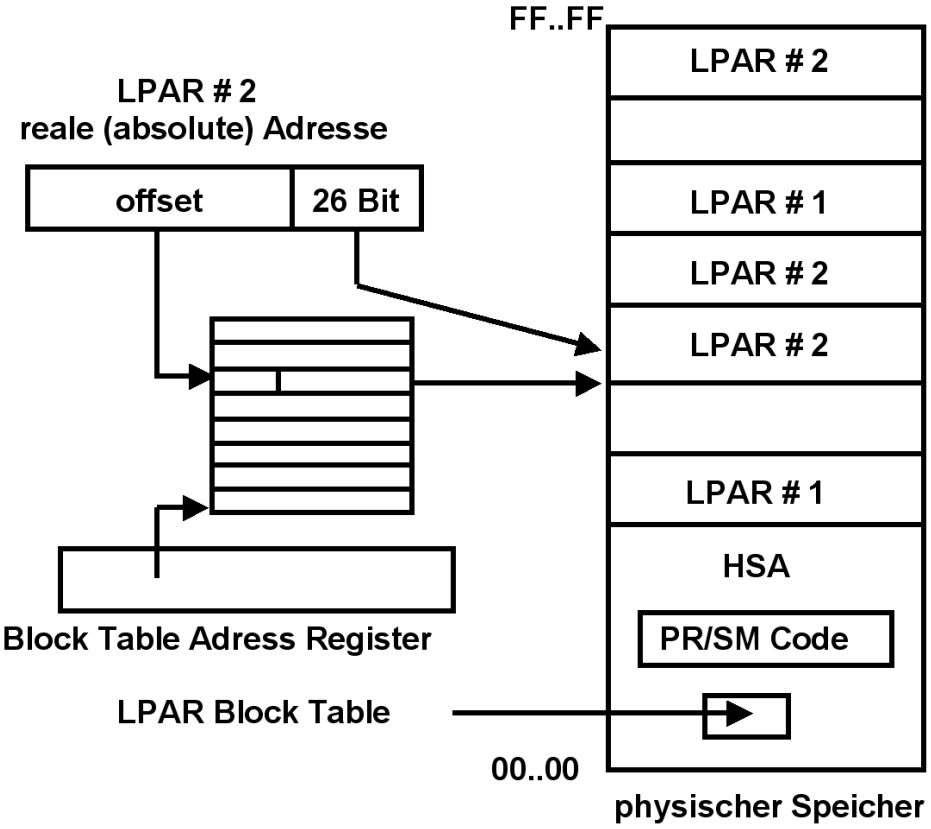


Abb. 12.4.3
PR/SM Adressumsetzung mit Hilfe eines Block Tables

Der physische Speicher wird in 64 MByte große LMBs aufgeteilt.

An Stelle des Zone Origin Registers besteht ein Block Table Adressregister. Dieses enthält die Anfangsadresse einer Block Table. Für jede LPAR ist eine derartige Block Table in der HSA enthalten. Die unteren 26 Bit der von einer LPAR erzeugten realen Adresse zeigen auf ein Datenfeld innerhalb eines 64 MByte Blockes. Die oberen Bit der von einer LPAR erzeugten realen Adresse zeigen auf einen Eintrag der LPAR Block Table. Diese enthält die Anfangsadresse eines 64 MByte Blockes im physischen Speicher.

Die Funktion ist vergleichbar mit einer einstufigen virtuellen Address Translation. Jede LPAR hat einen linearen Adressenraum für ihre realen Adressen.

Für jeden 64 MByte Block im physischen Speicher ist ein Eintrag in der LPAR Block Table vorhanden. Deshalb ist ein Fehlseiten-vergleichbarer Mechanismus nicht erforderlich.

12.4.2 LPAR CPU Management

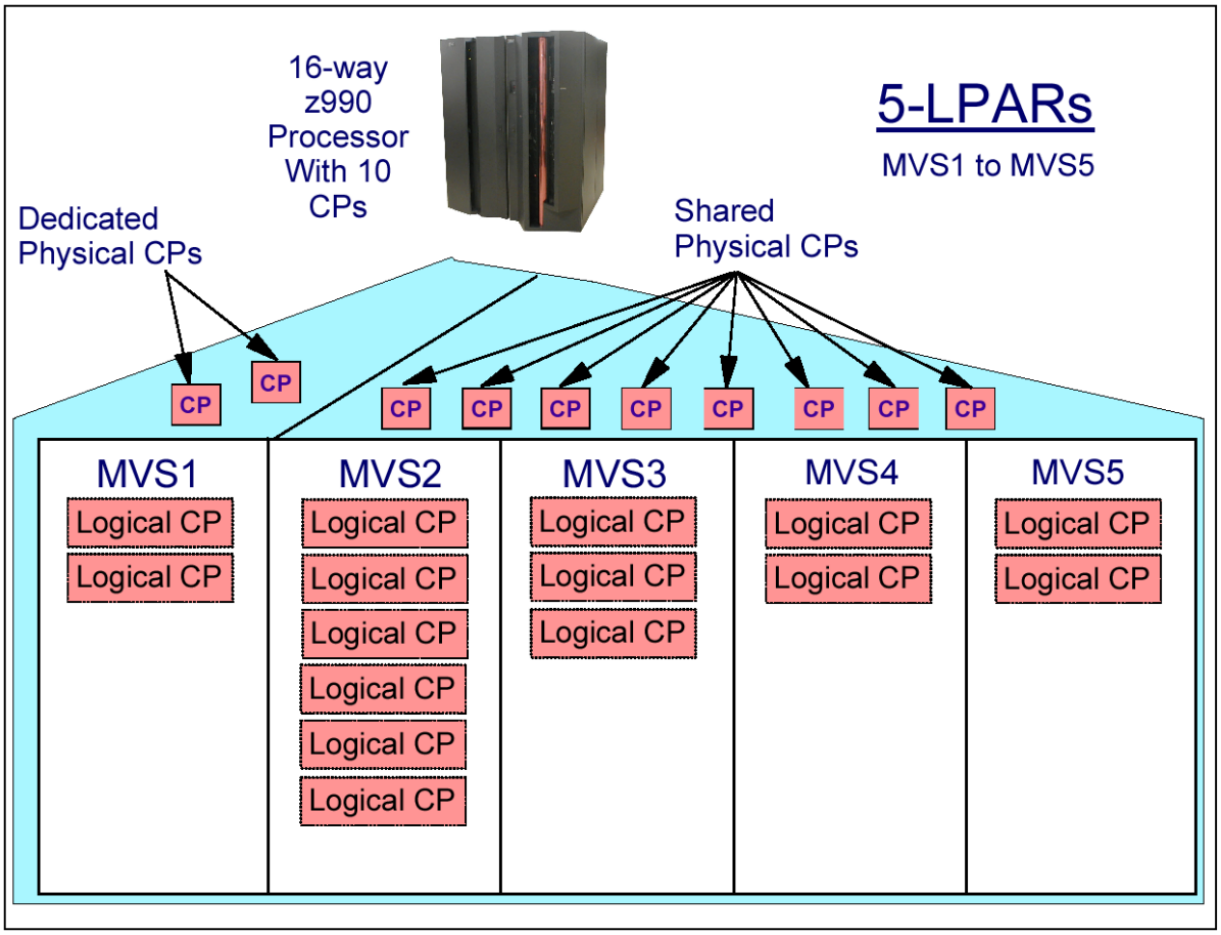


Abb. 12.4.4
Von mehreren LPARs gemeinsam genutzte CPUs

Physische CPUs (CPs) können einer bestimmten LPAR fest zugeordnet, oder von mehreren LPARs gemeinsam genutzt (shared) werden. Bei einer LPAR mit fest zugeordneten CPUs ist eine logische CPU permanent einer physischen CPU zugeordnet. Dies bedeutet weniger Overhead.

Gemeinsam genutzte (shared) physische CPUs erzeugen mehr Overhead. Dies wird überkompensiert, weil eine LPAR CPU Kapazität nutzen kann, die eine andere LPAR gerade nicht benötigt. Wenn ein Betriebssystem in den Warte- (wait) Zustand versetzt wird, gibt es die benutzten CPUs frei.

In dem gezeigten Beispiel sind der LPAR MVS1 zwei CPUs fest zugeordnet. Die vier LPARs MVS2, MVS3, MVS4 und MVS5 teilen sich in die Nutzung von acht physischen CPUs.

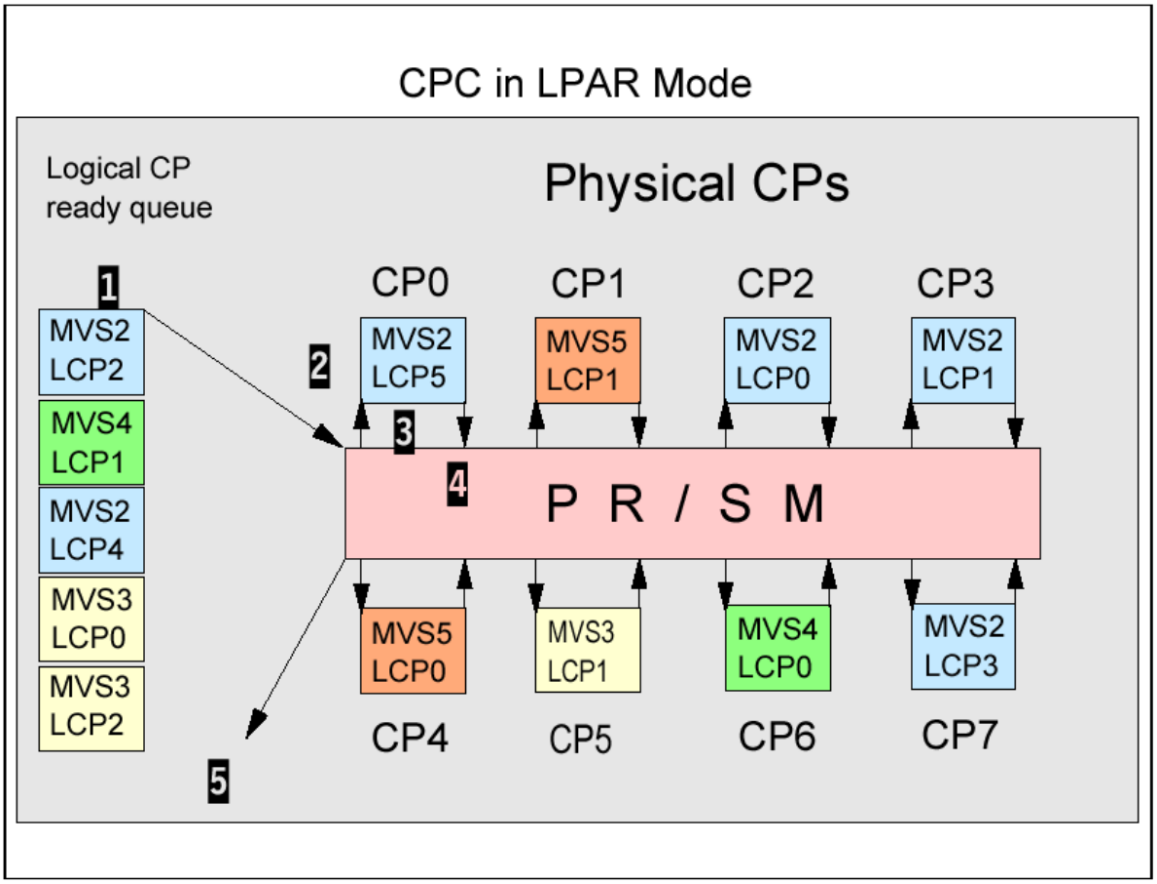


Abb. 12.4.5
LPAR dispatching

Den vier LPARs mit den Betriebssystemen MVS2, MVS3, MVS4 und MVS5 stehen 8 physische CPs (CPUs) zur Verfügung (CP0 … CP7).

Jedes der Betriebssysteme ist als Multiprozessor-Betriebssystem konfiguriert und glaubt, über eine bestimmte Anzahl (logischer) CPUs zu verfügen. Die Anzahl der logischen CPUs übertrifft die Anzahl der physisch vorhandenen CPUs.

Der PR/SM Hypervisor ordnet die logischen CPs den physisch vorhandenen CPs zu. Er unterhält eine „Ready Queue“ der ausführbaren logischen CPUs aller LPARs. Wenn eine physische CPU frei wird, ordnet ihr PR/SM eine logische CPU aus der Ready-Queue zu.

Der LPAR Dispatching Code ist Teil des PR/SM Hypervisor.

Angenommen, eine physische CPU wird frei, z.B. CP0 in Abb. 12.4.5. Die Zuordnung einer logischen CPU zur physischen CP0 erfolgt in den folgenden Schritten:

1. **Die als nächstens auszuführende logische CPU wird von der logischen CP Ready-Queue ausgewählt, entsprechend dem Gewicht (Weight) der logischen CPU.**
2. **LPAR Firmware (LIC) ordnet die selektierte Logische CPU (LCP5 von MVS2) auf eine physische CPU (CP0 in Abb. 12.4.5) zu.**
3. **Die z/OS Dispatchable Unit, die auf der logischen CPU (MVS2 logical LCP5) läuft, startet die Ausführung auf der physischen CP0. Sie läuft, bis ein Zeitintervall (typisch zwischen 12.5 und 25 ms) abgelaufen ist, oder ein Wait-Ereignis eintritt.**
4. **Bei Zeitintervallende wird die Laufzeitumgebung der logischen CP5 abgespeichert. Die Kontrolle geht zurück an die LPAR Firmware, die auf der physischen CP0 wieder startet.**
5. **LPAR Firmware ermittelt, warum die logische CPU die Ausführung beendete, und reiht diese z.B. in eine wartende Queue ein. Wenn LPC5 wieder ausführbar ist, wird sie wieder in die logische CP Ready-Queue eingereiht. Darauf beginnt Schritt 1 von neuem.**

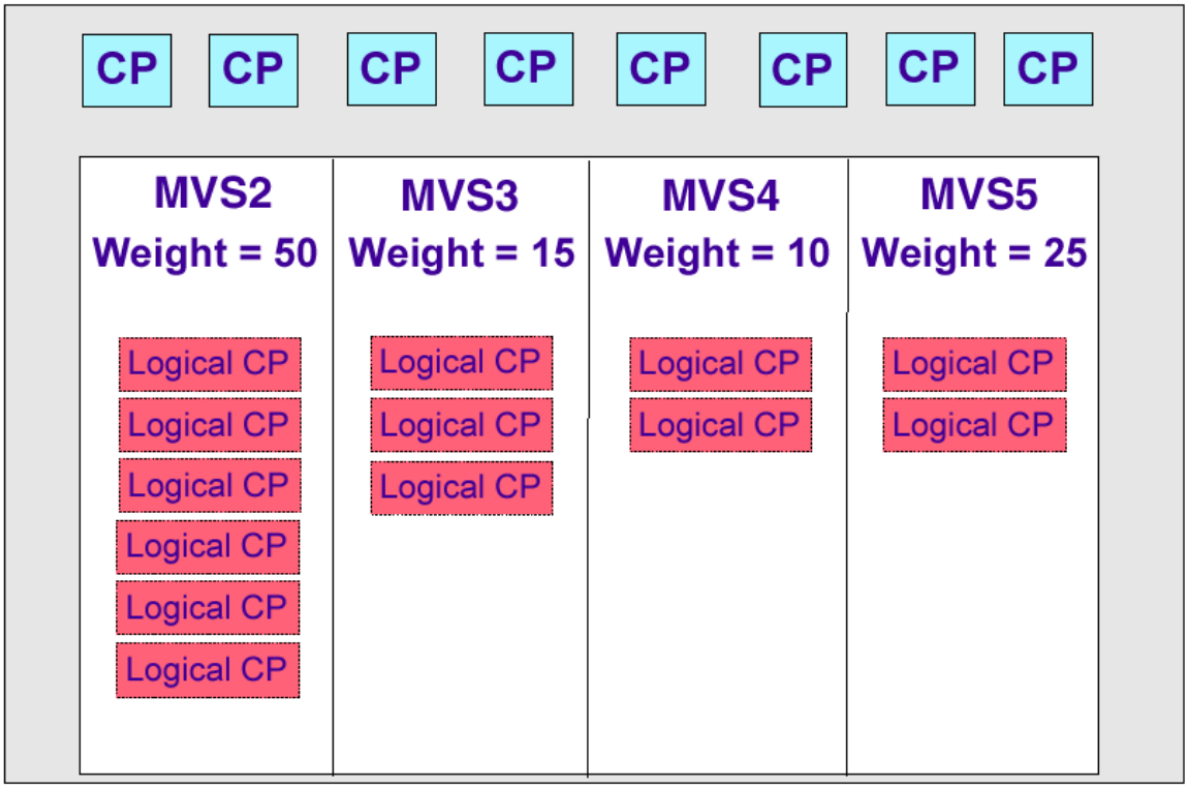


Abb. 12.4.6
LPAR Gewichte

Mit Hilfe von LPAR Gewichten (weights) wird die Zuordnung von logischen CPUs auf physische CPUs gesteuert.

LPAR Gewichte bestimmen das garantierte Minimum an physischen CPU Ressourcen, welche eine logische CPU erhält, falls diese sie anfordert. Dieser garantierte Betrag ist gleichzeitig das Maximum, wenn alle logischen CPUs das garantierte Minimum voll ausschöpfen.

Eine LPAR verbraucht möglicherweise weniger als das garantierte Minimum, wenn nicht ausreichend Arbeit anfällt. In diesem Fall steht den anderen LPARs mehr CPU Leistung zur Verfügung.

12.4.3 Dynamic Channel Path Management

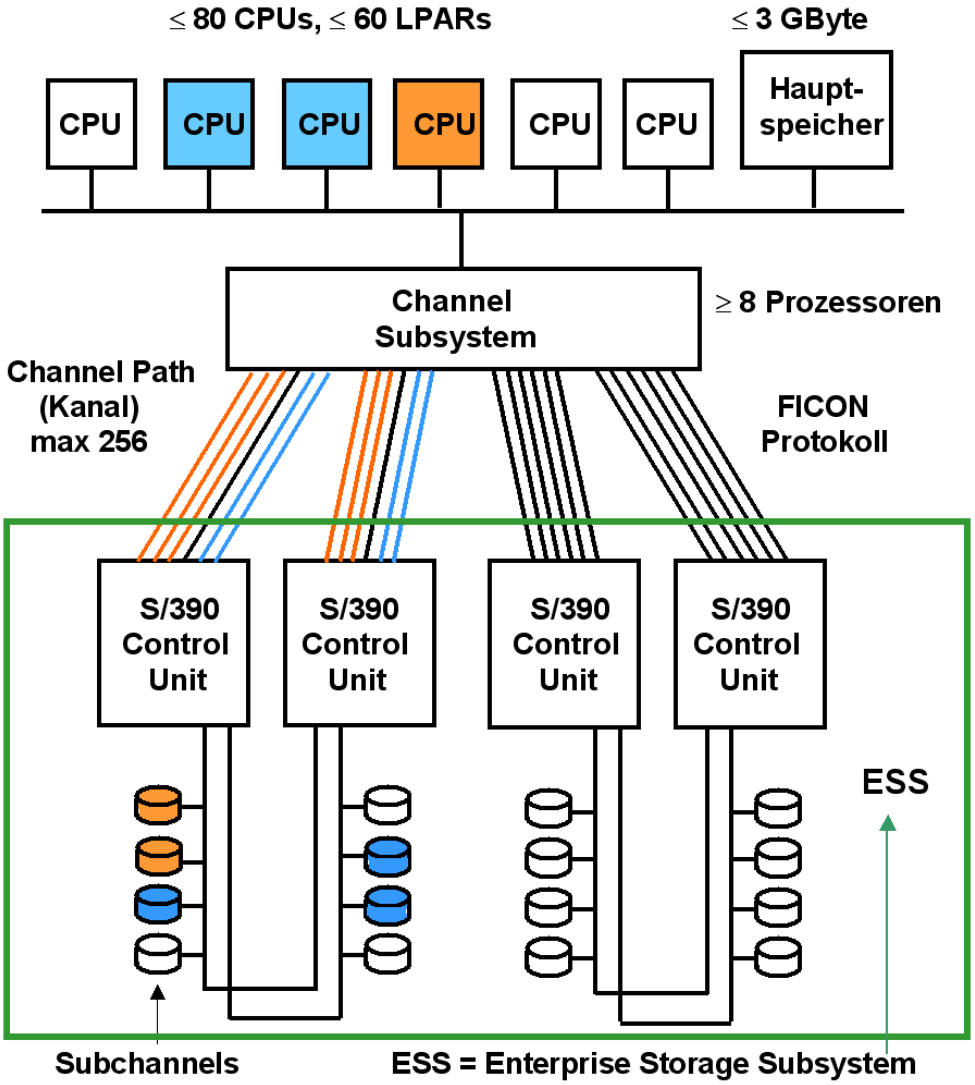


Abb. 12.4.7
Logische Zuordnung von Festplattenspeichern zu bestimmten LPARs

Festplattenspeicher werden über ein Enterprise Storage Subsystem (ESS) angeschlossen, welches S/390 Control Units abbildet.

Die blaue LPAR (mit 2 CPUs) verfügt derzeitig über 4 Kanäle. Die rote LPAR (mit 1 CPU) verfügt derzeitig über 6 Kanäle.

Dynamic Channel Path Management (DCM) ist in der Lage, die Kanal Konfiguration dynamisch an sich ständig ändernde Auslastungen anzupassen.

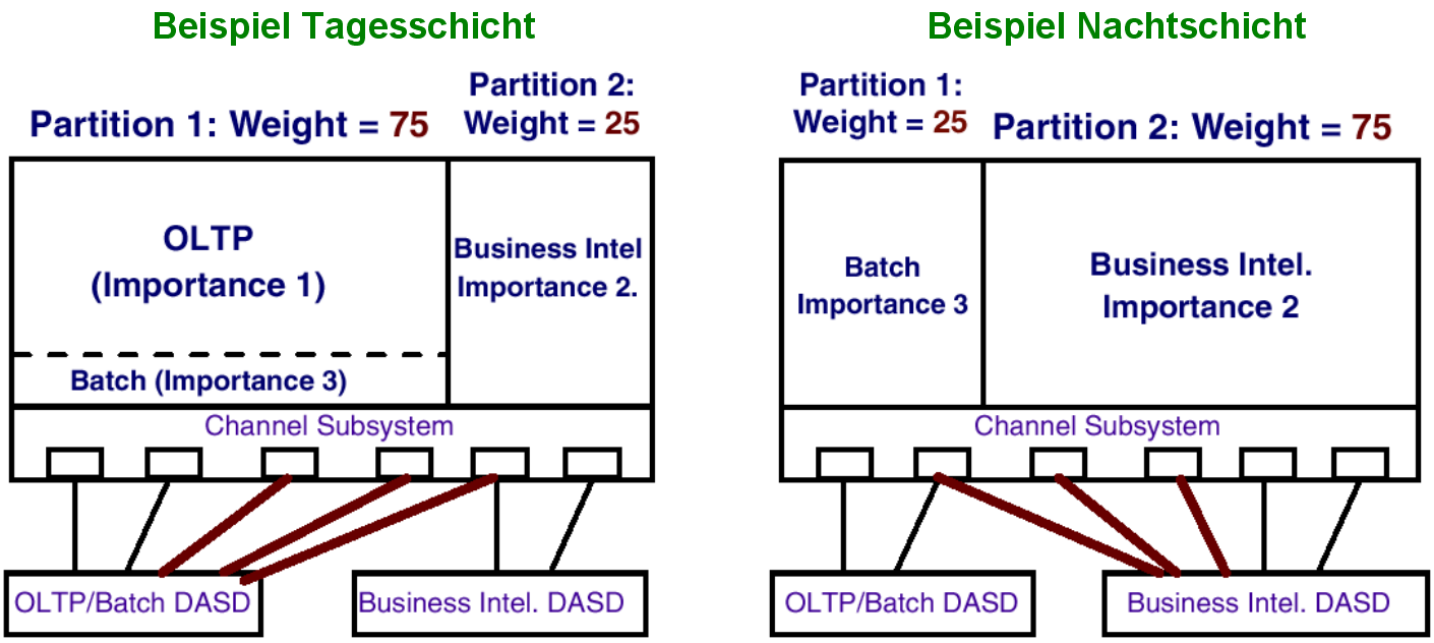


Abb. 12.4.8
Änderung der Kanalzuordnung im Laufe eines Tages

In dem in Abb. 12.4.8 gezeigten Beispiel sind während der Tagesschicht die Mehrzahl der Kanäle den OLTP (Online Transaction Processing) Anwendungen zugeordnet. Während der Nachtschicht sind die Mehrzahl der Kanäle einer LPAR für Batch Processing Anwendungen (z.B. Business Intelligence) zugeordnet.

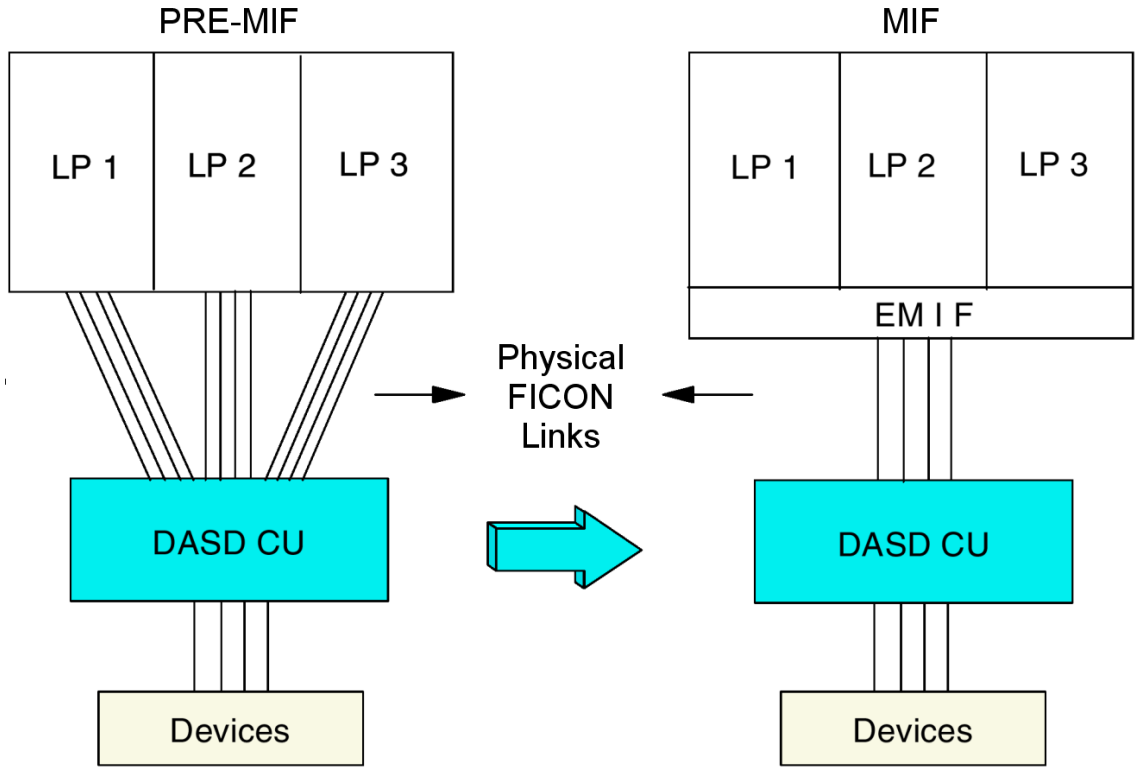


Abb. 12.4.9
Multiple Image Facility

Im einfachsten Fall sind die Kanäle einzelnen LPARs fest zugeordnet.

Mit Hilfe der Multple Image Facility (MIF) können Kanäle von mehreren LPARs gemeinsam genutzt werden. Damit kann MIF die Kanäle bestimmten LPARs unter Prioritätsgesichtpunkten dynamisch zuordnen.

An Stelle der Bezeichnung MIF wurde früher auch die Bezeichnung EMIF (extended Multple Image Facility) verwendet.

12.4.4 Channel Subsystem I/O Priority Queuing

Normalerweise werden I/O Anforderungen vom Channel-Subsystem auf einer First-In-First-Out-Basis abgearbeitet.

Wenn eine Arbeitseinheit mit hoher Priorität ihre Bearbeitungsziele wegen Überlastung der I/O Einrichtungen nicht erreicht, kann der z/OS Work Load Manager (siehe Kapitel 13) dieser Arbeitseinheit eine höhere Priorität als anderen Arbeitseinheiten zuordnen. Die „Channel Subsystem Priority Queuing" Einrichtung des Channel Subsystems wird dann diese Arbeitseinheit bevorzugt abfertigen, indem sie hierfür zusätzliche Bandbreite auf Kosten anderer Arbeitseinheiten mit weniger Priorität verfügbar macht.

Hierfür werden Dynamic Channel Path Management und MIF eingesetzt.

12.4.5 PR/SM und LPAR Sicherheit und Isolation

Zertifizierung: LPARs haben äquivalente Sicherheits-Eigenschaften wie physisch getrennte Rechner.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) stellt IBM für den Processor Resource/System Manager (PR/SM) des Mainframes das weltweit höchste Sicherheitszertifikat für einen Server aus. Die Bescheinigung nach dem internationalen Standard Common Criteria (CC) für die Stufen EAL4 und EAL5 wurde auf der CeBIT 2003 an IBM verliehen. Mainframes mit PR/SM waren damals die ersten Server, die nach der Evaluierungsstufe EAL5 für seine Virtualisierungstechnologie zertifiziert wurde.

Die Zertifizierung des BSI bescheinigt, dass Programme, die auf einem Mainframe Server in verschiedenen logischen Partitionen (LPAR) laufen, ebenso gut voneinander isoliert sind, als würden sie auf getrennten physikalischen Servern laufen.

Die Logische Partitionierung weist einzelnen Applikationen und Workloads unterschiedliche Bereiche auf dem Server zu und kann diese komplett voneinander abschirmen. So können beispielsweise Web-Anwendungen und Produktionsanwendungen, die in getrennten logischen Partitionen laufen, komplett voneinander isoliert betrieben werden, obwohl sie die physikalischen Ressourcen des zSeries-Servers gemeinsam nutzen.

IBM Presseinformation SG/94/2003, CeBIT 2003: 14. März 2003 -

12.4.6 Internal Queued Direct I/O

Trotzdem ist es manchmal erwünscht, dass LPARs Nachrichten miteinander austauschen. Hierzu ordnen wir jeder LPAR einen Ethernet Adapter zu.

Ethernet Ports sind in der Form einer I/O Adapter Karte verfügbar, siehe Band 1, Abb. 4.3.9. Die „OSA“ Adapter Card unterstützt vier Gbit Ethernet Ports oder zwei 10 Gbit Ethernets Ports. Die Ports können unterschiedlichen LPARs zugeordnet werden. Die OSA Adapter Card enthält außerdem die Funktion einer Control Unit. Ein als „Queued Direct I/O“ (QDIO) bezeichneter I/O Driver stellt die Verbindung zu z/OS her. PR/SM benutzt das OSI Schicht 5 Internet „Socket“ Protokoll für die Ethernet Verbindung (siehe Abschnitt 18.1.2).

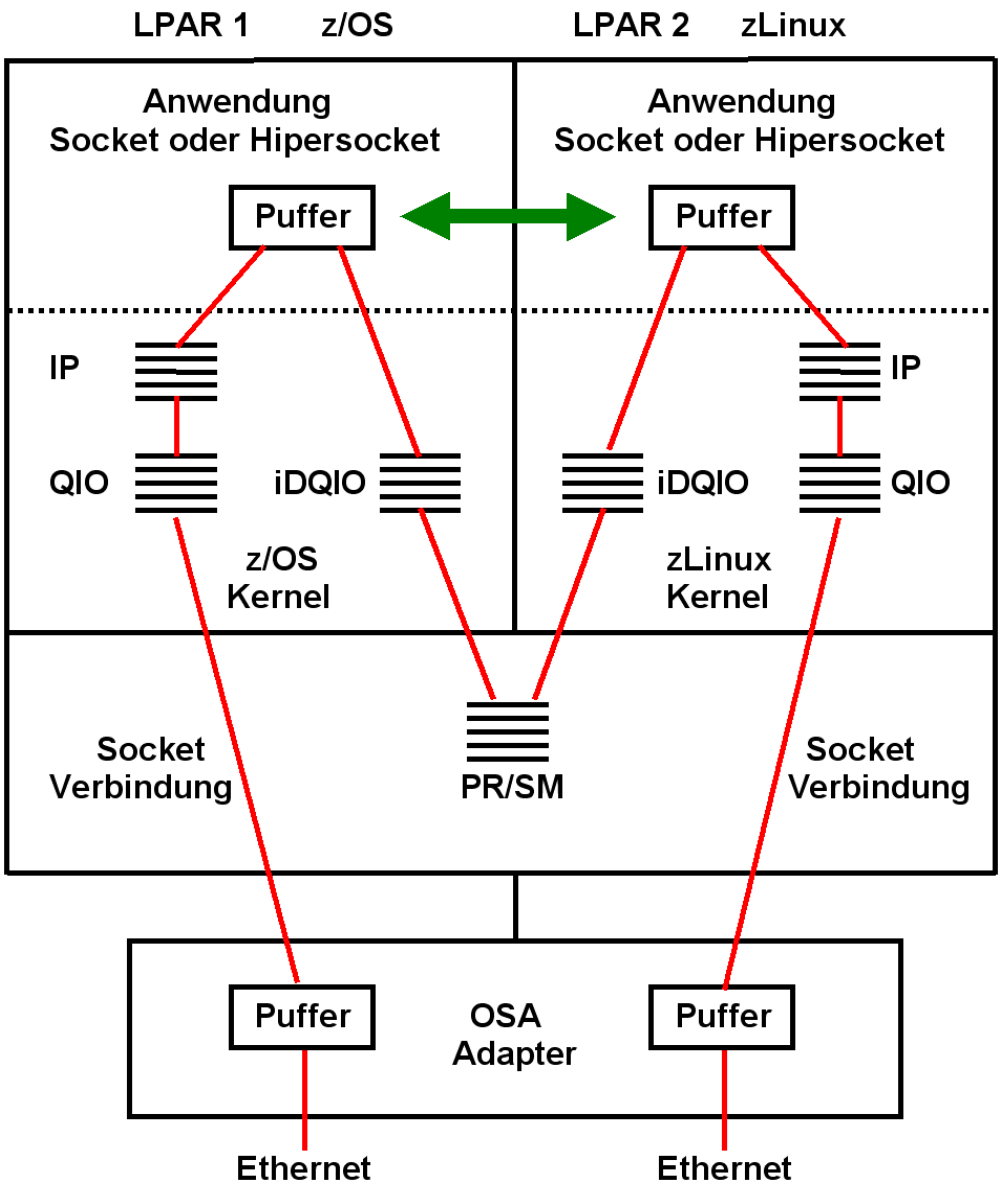


Abb. 12.4.10
HiperSockets

In Abb. 12.4.10 sind zwei LPARs dargestellt, die eine Nachricht austauschen möchten.

Da sich die LPARs wie getrennte physische Rechner verhalten, kann man jeder LPAR einen OSA Adapter Port zuordnen, und die beiden Ports über Sockets und einem Ethernet Kabel miteinander verbinden.

Für zwei LPARS auf dem gleichen z/OS Rechner ist das unnötiger Overhead. Statt dessen können die beiden LPARs „internal Queued Direct I/O“ (iDQIO) verwenden. IDQIO verwendet ebenfalls Sockets. Die Verbindung wird jedoch von PR/SM über eine Queue in der HSA hergestellt. Dies eliminiert den OSI Schicht 2 – 4 Overhead.

12.4.7 HiperSockets

HiperSockets ermöglichen es mehreren LPARs innerhalb des gleichen Rechners (oder Sysplex) miteinander zu kommunizieren, ohne auf ein externes, physisches Netzwerk zurückgreifen zu müssen (LPAR-übergreifende Kommunikation). Die Hochgeschwindigkeits-kommunikation über HiperSockets kann auch für die Kommunikation zwischen z/OS, z/VM und „Linux on System z“ in unterschiedlichen LPAR Instanzen eingesetzt werden. Mit dieser Funktion lässt sich innerhalb des Systems ohne eine zusätzliche physische Verbindung ein "systeminternes Netz" aufbauen.

Da die Kommunikation den Hauptspeicher benutzt, erfolgt sie mit Hauptspeicher-Zugriffsgeschwindigkeit. Es bestehen zusätzlich Sicherheitsvorteile, weil die Möglichkeit einer Netzwerk Interception (z.B. Man-of-the-Middle Attacke) von außen entfällt. HiperSockets verbessern Reliability und Availability, weil Network Hubs, Routers, Adapters oder Kabelverbindungen alle im Hauptspeicher virtuell dargestellt werden.

Die HiperSockets Implementierung basiert auf der OSA-Express Queued Direct I/O (QDIO)-Protokoll. Daher werden HiperSockets auch als internes QDIO oder IQDIO bezeichnet. Der Microcode des OSA Express Adapters emuliert die Link Control Schicht (Schicht 2 des OSI Stacks) einer OSA-Express QDIO Schnittstelle.

VSWITCH ist eine Firmware Funktion, die unter Nutzung von iQDIO wie ein OSI Layer 2 Switch arbeitet. Es verbindet ein externes Netzwerk über den OSA Adapter mit den virtuellen Gast Maschinen.

HiperSockets ermöglichen in einem Rechner bis zu 16 “virtual” Local Area Networks (LANs) mit deutlich reduziertem System Overhead.

HiperSockets können für die Kommunikation in einem einzigen physischen System oder zwischen Sysplex-Instanzen eingesetzt werden. Sie unterstützen ebenfalls die zBX (siehe Abschnitt 14.3).

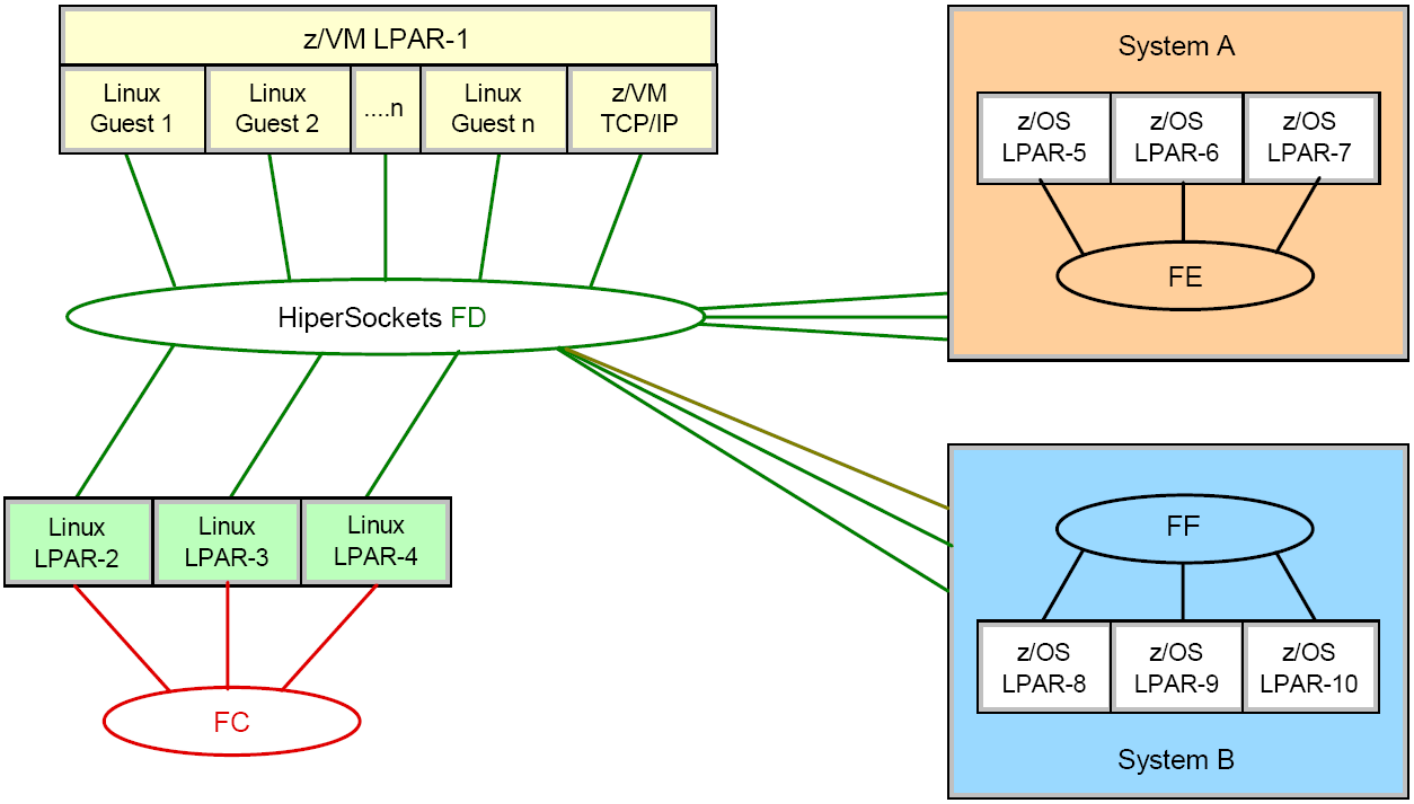


Abb. 12.4.11

HiperSocket Verbindung mit vier Virtuellen LANs

Abb. 12.4.11 zeigt eine Sysplex Konfiguration mit drei Systemen:

- **System A mit LPAR-5 , LPAR-6 und LPAR-7,**
- **System B mit LPAR-8 , LPAR-9 und LPAR-10,**
- **System C mit LPAR-1 , LPAR-2, LPAR-3 und LPAR-4.**

Die Verbindung erfolgt mit vier virtuellen LANs mit den Bezeichnungen FC, FD, FE und FF.

LAN FC verbindet die zLinux Systeme in LPAR-2, LPAR-3 und LPAR- 4.

LAN FE verbindet die z/OS Systeme in LPAR-5, LPAR-6 und LPAR- 7.

LAN FF verbindet die z/OS Systeme in LPAR-8, LPAR-9 und LPAR- 10.

LAN FD verbindet

- **die drei zLinux Systeme in LPAR-2, LPAR-3 und LPAR- 4,**
- **alle Linux Server, die unter z/VM in LPAR-1 laufen**
- **Alle z/OS Server in System A (LPARs 5 bis 7)**
- **Alle z/OS Server in System B (LPARs 8 bis 10)**
- **Den z/VM TCP/IP Stack, der in in LPAR-1 läuft**

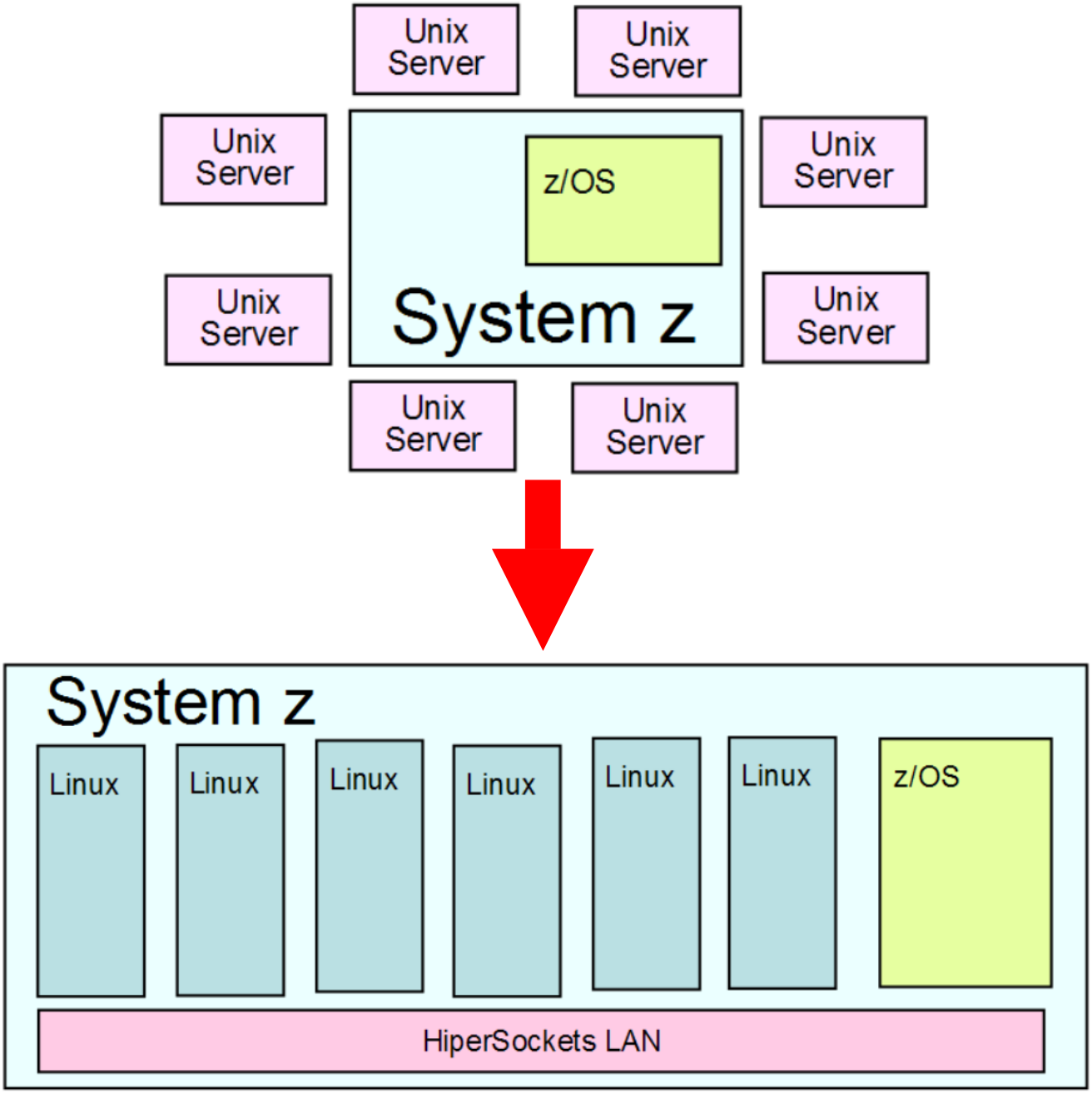


Abb. 12.4.12
Rezentralisierung mit „Linux on System z“

In vielen Unternehmen und staatlichen Organisationen existieren neben dem Mainframe Hunderte oder Tausende von dezentralisierten (distributed) Servern. Diese Server sind vielfach über ein historisch gewachsenes Netzwerk von Ethernet Verbindungen, Switches und Routern miteinander verbunden. Um die Administrationskosten zu senken ist es wünschenswert, diese Agglomeration von Servern zu zentralisieren.

Angenommen, auf allen dezentralisierten Servern läuft Linux als Betriebssystem. Es ist möglich, die dezentralen Server abzubauen und stattdessen eine Reihe von „Linux on System z“-Instanzen (zLinux) auf einem größeren Mainframe Server einzurichten.

Die CPU Auslastung auf dezentralen Servern liegt häufig bei unter 20 %. Da zLinux Prozessoren mit einer höheren Auslastung (bis zu 100 %) betrieben werden können, kann die Anzahl der zLinux Instanzen deutlich kleiner als die Anzahl der bisherigen Linux Server sein.

Die Firma IBM macht Reklame mit dem relativ niedrigen Energie Verbrauch und der Umweltfreundlichkeit ihrer Mainframe Server. In Bezug auf Linux Server Konsolidierung ist dies berechtigt.

Mit der in Abb. 12.4.12 gezeigten Rezentralisierung zahlreicher Linux Server auf dem Mainframe kann die komplexe Ethernet Infrastruktur vereinfacht werden. Da physische Ethernetkabel durch virtuelle HiperSocket-Verbindungen ersetzt werden, ist ein Abhören der Verbindungen (Man-of-the-Middle-Attacke) nicht mehr möglich. Komplexe Verschlüsselungsverfahren können entfallen. Separate Firewall Rechner und mehrfache Demilitarized Zones (DMZ) sind möglicherweise ebenfalls nicht mehr erforderlich. Die physische Ethernet Struktur wird durch virtuelle LANs ersetzt und dabei vermutlich wesentlich vereinfacht.

12.5 Weiterführende Information

Vielleicht interessiert Sie hier ein Video vom IBM Vertrieb

Virtualization with IBM System z
http://www.youtube.com/watch?v=4xL8s8WZUxo&NR=1

LPARS werden neben System z auch von System p implementiert:
http://www.ibmsystemsmag.com/aix/administrator/systemsmanagement/Configuring-Processor-Resources-for-System-p5-Shar/
und
http://www.ibmsystemsmag.com/aix/administrator/lpar/An-LPAR-Review/

Einen Aufsatz "A Comparison of Software and Hardware Techniques for x86 Virtualization" können Sie herunterladen unter

http://www.informatik.uni-leipzig.de/cs/Literature/esiisup/part01.pdf

Das folgende Video zeigt, wie man z/OS unter Hercules installiert

http://www.youtube.com/watch?v=GRW4iPhCDSM

Falls Sie dies interessiert, enthalten die folgenden Seiten zusätzliche Informationen zu den Virtualisierungsproblemen bei der x86 Architektur.

12.5.1 Backup Information zu VMware

VMware ist ein Produkt mit Virtual Machine Hypervisor Funktionalität für die x86 Architektur und ist in zwei Versionen verfügbar: *VMware Workstation* und *ESX Server*.

VMware Workstation ist die ursprüngliche Version. Es handelt sich um ein als Host-OS bezeichnetes Anwendungsprogramm, das entweder unter Windows oder unter Linux auf einer x86 Plattform im Benutzer-Modus läuft, und als Kombination die Funktionalität eines Host-Kernels bereitstellt. Eine zusätzliche, als VMDriver bezeichnete Komponente wird in Windows oder Linux integriert. Es bestehen Ähnlichkeiten mit der Art, mit der eine virtuelle Maschine in einem DOS-Fenster unter Windows abläuft. VMware Workstation nutzt die existierenden Ein-/Ausgabe Einrichtungen des Host-Betriebssystems. Wenn ein Gast-Betriebssystem eine Ein-/Ausgabe Operation ausführt, wird diese von VMDriver abgefangen und unter Benutzung geeigneter Systemaufrufe interpretiert. Ebenso werden Ein-/Ausgabe Unterbrechungen unter Beteiligung des VMDrivers bearbeitet.

Das Host-OS kann Seiten auslagern, die einer spezifischen virtuellen Maschine zugeordnet sind. Lediglich ein kleiner Satz an Seiten wird ständig im Hauptspeicher gehalten. Dies bedeutet, dass VMware Workstation vom Host-Betriebssystem wie ein normaler Benutzer-Prozess behandelt wird. Falls der Algorithmus des Hosts zur Seitenersetzung nicht optimal arbeitet, kann dies zu einer VMware Leistungsverschlechterung führen.

Die Nutzung von Host-Betriebssystem Komponenten verursacht einen Leistungsverlust, insbesondere bei Ein-/Ausgabe-Operationen. VMware stellt deshalb mit seinem ESX Server einen eigenen Host-Kernel zur Verfügung. Dieser benötigt keine Unterstützung durch das Host-Betriebssystem, läuft auf der realen Hardware und hat vergleichbare Funktionen wie der VM/370 Host-Kernel. Es werden bis zu 64 virtuelle Maschinen unterstützt. Gast-Betriebssysteme, die unter dem ESX Host–Kernel laufen, verfügen über einen mit der Adresse 0 beginnenden linearen virtuellen Adressenraum. *Die* Adressumsetzung erfolgt ähnlich wie bei VM/370 mit Hilfe von shadow page tables und einer als pmap bezeichneten Datenstruktur.

Gast-Maschinenbefehle, die Gast-Seitentabellen oder den TLB abändern, werden vom ESX Host–Kernel abgefangen und interpretiert. Der TLB enthält hierzu immer die in den shadow page tables enthaltenen Adressumsetzungen. Die wichtigsten Ein-/Ausgabe-Treiber sind in bezug auf maximale Leistung im Gastbetrieb-Modus optimiert. Die derzeitige Version ist in der Lage, einen symmetrischen Multiprozessor (SMP) mit bis zu 2 CPUs zu unterstützen.

Im Vergleich zu VM/370 sind der ESX Server und VMware benachteiligt, weil einige kritische Eigenschaften in der x86-Architektur fehlen. Für den Betrieb von Gast-Maschinen ist es erforderlich, dass alle Maschinenbefehle, welche den privilegierten Maschinenstatus abändern oder auch nur lesen, nur im Kernel-Modus ausgeführt werden können.

Dies sei an einem Beispiel erläutert. Wenn ein Gast ein Kontroll-Register schreibt, muss der Host-Kernel diesen Maschinenbefehl abfangen, damit nicht das reale Kontroll-Register des Hosts verändert wird. Der Host-Kernel wird jetzt nur die Effekte der Instruktion für diesen Gast simulieren. Liest der Gast anschließend diese Kontroll-Register wieder aus, so muss diese Instruktion ebenfalls abgefangen werden, damit der Gast wieder den Wert sieht, den er vorher in das Register geschrieben hat, und nicht etwa den realen Wert des Kontroll-Registers, der nur für den Host sichtbar ist.

Da die x86 Architektur diese Bedingung nicht erfüllt, ist es nicht möglich, wie unter VM/370 alle Maschinenbefehle einfach im Benutzer-Modus auszuführen, und auf Programmunterbrechungen zu vertrauen, wenn auf privilegierten Maschinenstatus Information zugegriffen wird. Beispielsweise:

> ***Many models of Intel's machines allow user code to read registers and get the value that the privileged code put there instead of the value that the privileged code wishes the user code to see***

VMware's ESX Server überschreibt hierzu dynamisch Teile des Gast-Kernels und schiebt Unterbrechungsbedingungen dort ein, wo eine Intervention des Host-Kernels erforderlich ist. Als Folge hiervon tritt ein deutlicher Leistungsverlust auf, besonders bei Ein-/Ausgabe-Operationen. Manche Funktionen sind nicht vorhanden oder können nicht genutzt werden. Kompatibilitätsprobleme treten auf; es kann sein, dass bestimmte Anwendungen nicht lauffähig sind.

12.5.2 Weitere Implementierungen

Ein anderes Produkt für die x86 Plattform ist VirtualPC von Microsoft. VirtualPC wurde ursprünglich von Connectix entwickelt.

VirtualBox ist eine von der Firma innotek (von Sun Microsystems übernommen, mittlerweile zu Oracle gehörend) entwickelte Virtualisierungslösung, die es dem Benutzer erlaubt, weitere Betriebssysteme (Gäste) unter einem laufenden System (Host) zu installieren und wie eine normale Anwendung zu nutzen. Als Hostsysteme werden Windows ab XP, Mac OS X, Linux (ab Kernel 2.4) und FreeBSD (ab 7.0), als Gastsysteme neben diesen zusätzlich noch Windows NT und 2000, OS/2, DOS-basierte Betriebssysteme, Linux (ab Kernel 2.2), L4, Solaris, NetWare sowie diverse BSD-Derivate unterstützt. In einer 2010 durchgeführten Survey durch LinuxJournal.com und LifeHacker.com, war VirtualBox das populärste Virtualisierungs Produkt mit über 50% der Stimmen.

Seit dem 15. Januar 2007 steht VirtualBox in zwei Versionen zur Verfügung: Eine Open-Source-Edition (kurz: OSE), die unter der GPL v2 veröffentlicht wird, und eine PUEL-Variante (Personal Use and Evaluation License), die unter bestimmten Bedingungen kostenlos verwendet werden darf. Diese Bedingungen sind recht weit gefasst, so dass der Gebrauch am heimischen PC bedenkenlos möglich ist.

Kernel-based Virtual Machine (KVM) ist eine Virtual Machine Implementierung, die den Operating Systems Kernel benutzt. Dies ermöglicht eine bessere Performance als Lösungen, welche User-Space Driver einsetzen, In den meisten Fällen verwendet KVM eine Linux Kernel Virtualization Infrastruktur. KVM unterstützt native Virtualization für x86 Prozessoren, welche die Intel VT-x oder AMD-V Erweiterungen unterstützen. Es wurde auch auf die S390, PowerPC, und IA-64 (Itanium) Plattformen portiert. Ein ARM Port findet derzeitig statt

KVM, unterstützt zahlreiche Gastbetriebssysteme, z.B. Linux, BSD, Solaris, Windows, Haiku, ReactOS and AROS Research Operating System, Eine modifizierte Version von Qemu ermöglicht die Benutzung von Mac OS X als Gast.

Die seit Herbst 2010 verfügbare z196 Mainframe Version mit ihrer zEnterprise Blade Extension (zBX) macht umfangreichen Gebrauch von KVM.

12.5.3 Paravirtualisierung

Mehrere Projekte adressieren das Problem der fehlenden Virtualisierungseinrichtungen der Host-Architektur, indem sie das Gast-Betriebssystem abändern.

Xen ist ein GNU Open Source Software Virtual Machine Monitor, der von der Systems Research Group des University of Cambridge Computer Laboratory entwickelt wurde und ebenfalls auf der x86 Plattform läuft. Um die Probleme mit der ursprünglichen x86 Architektur zu umgehen, wird in einer Version ein als *Paravirtualization* bezeichneter Ansatz verwendet. Hierbei ist die Architektur der virtuellen Maschine nicht vollständig mit der Host Architektur identisch. Die Benutzerschnittstelle (Application Binary Interface, ABI) ist die gleiche. Der Gast-Kernel unterstellt jedoch Abweichungen zu der x86 Architektur. Dies verbessert das Leistungsverhalten, erfordert aber Änderungen des Gast-Kernels. Hiervon ist nur ein sehr kleiner Teil des Kernel-Codes betroffen. Derzeitig existiert ein funktionsfähiger Linux-Port (XenoLinux), dessen ABI mit dem eines nicht-virtualisierten Linux 2.4 identisch ist. An Portierungen für Windows XP und BSD wird gearbeitet.

Die Firma SW-Soft setzt ihre Virtuozzo Virtual Private Servers (VPS) Software vor allem für Hosting Services bei Hosting Providern ein. Als Konkurrent tritt die Firma Ensim mit ihrer konzeptuell ähnlichen Extend Software auf. SW-Soft bezeichnet seine Produkte als virtuelle Server. Der als Virtual Environment (VE) bezeichnete Gast ist vom gleichen Typ wie das Host-Betriebssystem, normalerweise Linux. Er erscheint nach außen hin wie ein kompletter Server, verfügt aber über keinen eigenen Kernel, sondern benutzt stattdessen die Kernel Funktionen des Hosts. Zwischen dem Host-Betriebssystem und den virtuellen Environments befindet sich eine Zwischenschicht, die die Steuerung und Verwaltung der VEs übernimmt. Sie erweckt den Anschein, als ob eine Gruppe von Anwendungsprozessen auf getrennten Instanzen des Kernels läuft. Die VEs sind gegeneinander abgeschottet, können unabhängig voneinander gestartet und beendet werden und besitzen je ein eigenes Dateisystem.

Ein ähnlicher Ansatz wird von Denali verfolgt. Als Gast-Betriebssystem dient Ilwaco, eine speziell an den Denali Hypervisor angepasste BSD Version. Denali unterstützt nicht das vollständige x86 ABI. Es wurde für Netzwerk-Anwendungen entwickelt und unterstellt Einzelbenutzer-Anwendungen. Mehrfache virtuelle Adressräume sind unter Ilwaco nicht möglich.

Disco ist ein Hypervisor für einen ccNUMA Cluster mit MIPS R10000 Prozessoren und IRIX 5.3 als Gast-Betriebssystem. Auch die MIPS Architektur gestattet keine vollständige Virtualisierung des virtuellen Adressenraums des Kernels. Ähnlich Xen wurde der IRIX 5.3 Gast-Kernel für einen Betrieb unter Disco leicht abgeändert.

Plex86 ist ebenfalls ein Open Source Projekt. Als Gast Betriebssystem wird ein abgeändertes Linux verwendet, welches z.B. keine Ein-/Ausgabe Unterstützung enthält. Ein-/Ausgabe Operationen werden von einem Hardware Abstraction Layer direkt an den Plex86 Hypervisor weitergereicht.

13. z/OS Betriebssystem

13.1 Work Load Manager Komponenten

13.1.1 Work Load Management für einen WWW Cluster

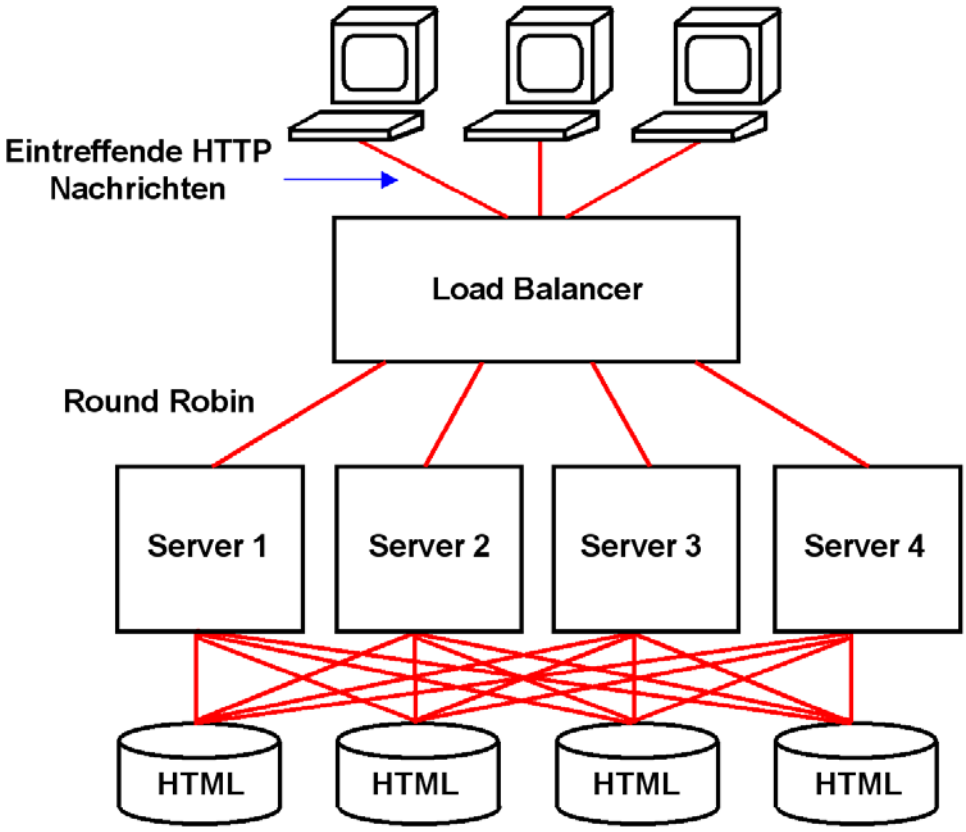


Abb. 13.1.1
Round Robin Routing für statische HTML Server

Angenommen eine Gruppe von Web Servern, die nur Anfragen bearbeiten, z.B. Google. Bei großen Providern kann die Web Site aus Hunderten oder Tausenden von Servern bestehen

Vorteil: Einheitliche Anwendung, nur Lesezugriffe zu den HTML Daten.

In diesem einfachen Fall verteilt ein vorgeschalteter Router (auch als Sprayer bezeichnet) die eintreffenden Anfragen nach dem Round Robin Verfahren auf die einzelnen Server.

Beispiel Google: Die Workload Aufteilung ist trivial einfach, weil alle Anfragen identisch sind und nur wenige unterschiedliche Server-seitige Verarbeitungsprogramme benötigt werden. Fast alle Datenbankzugriffe sind Lesezugriffe, und eine Datenintegrität der in mehreren duplizierten Datenbanken abgespeicherten Daten ist nicht erforderlich.

Typische Mainframe Aufgaben wie Datenbanken, Transaktions- und Stapelverarbeitung erfordern sehr viel komplexere Verfahren der Workload Aufteilung.

13.1.2 z/OS Work Load Management Übersicht

Dieser Abschnitt enthält eine sehr schöne Übersicht über das z/OS Work Load Management aus Wikipedia (http://de.wikipedia.org/wiki/Workload_Manager). Wir gehen später auf die hier erwähnten Eigenschaften im Detail ein.

Die in jedem Augenblick von einem z/OS System auszuführende Datenverarbeitung wird als „Work“ (Arbeit) bezeichnet. Der einem bestimmten Prozess zuzuordnende Teil davon wird als Work Unit (Arbeitseinheit) bezeichnet. Die in einem bestimmten Augenblick anfallende „Work Load“ ist die Summe aller Work Units.

Der Workload Manager (WLM) ist die z/OS Betriebssystemkomponente, die für die Arbeit auf dem Rechner den Zugang zu den Betriebsmitteln steuert. Auf einem Großrechner ist ein prioritisierter Zugang zu Betriebsmitteln (Resourcen) notwendig, da viele unterschiedliche Anwendungen den Rechner gleichzeitig nutzen und eine den Benutzerwünschen entsprechende Resourcenverteilung erfolgen muss.

Betriebsmittel sind z.B. CPU Zeit, Hauptspeicherplatz, I/O Kanalkapazität oder Plattenspeicherzugriffe. Auch Netzwerk Adapter oder Locks können Resourcen sein. WLM steuert die Betriebsmittelvergabe auf der Basis von Service Classes (Dienstklassen). Work Units werden den Service Classes über einen Klassifizierungsmechanismus zugeordnet. Die Klassifizierung wird durch den Systemadministrator des z/OS-Systems vorgenommen und kann anhand von Attributen, die für die Programmprodukte unter z/OS existieren, vorgenommen werden. Beispiele für Attribute sind Benutzernamen, Transaktionsnamen, Transaktionsklassen oder Programmnamen, die in den Anwendungen verwendet werden. Als weiteres definiert der Systemadministrator eine Zielvorgabe (Goal) für die Dienstklassen. Die Zielvorgabe kann die durchschnittliche Antwortzeit der Work Units, die in der Klasse laufen, umfassen, einen prozentualen Anteil der Work Units, die in einer bestimmten Zeit enden sollen, oder eine durchsatzorientierte Vorgabe darstellen. Welches Ziel für eine Service Class vergeben werden kann, hängt davon ab, wie viele Informationen der Workload-Manager über die Anwendungen erhält. Neben der Zielvorgabe wird jeder Dienstklasse eine Importance (Wichtigkeit) zugeordnet, die festlegt, welche Klassen bevorzugt bzw. benachteiligt werden sollen, wenn die Betriebsmittel im System nicht mehr ausreichend zur Verfügung stehen.

WLM benutzt einen Regelmechanismus, um zur Laufzeit den Zugang zu den Betriebsmitteln zu steuern. Dazu werden kontinuierlich Daten aus dem z/OS-System gesammelt. Dies sind Informationen über die Wartezustände der Work Units auf die Betriebsmittel, die Anzahl der laufenden Work Units und deren Abarbeitungszeiten. Die Informationen werden in Service Classes (Dienstklassen) zusammengefasst entsprechend der Klassifizierung, die durch den Systemadministrator vorgenommen wurde. Dann wird auf der Basis dieser Informationen die Zielerfüllung für jede Klasse berechnet und, falls notwendig, der Zugang zu den Betriebsmitteln angepasst. Die Anpassung erfolgt immer in Abhängigkeit von der Wichtigkeit (importance) der Klassen und dem Grad in dem das Ziel verfehlt wird. Das heißt, die wichtigste Klasse, die am weitesten ihr vorgegebenes Ziel verfehlt hat, wird als erste betrachtet und die Klassen mit der geringsten Wichtigkeit sind die potenziellen Kandidaten, um Betriebsmittel abzugeben. Dabei wird allerdings berücksichtigt, ob ein potenzieller Spender (Donor) auch tatsächlich das benötigte Betriebsmittel verwendet. Dieser Regelmechanismus läuft typischerweise alle 10 Sekunden unter z/OS ab; in der Zwischenzeit werden die Daten für das nächste Berechnungsintervall gesammelt. Ein Berechnungsintervall endet, wenn eine Anpassung zugunsten einer Dienstklasse durchgeführt werden kann.

WLM steuert den Zugang zu den Prozessoren und I/O-Einheiten des Systems, den Zugang zum Speicher und die Bereitstellung von Adressräumen, um Programme für bestimmte Anwendungen abarbeiten zulassen. Der Zugang zu den Prozessoren wird zum Beispiel über Dispatch Priorities geregelt. Dazu wird allen Arbeiteinheiten einer Dienstklasse (Service Class) dieselbe Priorität zugeordnet, wobei jedoch die Vergabe dieser Priorität nicht in jedem Fall mit der Definition der Wichtigkeit (Importance) der Dienstklasse übereinstimmen muss. Vielmehr orientiert sie sich an der aktuellen Auslastung des Systems, den Anforderungen der Klasse und ihrer Zielerfüllung. Dieses Verhalten des z/OS-WLM nennt man auch zielorientiertes (Goal oriented) Workload-Management, und es ist ein wichtiges Unterscheidungskriterium zu anteilsorientiertem Workload-Management, bei dem feste Zugänge zu den Betriebsmitteln vergeben werden. Letzteres findet sich häufig in Workload-Management-Komponenten von Unix-Systemen, z.B. Sun M9000 Solaris oder Superdome HP-UX.

Der WLM setzt umfangreiche adaptive Algorithmen ein um Rechner zu steuern, so dass Zielvorgaben (Business Goals) des Anwenders erfüllt werden. Im Gegensatz zu den Implementierungen auf anderen Plattformen (z.B. HP Superdome, Sun M9000) erfolgt die hier beschriebene Steuerung voll automatisch, ohne Eingriffe durch den Systemadministrator. Es wird ganz auf den Einsatz von vorgefertigten Regeln und von fest einzustellenden Parametern verzichtet . Die Steuerungsalgorithmen passen sich ebenfalls automatisch an das Umfeld an. Die hierfür erforderlichen Anpassungen an eine sich ständig ändernde Workload erfolgen automatisch mit einer Taktrate von z.B. 10 Sekunden.

Der zweite essentielle Unterschied des z/OS-WLM zu den Workload-Management Implementierungen auf anderen Plattformen ist die starke Verflechtung mit den Anwendungen und Programmprodukten, die unter einem z/OS-Betriebssystem ablaufen. So ist es durch die ständige Kommunikation zwischen dem WLM und diesen Anwendungen möglich, die Eigenschaften der Anwendungen zu erkennen und im System durch den WLM zu steuern. Dies ist bis dato auf keinem anderen System möglich, in denen jedwede Steuerung auf Prozesse begrenzt ist.

Neben der Steuerung eines Systems bietet der z/OS-WLM eine Reihe von Schnittstellen, die es Lastverteilungskomponenten erlauben, Informationen aus dem System zu erhalten, um eine intelligente Verteilung von Arbeit auf eines oder mehrere z/OS-Systeme vorzunehmen. Mehrere z/OS-Systeme können in einem Parallel Sysplex zusammengeschaltet werden, und diese Kombination wird ebenfalls unterstützt, um nach außen ein einheitliches Bild abzugeben. z/OS WLM verfügt außerdem über eine Reihe von weiteren Funktionen, die die Lastverteilung auf einem physischen System zwischen mehreren logischen Systeme unterstützen und den Zugang zu großen Plattenfarmen in Abhängigkeit von der daraufzugreifenden Arbeit steuern.

Soweit der Wikipedia Beitrag.

Zusammenfassung:

Der Work Load Manager (WLM) ist ein z/OS Alleinstellungsmerkmal. Andere Plattformen, z.B. HP Superdome oder Sun M9000, verfügen ebenfalls eine als Work Load Manager bezeichnete Komponente, die vor allem benutzt wird, um virtuelle Maschinen realen CPUs zuzuordnen. Diese haben jedoch bei weitem nicht den Funktionsumfang des z/OS WLM.

13.1.3 Traditionelle Work Load Management Verfahren

Es existieren viele Möglichkeiten und Einstellungsalternativen, die Ausführung der Work Load zu steuern. Beispiele sind:

- **Programme laufen Run-to-Completion, oder unterliegen einer Zeitscheibensteuerung,**
- **Die Anzahl der gleichzeitig aktiver Prozesse kann vergrößert oder verkleinert werden, um z.B. jedem Prozess einen optimalen Umfang an realem Speicher zuzuordnen. Prozesse können zeitweise ausgelagert werden, wenn Ressourcen knapp werden,**
- **Multithreading, Anzahl von Subsystem Instanzen,**
- **In einem Sysplex können Prozesse bestimmten physischen Servern optimal zugeordnet werden,**
- **I/O Kanäle können den LPARs optimal zugeordnet werden,**
- **Prioritäten können angepasst werden,**
- **usw.**

Die Komplexität wächst mit der Systemgröße. Problembereiche sind

- **Stapelverarbeitung und interaktive Verarbeitung müssen gleichzeitig laufen,**
- **Belastungsschwankungen treten auf (von Minute zu Minute, während des Tages, innerhalb einer Woche),**
- **Die Affinität der Prozesse zu ihren Daten soll optimiert werden,**
- **Die Zuordnung der Prozesse zu realen CPUs muss erfolgen,**
- **Prozesse erzeugen unterschiedliche I/O Anforderungen und I/O Belastungen.**

In einer IT-Umgebung ohne Mainframe ist es üblich, unterschiedlichen Anwendungen unabhängige physische Server zuordnen. Die Folge ist eine schlechte Auslastung der Server (typisch 20 % CPU-Auslastung). Dies erscheint akzeptabel, denn die Hardware ist billiger als bei einer Mainframe Lösung. Jedoch die Folgen sind:

- **Viele Server (tausende in großen Unternehmen),**
- **Heterogene IT-Landschaft,**
- **Komplexe LAN Strukturen,**
- **Hoher Administrationsaufwand,**
- **Hoher Energieverbrauch und Kühlungs-/Klimatisierungsaufwand.**

Vor allem die hohen Administrationskosten machen diese Lösung zunehmend unattraktiv.

In einer Mainframe Installation sind Energieverbrauch und Kühlungs-/Klimatisierungsaufwand deutlich geringer, da eine CPU Auslastung im 90 – 100% Bereich möglich ist. Dies ermöglicht der z/OS Workload Manager. Weiterhin ist der Administrationsaufwand deutlich geringer, wozu die Tivoli und Unified Resource Manager (siehe Abschnitt 14.3.13) Komponenten beitragen.

Für die weitere Diskussion müssen Sie sich mit einer Reihe von Begriffen vertraut machen:

Work	**andere Bezeichnungen Datenverarbeitung. Data Processing. Ist das, was ein Rechner durchführt.**
Work Unit	**ein Stück Datenverarbeitung, z.B. Durchführung einer Transaktion oder eines Stapelverarbeitungsjobs.**
Work Load	**= ∑ Work Units , die Summe aller Work Units, die ein Rechner in einem gegebenen Zeitpunkt bewältigt.**
Resourcen	**die Betriebsmittel, über die ein Rechner verfügt. Beispiele sind CPU Zyklen, Hauptspeicherplatz, I/O Kanal Kapazität, Plattenspeicherzugriffe**
Service Unit	**Resourcenverbrauch einer Work Unit.**
Service Class	**(Dienst Klasse), Gruppe von Work Units mit ähnlichen Vorgaben wie Antwortzeit (Response Time), Turn-around-Time, Priorität, usw.**
Goal	**andere Bezeichnung Business Goal, Zielvorgabe für eine Service Class, z. B. 90 % aller Transaktionen einer bestimmten Service Class sollen eine Response Time < 0,3 Sekunden aufweisen.**

Unter System Resourcen versteht man Elemente wie

- **Verarbeitungskapazität der individuellen CPUs,**
- **Platz im Hauptspeicher,**
- **Channel Subsystem Verarbeitungskapazität,**
- **I/O Kanal Übertragungskapazität,**
- **Plattenspeicherzugriffe,**
- **...**

Der Workload Manager hilft, alle System Resourcen optimal auszunutzen.

Anmerkung: Die deutsche Sprache verwendet die Schreibweise Ressource. Die engliche Schreibweise ist Resource. Wir benutzen den Begriff Ressource/Resource teilweise alleinstehend und teilweise als Teil eines Begriffes, z.B. „System Resource Manager“.

Die unterschiedliche Schreibweise ist verwirrend. Wir verwenden deshalb ausschließlich die englische Schreibweise.

Die Verwendung der deutschen Übersetzung „Betriebsmittel“ ist im Zusammenhang mit WLM eher ungebräuchlich.

13.1.4 z/OS Work Load Manager Komponenten

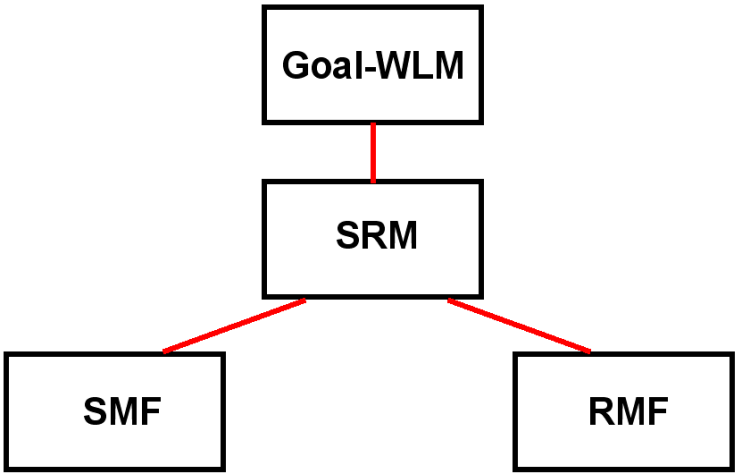


Abb. 13.1.2
Die Komponenten des Work Load Managers

Es existieren vier z/OS Work Load Manager Komponenten:

- **Der Goal oriented Work Load Manager Goal-WLM stellt die Schnittstelle zum Systemadministrator dar und erlaubt eine voll-automatische Ablaufsteuerung-**
- **Der System Resource Manager (SRM) ist eine Komponente des z/OS Kernels, welche die eigentlichen Anpassungen vornimmt.**
- **Die System Management Facility (SMF) sammelt System-relevante Information über die Konfiguration, die Auslastung der einzelnen Systemkomponenten (z.B. Hauptspeicher, CPUs, paging/swapping Aktivitäten, I/O Aktivitäten usw. Die Ergebnisse werden in SYS1.MANn Data Sets festgehalten.**
- **Die Resource Measurement Facility (RMF) erstellt Durchschnittswerte über festgelegte Zeitintervalle und macht sie als Input für die erwähnte Tivoli Komponente verfügbar.**

Im allgemeinen Sprachgebrauch wird unter dem Begriff Work Load Manager (WLM) die Summe aller 4 Komponenten verstanden. Häufig versteht man unter WLM aber auch nur die oberste Komponente, den Goal Oriented Work Load Manager Goal-WLM.

13.1.5 System Resource Manager

Die System Resource Manager Komponente des Work Load Managers beobachtet für alle angeschlossenen Systeme:

- **CPU Auslastung**
- **Hauptspeicher Nutzung**
- **I/O Belastung**

Der System Resources Manager stellt sicher dass:

- **die System Resourcen möglichst voll genutzt, aber nicht übercommitted werden, und**
- **alle Benutzer einen fairen Anteil der System Resource erhalten.**

SRM überwacht die Nutzung der Resourcen und ordnet sie periodisch neu zu. Wenn eine Resource schlecht ausgelastet ist, versucht SRM die Systembelastung zu erhöhen. Wenn eine Resource überlastet ist, versucht SRM die Systembelastung zu verringern.

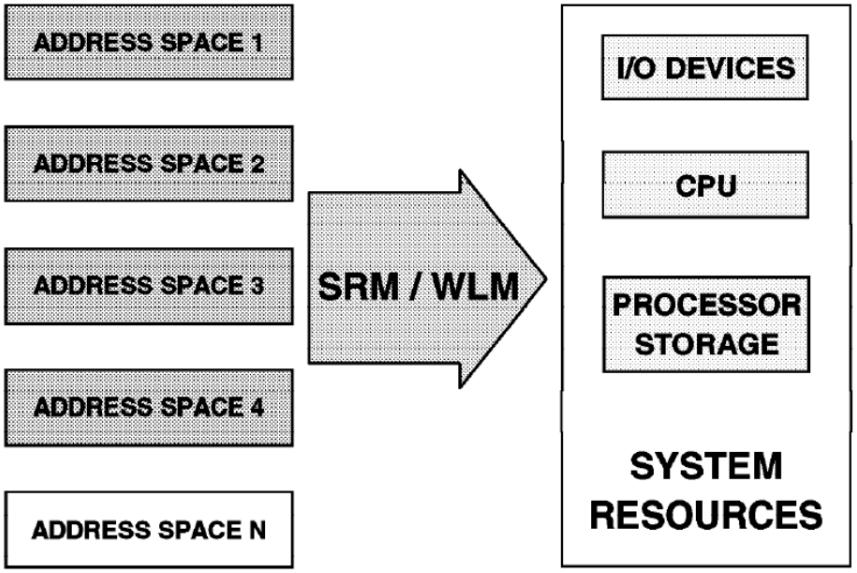


Abb. 13.1.3
System Resources Manager

Prozesse sind entweder aktiv (ihnen ist ein Adressenraum mit realem Hauptspeicherplatz zugeordnet), oder sie sind inaktiv (auf einen Plattenspeicher ausgelagert).

SRM bestimmt, welchen aktiven Prozessen (bzw. ihren Adressenräumen) welche System Resourcen (und wie viel davon) zugeordnet werden. SRM entscheidet dies um zwei grundsätzlich miteinander konkurrierende Ziele zu erreichen:

1. **System Resources werden einzelnen Prozessen zugeteilt um die für die Installation festgelegten Ziele in Bezug auf Antwort-Zeit, turnaround-Zeit und Prioritätsanforderungen optimal zu erfüllen**

2. **System Resourcen werden optimal genutzt um einen möglichst hohen Systemdurchsatz zu erreichen.**

13.1.6 System Management Facility

Die System Management Facility (SMF) überwacht kontinuierlich, welche Betriebsmittel eine Service Class (Gruppierung von Work Units mit ähnlichen Anforderungen) 1. benutzt und 2. auf welche sie wartet.

SMF sammelt Daten über den aktuellen Zustand aller verwalteten Betriebsmittel. Dies sind Information über

- **die Prozessoren,**
- **den Speicher und**
- **die Nutzung der Platteneinheiten und Kanäle.**

Aus den gesammelten Daten wird für jede Work Unit festgestellt, ob sie Betriebsmittel benutzt oder darauf wartet. Der Workload-Manager verwendet diese Daten, um den Zugang der Service-Klassen zu den Betriebsmitteln zu regeln.

Die System Management Facility (SMF) bewirkt, dass System-bezogene Information über die Konfiguration, die Workload und paging/swapping Aktivitäten in SYS1.MANn Data Sets gesammelt werden.

SMF Records des Job Entry Subsystems werden ebenfalls in den SYS1.MANn Data Sets gespeichert. Diese Records enthalten Information über die Start- und Stopzeiten, Prozessor Aktivitäten und Hauptspeichernutzung für jeden Job Step und jede TSO Session. Hilfsprogramme analysieren die Daten und erstellen Berichte.

13.1.7 Resource Measurement Facility

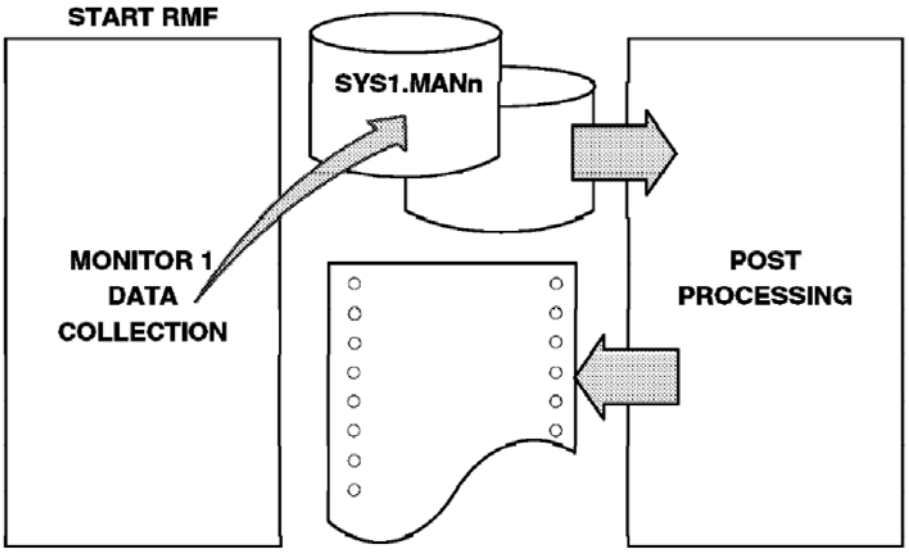


Abb. 13.1.4
RMF Monitor

Die Resource Measurement Facility (RMF) ist ein eigener Prozess in einem eigenen Adressenraum.

Ein RMF Monitor sammelt System Daten, z.B. CPU Auslastung, DASD Aktivitäten, Belastung der Kanäle, Hauptspeicher Auslastung und Nutzung sowie andere Aktivitäten. RMF erstellt Zusammenfassungen und Berichte hierüber. Der System Administrator bestimmt die Größe der Messintervalle.

13.1.8 Goal oriented Work Load Manager

Der System Resource Manager ist auf statische Parameter des System Administrators angewiesen und kann sich nur in geringem Umfang an dynamische Änderungen anpassen.

Der Goal Oriented Work Load Manager verfügt über umfangreiche adaptive Algorithmen um die Betriebsmittel abhängig von der sich dynamisch ändernden Systemlast zu steuern, ohne erforderliche Eingriffe durch den Systemadministrator.

Der z/OS Goal oriented Work Load Manager (Goal-WLM) erweitert den Funktionsumfang des SRM, und dehnt ihn auf den ganzen Sysplex aus. Er stellt eine Shell für den System Resource Manager (SRM) dar. Goal-WLM ist eine SRM Erweiterung, welche die existierenden SRM Mechanismen für die Zuordnung vom Resourcen zu Work Units um Performance Management Funktionen erweitert.

Goal oriented Workload Management stellt eine Veränderung der Fokussierung dar, von

- **tuning auf der System Resource Ebene, nach**
- **Definitionen des erwarteten Leistungsverhaltens.**

Diese Eigenschaften sind es, was den z/OS WLM von Work Load Management Funktionen in anderen Betriebssystem unterscheidet.

13.2 System Resource Manager

13.2.1 Resource Management

Prozesse, die in einem z/OS System laufen, benötigen Betriebsmittel (Resourcen)

- **CPU Zeit**
- **Hauptspeicherbedarf (Rahmen im Hauptspeicher)**
- **I/O Operationen**

Es existieren große Unterschiede, in welchem Umfang die einzelnen Prozesse Resourcen benötigen. Manche Prozesse benötigen viel CPU Zeit, aber wenig Hauptspeicherplatz. Bei anderen Prozessen ist es umgekehrt.

Der Resourcen Verbrauch eines Prozesses wird in Service Units (Resourcenverbrauch einer Work Unit) gemessen. Es existieren unterschiedliche Service Unit Definitionen für die CPU Zeit, den Hauptspeicherbedarf und die I/O Operationen.

Wir schauen uns das Resourcen Management an Hand der Hauptspeicherverwaltung näher an.

13.2.2 Flattern (Thrashing)

Auf einem Rechner laufen gleichzeitig zahlreiche Prozesse. Sie wechseln zwischen den Zuständen laufend, wartend und ausführbar. Wir bezeichnen diese Prozesse als aktive Prozesse (siehe Einführung in z/OS, Verarbeitungsgrundlagen, Teil 1, http://jedi.informatik.uni-leipzig.de/de/Vorles/Einfuehrung/Grundlag/vg01.pdf#page=12). Normalerweise steigt die Auslastung der CPUs, je größer die Anzahl der aktiven Prozesse ist.

Auf einem Rechner laufen gleichzeitig zahlreiche Prozesse. Sie wechseln zwischen den Zuständen laufend, wartend und ausführbar. Wir bezeichnen diese Prozesse als aktive Prozesse (siehe Band 1, Abschnitt 2.1.9) Normalerweise steigt die Auslastung der CPUs, je größer die Anzahl der aktiven Prozesse ist.

Jeder aktive Prozess beansprucht realen Hauptspeicher für seine aktiven Seiten (im Hauptspeicher abgebildet). Nur ein Teil der Seiten des virtuellen Speichers ist in jedem Augenblick in Rahmen des realen Hauptspeichers abgebildet. Der größere Teil der Seiten eines virtuellen Speichers ist in jedem Augenblick auf einem externen Seitenspeicher ausgelagert. Der reale Speicher besteht somit aus 2 Teilen: dem realen Hauptspeicher und dem externen Seitenspeicher (pagefile.sys unter Windows). Beim Zugriff zu einer ausgelagerten Seite (nicht in einem Rahmen des Hauptspeichers abgebildet) erfolgt eine Fehlseitenunterbrechung (Page Fault). Diese bewirkt den Aufruf einer Komponente des Überwachers (Seitenüberwacher, Paging Supervisor), der die benötigte Seite aus dem externen Seitenspeicher holt und in den Hauptspeicher einliest. In den meisten Fällen muss dafür Platz geschaffen werden, indem eine andere Seite dafür auf den externen Seitenspeicher ausgelagert wird. Dieser Vorgang wird als Demand Paging bezeichnet.

Wenn Ihr Windows-, Linux oder Apple Rechner genügend viel Hauptspeicher hat, können evtl. 100 % der Seiten eines virtuellen Speichers in Rahmen des realen Hauptspeichers abgebildet werden. Bei einem Mainframe mit bis zu mehreren TByte Hauptspeicher ist dies fast nie der Fall.

Steigt die Anzahl der aktiven Prozesse, so wird der Prozentsatz der Seiten immer kleiner, die für jeden Prozess in Rahmen des realen Hauptspeichers (an Stelle des externen Seitenspeichers) abgebildet werden. Je kleiner der Prozentsatz, um so größer ist die Wahrscheinlichkeit einer Fehlseitenunterbrechung. Die Seitenfehlerrate steigt umso mehr an, je geringer die Anzahl der im Hauptspeicher verfügbaren Rahmen ist.

Wird die Anzahl zu gering, tritt ein als „Flattern“ (Thrashing) bezeichnetes Problem auf. Die Prozesse stehlen sich gegenseitig benötigte Rahmen und können auf Grund zahlreicher Fehlseitenunterbrechungen kam noch produktive Arbeit leisten.

13.2.3 Auslagern von Prozessen

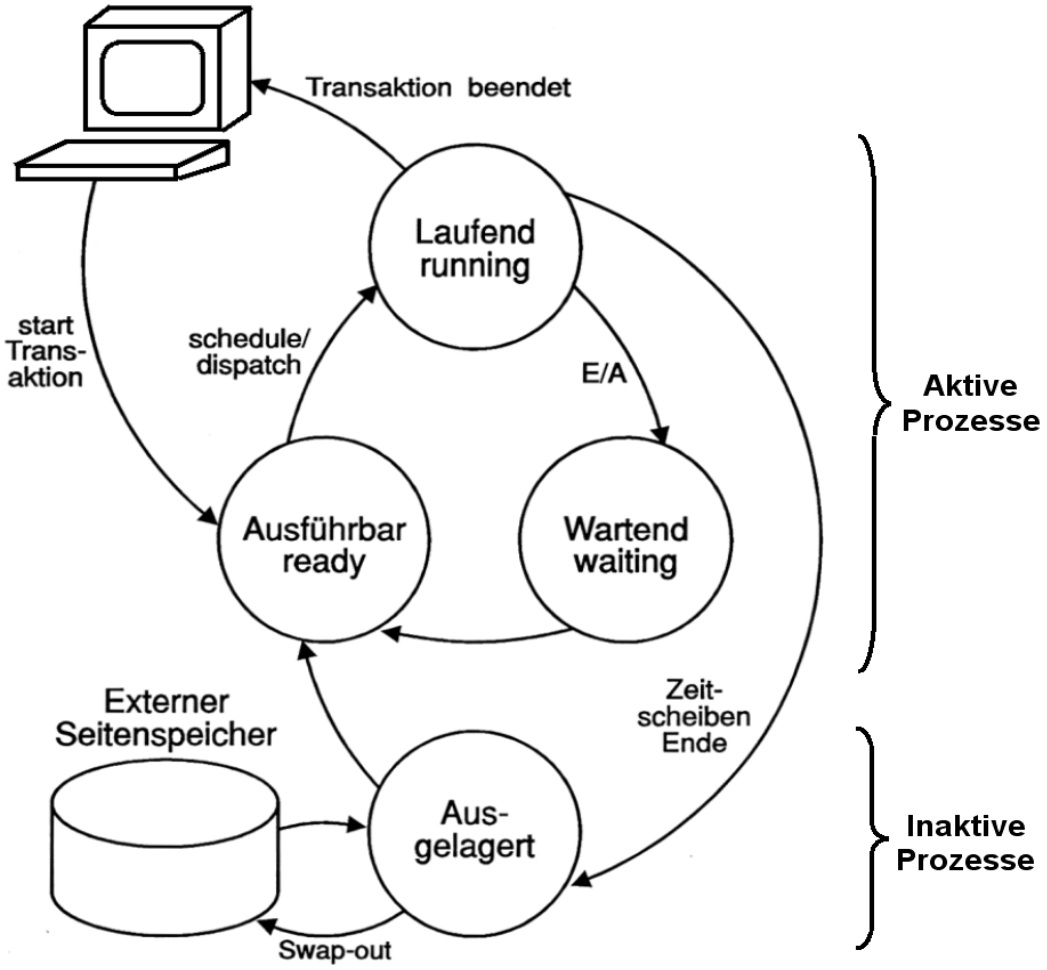


Abb. 13.2.1
Aktive und inaktive Prozesse

Aktive Prozesse beanspruchen Rahmen im realen Hauptspeicher.

Inaktive Prozesse sind auf den externen Seitenspeicher (auxiliary Storage) ausgelagert und verfügen über keine Rahmen im realen Hauptspeicher.

Verringert man die Anzahl der aktiven Prozesse indem man die Anzahl der (ausgelagerten) inaktiven Prozesse erhöht, stehen den restlichen aktiven Prozessen mehr Rahmen im realen Hauptspeicher zur Verfügung. Die Seitenfehlerrate sinkt.

Wenn die Anzahl der aktiven Prozesse zu klein ist, werden möglicherweise andere Resourcen, z.B. CPU Zeit schlecht ausgenutzt.

13.2.4 Begrenzung der Seitenfehlerrate

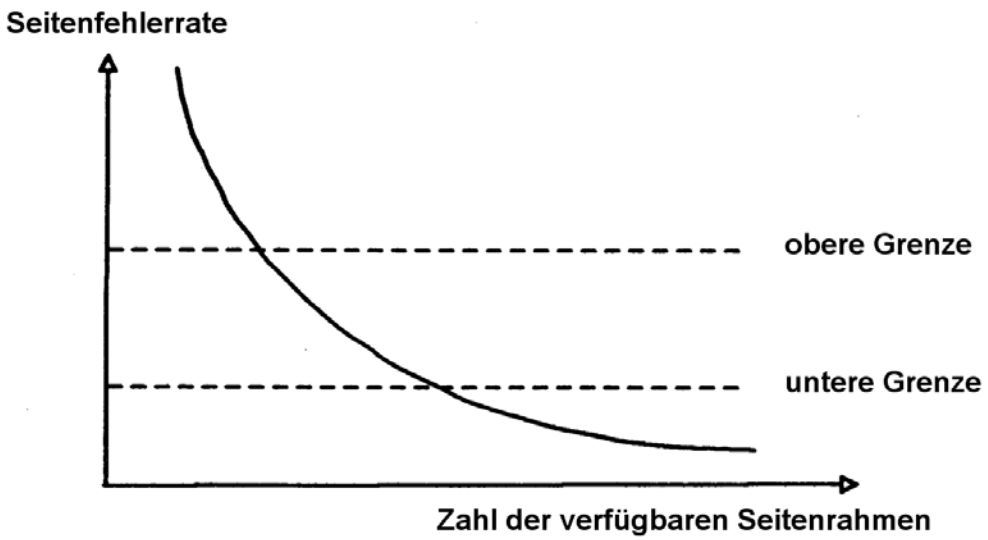


Abb. 13.2.2
Eingrenzung der Seitenfehlerrate

Um Flattern zu verhindern, beobachtet der Workload Manager ständig die Anzahl der Fehlseitenunterbrechungen pro Zeitintervall (z.B. alle 10 Sekunden). Übersteigt die Anzahl der Fehlseitenunterbrechungen eine festgelegte obere Grenze, reduziert WLM die als „Multiprogramming Level“ bezeichnete Anzahl der aktiven Prozesse. Einige Prozesse werden inaktiviert, indem sie mit einem als Swap-out bezeichneten Verfahren ganz auf den externen Seitenspeicher ausgelagert werden; sie verlieren alle Rahmen im realen Hauptspeicher. Den übrigen aktiven Prozessen stehen damit mehr Hauptspeicherrahmen zur Verfügung, und die Seitenfehlerrate sinkt. Es werden solange Prozesse mit niedriger Priorität inaktiviert, bis die Seitenfehlerrate auf einen akzeptablen Wert sinkt.

Unterschreitet die Seitenfehlerrate eine festgelegte untere Grenze, wird WLM die Anzahl der aktiven Prozesse wieder erhöhen (Swap-in), um die Auslastung der CPUs und/oder anderer Resourcen zu verbessern.

13.2.5 Wirkungsweise des Seitenwechsel-Begrenzers

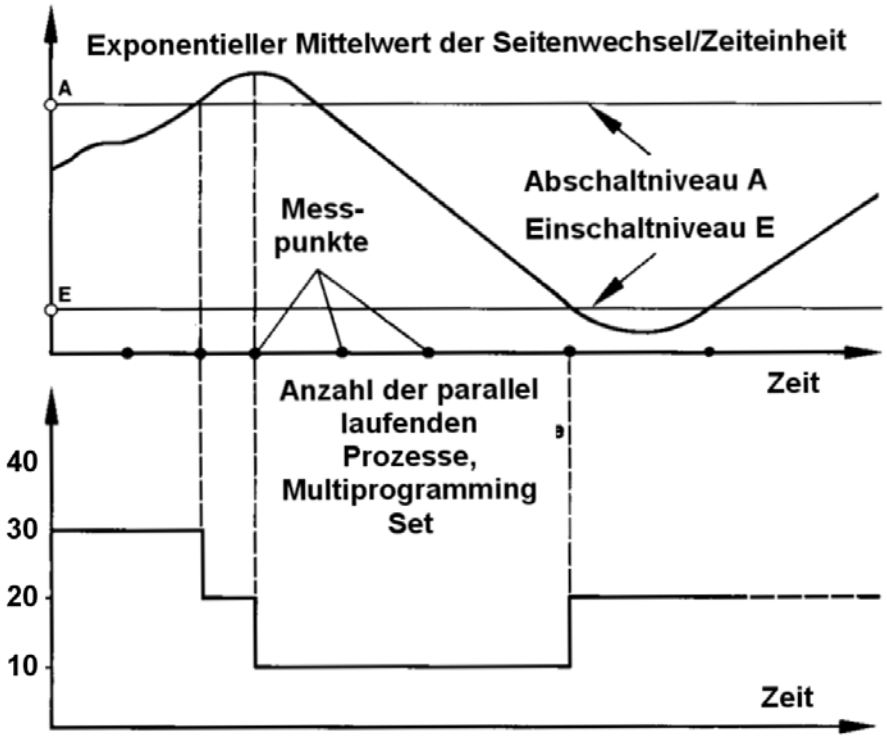


Abb. 13.2.3
Die Multiprogramming Set Größe wird entsprechend der Seitenfehlerrate angepasst

Als Multiprogramming Set wird die Menge der derzeitig aktiven Prozesse bezeichnet.

Dargestellt ist, wie über festgelegte Zeitintervalle die durchschnittlich Seitenfehlerrate ermittelt wird.

Übersteigt die durchschnittliche Seitenfehlerrate den Grenzwert A, so wird die Multiprogramming Level (Anzahl der Prozesse im Multiprogramming Set) reduziert.

Unterschreitet die durchschnittliche Seitenfehlerrate den Grenzwert E, so wird die Multiprogramming Level (Anzahl der Prozesse im Multiprogramming Set) wieder erhöht.

Als Target Multiprogramming Level (TMPL) bezeichnet man die Anzahl der swapped-in Address Spaces (aktive Prozesse), die nach der Meinung von WLM das System mit seinen verfügbaren Resourcen optimal bedienen kann.

Dieser Wert ändert sich ständig..

Die Current Multiprogramming Level (CMPL) ist im Gegensatz zu TMPL die tatsächliche Anzahl der swapped in Address Spaces. Diese befinden sich untereinander im Wettbewerb um die verfügbaren System Resourcen. CPML ist identisch mit der Anzahl der aktiven Prozesse.

WLM bemüht sich, den Unterschied zwischen CMPL und TMPL möglichst klein zu halten.

13.2.6 Prioritätssteuerung für inaktive Prozesse

Abb. 13.2.4
Entscheidungsparameter für Swap-in bzw. Swap-out

WLM entscheidet unter Prioritätsgesichtspunkten, welcher aktive Prozess für ein Swap-out selektiert wird bzw. welcher inaktive Prozess für ein Swap-in selektiert wird. Die zugeordneten Prioritäten können sich ständig ändern. Ist z.B. zu einem Zeitpunkt die CPU Auslastung hoch, I/O Kapazität ist aber verfügbar, dann wird WLM die Priorität von CPU-lastigen Prozessen heruntersetzen, und die Priorität von I/O lastigen Prozessen heraufsetzen.

13.2.7 Automatische Steuerung der System-Resourcen

Die unterschiedlichen Arten von Prozessen haben unterschiedliche Zielsetzungen. So sind z.B. für viele interaktive CICS oder Webserver Zugriffe optimal Antwortzeiten (gemessen in Sekundenbruchteilen) auf Kosten einer optimalen Resourcen Ausnutzung erwünscht. Bei Stapelverarbeitungsprozessen ist es umgekehrt. Es kann aber auch sein, dass für bestimmte Stapelverarbeitungsprozesse eine optimale Durchlaufzeit erwünscht ist.

Diese Zielsetzungen müssen dem System Resource Manager (SRM) vom Systemadministrator in irgend einer Form mitgeteilt werden. Die Zusammenhänge zwischen den unterschiedlichen Zielsetzungen sind jedoch sehr komplex, und ändern sich im Laufe einer Stunde, eines Tages oder Monats ständig. Mit der zunehmenden Komplexität wird es für den Systemadministrator immer schwieriger zu entscheiden, welche von zahlreichen Einstellparametern er für einen optimalen Betrieb ändern muss. Oft kann er auch gar nicht mehr schnell genug reagieren.

Immer häufiger tritt auch das Problem auf, dass die Änderung von Systemparametern unerwartete und unerwünschte Seiteneffekte erzeugt. Bei der Vielzahl der Einstellungsmöglichkeiten ist es denkbar, dass eine bestimmte Änderung keine erwartete Verbesserung, sondern eine Verschlechterung bringt.

Als Lösungsansatz wird dem Rechner die automatische Verwaltung und Steuerung aller System Resourcen übertragen. Hierzu muss ihm aber gesagt werden, unter welchen Gesichtspunkten die Optimierung erfolgen soll.

Dies ist die Aufgabe des Goal Oriented Work Load Managers.

Abb. 13.2.5
Zusammenspiel der WLM Komponenten

Vor der Einführung des Goal orientierte Workload-Managers existierte bereits der System Resource Manager (SRM). SRM ist in der Lage, die Verteilung der Betriebsmittel abhängig von der anfallenden Systemlast zu steuern. Die Verteilung der Betriebsmittel erfolgte abhängig von der anfallenden Systemlast.

Hierzu sammelt die System Management Facility (SMF) System-relevante Information über die Auslastung der einzelnen Systemkomponenten. SMF misst die gesamte Systemleistung und beobachtet die Nutzung und Auslastung der System Resourcen (z.B. CPU, Hauptspeicher, Ein/Ausgabe).

Engpässe an Resourcen können von SRM zum Beispiel durch eine Verringerung des Multiprogramming Level und durch das Swapping von Adressenräumen aus dem Hautspeicher in den externen Seitenspeicher aufgelöst werden.

Die Steuerung erfolgte durch die mehr oder weniger statische Eingabe von SRM Konfigurationsparametern durch den Systemadministrator. Diese ließen sich in der Vergangenheit nur schlecht den sich dynamisch ändernden Gegebenheiten anpassen. Weiterhin ist es schwierig, bestimmte angestrebte Ziele (Business Goals, z.B. alle Transaktionen vom Typ x sollen eine Antwortzeit < 0,3 s haben) in SRM Konfigurationsparameter (z.B. Target Multiprogramming Level ≤ y) zu übersetzen.

13.2.8 Goal Orientierung

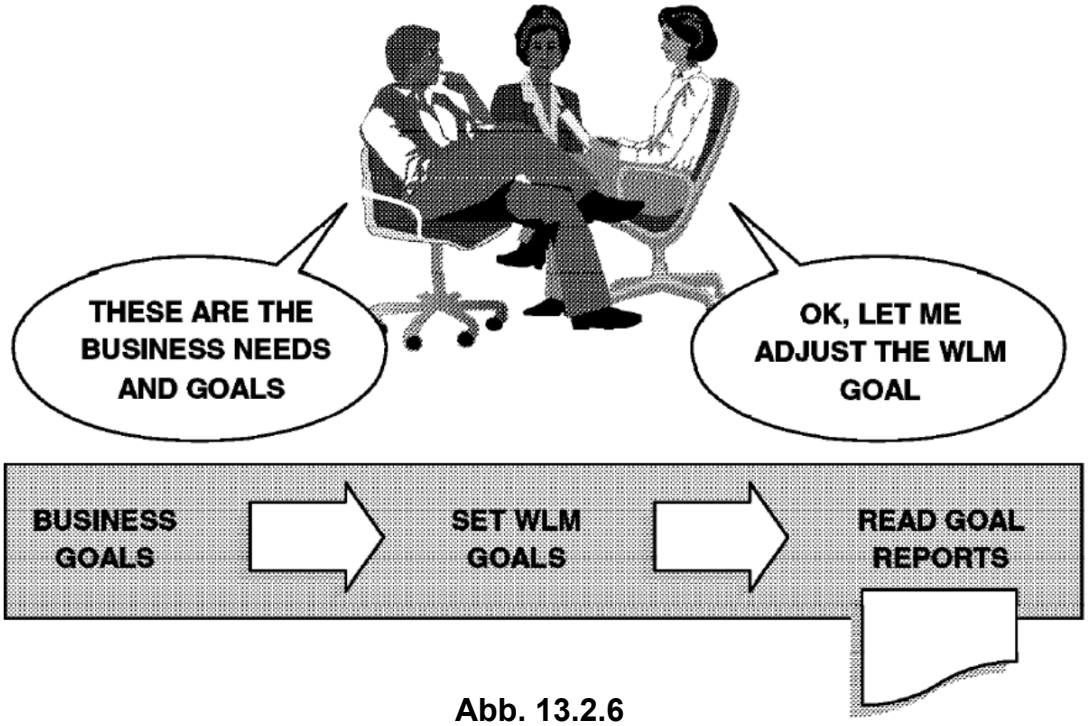


Abb. 13.2.6
Die Festlegung der Goals erfolgt unabhängig von Kenntnissen über die Systemstruktur

Der Goal oriented Work Load Manager (WLM) reduziert den Aufwand für den System Administration und die Notwendigkeit für Detailwissen

Der Goal oriented Workload Manager ist ein integraler Bestandteil des z/OS-Betriebssystems. Er erweitert den Funktionsumfang des SRM, und dehnt ihn zusätzlich auf den ganzen Sysplex aus. Es ist die Aufgabe des Goal oriented Workload Managers, die von menschlichen Benutzern gewünschte Optimierungs-Anweisungen in der Form von Geschäftsbegriffen (Business Goals, wie z.B. erwünschte Antwortzeit) entgegen zu nehmen. Das Leistungsverhalten der Installation wird durch Vorgaben für Zielsetzungen (Business Goals) festlegt. Der Goal oriented Workload Manager übersetzt diese Anweisungen in System Resource Manager (SRM) Parameter und passt sie dynamisch an die sich laufend ändernde Systembelastung an.

z/OS Subsysteme brauchen zusätzliche Funktionen, um mit WLM kooperieren zu können. Der Goal oriented Workload-Manager unterstützt beispielsweise folgende Subsysteme :

- **IMS**
- **JES2/JES3**
- **TSO/E**
- **CICS**
- **OMVS**
- **DB2**
- **TCP**
- **SNA**
- **HTTP Server**
- **WebSphere**
- **MQSeries**
- **non-z/OS LPARs, z.B zLinux**

Zur Integration der Subsysteme enthalten manche Subsysteme ihre eigenen Work Load Management Komponenten. Beispiele sind CICS und WebSphere. CICS Transaktionen können nach Ihrem Namen (TRID) klassifiziert werden. Die z/OS WebSphere Version unterscheidet sich von den WebSphere Versionen auf anderen Betriebssystem Plattformen (distributed WebSphere) unter Anderem dadurch, dass alle z/OS WLM Funktionen in Anspruch genommen werden können.

13.2.9 Service Klassen

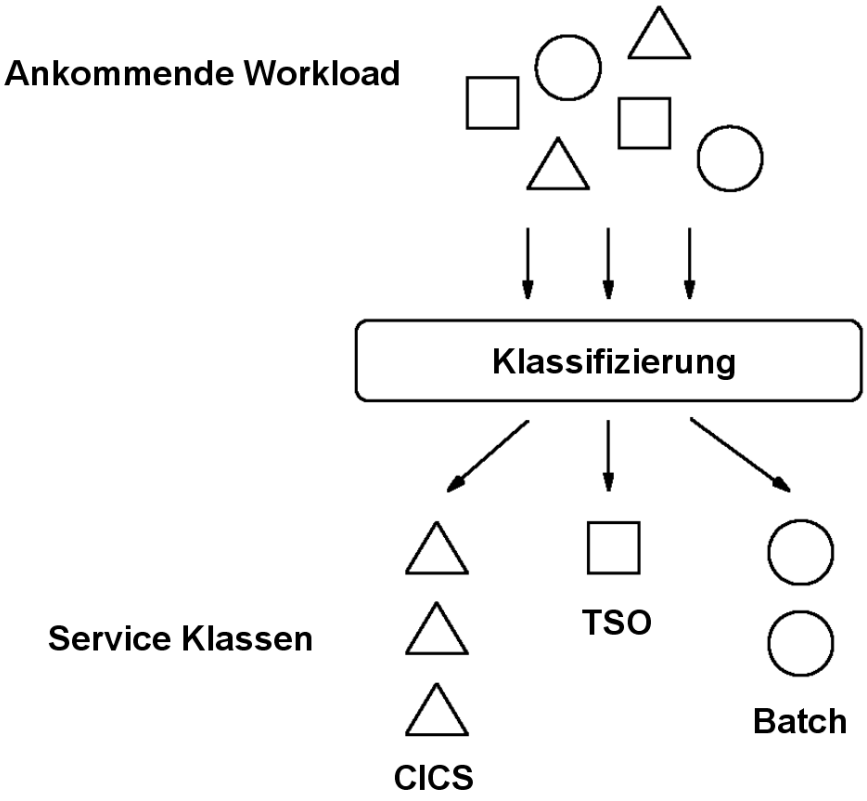


Abb. 13.2.7
Einordnung aller Work Units in Service Klassen

Die von einem Rechner zu verarbeitende Workload besteht aus vielen einzelnen Work Units (Arbeitseinheiten). Eine Work Unit oder Arbeitseinheit ist eine Verarbeitungsaufgabe für einen Rechner. Beispiele für Work Units sind

- **CICS Transaktionen,**
- **TSO Sitzungen,**
- **Batch Jobs.**

In einem z/OS Rechner werden in jedem Augeblick zahlreiche Work Units parallel verarbeitet. Eine Work Unit wird normalerweise durch einen Prozess ausgeführt. CICS Transaktionen sind ebenfalls Work Units. Bei der Stapelverarbeitung werden Work Units als Jobs bezeichnet.

Der Benutzer definiert mittels Zielvorgaben, welche Leistungen unter welchen Umständen von dem Rechnersystem zu erbringen sind. Das System überwacht während der Abarbeitung die Workloads, vergleicht ihre Laufzeitergebnisse mit den Zielvorgaben (Business Goals) und passt den Zugang zu und Verbrauch an Resourcen dynamisch auf der Basis der Vorgaben (Goals) an.

Theoretisch wäre es denkbar, für jede einzelne Work Unit eigene Zielvorgaben zu erstellen. Da dies unpraktisch ist, fasst man Work Units mit ähnlichen Zielvorgaben zu „Service Klassen“ zusammen. Alle in einer Service Klasse zusammengefassten Work Units erhalten die gleichen Zielvorgaben.

Service-Klassen (Dienst Klassen) sind Einheiten von Workloads mit ähnlichen Charakteristiken, für die die Zielvorgaben definiert werden. Eine mögliche - sehr einfache - Aufteilung der Work Units in Service-Klassen könnte z.B. drei Klassen vorsehen:

- **CICS Jobs,**
- **TSO Jobs,**
- **Batch Jobs**

Es ist aber auch möglich, z.B. CICS Jobs entsprechend ihrer User ID oder ihrer TRID in mehrere unterschiedliche Service-Klassen aufzuteilen

Service-Klassen (Dienst Klassen) sind Einheiten von Workload mit ähnlichen Charakteristiken, für die die Zielvorgaben definiert werden.

Die Service-Klasse stellt das Grundkonstrukt für den Goal oriented Workload Manager dar. dar. Sie ist das Ergebnis der Klassifizierung der Workloads mit unterschiedlichen Leistungsmerkmalen in Gruppen, die mit den gleichen Ziel-Vorgaben versehen werden können.

Die Anzahl der Service Klassen sollte nicht zu niedrig (schlechte Optimierung) und nicht zu hoch (hoher Administrationsaufwand und WLM Verarbeitungsaufwand) sein. Eine Anzahl von 25 - 30 Service Klassen ist in vielen Installationen ein guter Wert.

13.2.10 Einordnung in Service-Klassen

Classification Rules werden benutzt, um die Menge aller möglichen Arbeitsanforderungen in Serviceklassen (Dienstklassen) einzuordnen (klassifizieren). Die Klassifikation basiert auf den Attributen einer individuellen Arbeitsanforderung. Dies kann z.B. sein:

- **User ID**
- **Art des aufgerufenen Prozesses (Transaktionstyp, Stapel,)**
- **Standort oder Art des Terminals oder Klientenrechners (z.B. Standort Personalabteilung, Art Autorisierung = xxx)**
- **Accounting Information**
- **.....**

Jeder Serviceklasse sind „Ziele“ zugeordnet

- **Antwortzeit (Response Time)**
- **Geschwindigkeit (Velocity)**
- **Stellenwert (Importance)**
- **andere (discretionary)**

Jede Serviceklasse besteht aus zeitlichen Perioden (z.B. 10 Sekunden):

- **Während einer Periode sind begrenzte Resourcen verfügbar (z.B. CPU Zyklen, I/O Zugriffe, ...)**
- **Nach Ablauf der Periode `i` erfolgt eine Migration nach Periode `i + 1`**
- **Für jede Periode können unterschiedliche Ziele existieren.**

Eine realistische Annahme ist, dass nicht alle Ziele erreicht werden können. WLM platziert Arbeitsanforderungen so, dass die Wahrscheinlichkeit, alle Ziele zu erreichen, optimiert wird.

Die Einordnung aller Service Units in Service Klassen wird dem System in der Form einer Workload-Manager „Service Definition“ mitgeteilt. Hierzu benutzt der System Administrator eine speziellen administrative Anwendung, die „WLM Administrative Application“. Diese kann unter der Interactive System Productivity Facility (ISPF) von autorisierten TSO-Benutzern gestartet werden.

Der wichtigste Punkt bei der Definition einer Service-Klasse ist es festzulegen, wie wichtig (important) sie ist, um die übergeordneten Geschäftsziele zu erreichen. Entscheidend ist, dass manche Service-Klassen weniger wichtig eingestuft werden als andere. Bei einem optimierten System kann es durchaus sein, dass nicht alle Ziele erfüllt werden können.

Die Wichtigkeit wird dargestellt, indem eine Business Importance für die Service-Klasse definiert wird. Diese Business Importance ist später für das System der wesentliche Entscheidungsfaktor, wenn der Zugang zu den Resourcen geregelt/optimiert werden muss.

13.3 Goal Management

13.3.1 Work Load Manager Operation

Die System Management Facility (SMF) Komponente des Workload-Managers sammelt Daten über den aktuellen Zustand aller durch WLM verwalteten Resourcen. Dies sind Information über die Prozessoren, den Hauptspeicher, die Benutzung der I/O Einrichtungen und die von WLM für die von Anwendungen verwaltete Warteschlangen. Das Sampling Intervall beträgt z.B. 250 Millisekunden. Zeiträume, in denen die Work Unit nicht gearbeitet hat und auch keine Anforderungen an das System gestellt hat, werden nicht berücksichtigt. Dasselbe gilt für Zustände, die vom Workload-Manager nicht erfasst werden können, wie zum Beispiel das Warten auf die Freigabe eines Locks, das durch eine Anwendung verwaltet wird.

Aus den gesammelten Daten wird für jede Work Unit festgestellt, ob sie Resourcen benutzt (Using) oder darauf wartet (Delay). Der Workload-Manager verwendet diese Daten, um die Verteilung von Resourcen auf Service Classes vorzunehmen. Da sie die Basis für die WLM-Algorithmen darstellen, können sie auch für die Definition von Zielen verwendet werden.

Für die Definition von Zielvorgaben bietet der z/OS-Workload-Manager drei Arten von Zielen:

- **Response Time Goals**
- **Execution Velocity Goals**
- **Discretionary**

Jingkai Chen: Modellierung eines Goal orientierten Workload-Managers. Diplomarbeit Universität Tübingen, November 2006, http://www.cedix.de/DiplArb/Chen07.pdf

13.3.2 Arten von Zielen

Response Time Goals

Unter einem Response Time Goal versteht man die Zeitspanne zwischen Start und Fertigstellung einer Work Unit, die in Sekunden pro Programm gemessen wird. Als Zeit wird die vollständige Verweildauer inklusive der Zeit, in der die Work Unit nicht arbeitet, betrachtet. Dadurch kann man die Wartezeit eines Benutzers erkennen.

Execution Velocity Goals

Vor allem für Service Klassen mit Stapelverarbeitung Work Units macht ein Response Time Goal wenig Sinn. Um für derartige Work Units Ziele definieren zu können, kann ein Execution Velocity Goal verwendet werden. Execution Velocity legt den Anteil der akzeptablen Wartezeit bei der Ausführung von Programmen fest. Bei einem Execution Velocity Goal werden nur die Zustände betrachtet, bei denen eine Work Unit Resourcen benutzt (Using) oder darauf wartet (Delay).

Discretionary

Für eine Gruppe von Work Units existieren keine bestimmten Zielvorgaben. In z/OS Systemen werden diese Gruppen von Work Units als Discretionary bezeichnet. Diese Work Units erhalten nur dann Resourcen, wenn diese ausreichend vorhanden sind, und sie sind die Ersten, die keinen Zugang mehr erhalten, wenn es eng im System wird. Auf der anderen Seite tritt ein Workload Management Regelmechanismus in Kraft, der den Resourcen Zugang für Service Klassen mit Zielvorgaben begrenzt, wenn diese ihre Ziele deutlich übererfüllen.

13.3.3 Response Time Goal

Response Time oder Antwortzeit drückt den Wunsch aus, dass die Zeit, die Work Units (z.B. Transaktionen) im System verweilen, maximal einem vorgegebenen Wert entsprechen. Es gibt 2 alternative Möglichkeiten, dieses Ziel zu spezifizieren

Durchschnittliche Antwortzeit (Average Response Time). Beispiel: Avg. Resp. Time = 0,75 s

Prozentsatz Antwortzeit (Percentile Response Time). Beispiel % Resp. Time: 90 % < 0,5 s (90 % aller Transaktionen werden in weniger als 0,5 s beantwortet).

Die Percentile Response Time ist häufig wichtiger als die Average Response Time (wie häufig ist die Antwortzeit zu lang ?).

Die Antwortzeit betrifft nur die Zeit zwischen Eintreffen und Verlassen der Nachricht in Mainframe Rechner. Netzverzögerungen können von WLM nicht erfasst, und deshalb auch nicht berücksichtigt werden (obwohl dies oft erwünscht wäre).

13.3.4 Execution Velocity Goal

Bei einem Execution Velocity Goal werden nur die Zustände betrachtet, bei denen eine Work Unit Resourcen benutzt (using) oder darauf wartet (delay). Ein ausführbarer Prozess (nicht laufend) verbraucht keine Resourcen.

Velocity = theoretisch beste Zeit / wirklich verbrauchte Zeit

Execution Velocity ist wie folgt definiert:

$$\text{Execution Velocity} = \frac{\text{Total Usings}}{\text{Total Usings + Total Delays}} \times 100$$

Näherungsweise ist dies:

$$\text{Velocity} = \frac{\text{Zeit für CPU + I/O}}{\text{Zeit für CPU + I/O + paging + MPL + swapin + queues}}$$

(MPL = Multiprogramming Level)

Nehmen wir z.B. an, dass eine Work Unit insgesamt 5 ms lang auf Resourcen wartet und insgesamt 5 ms lang Resourcen benutzt. Man erhält eine *Execution Velocity* von 50:

$$\text{Execution Velocity} = \frac{\text{5 Usings}}{\text{5 Usings + 5 Delays}} \times 100 = 50$$

Der Wert der Execution Velocity liegt immer zwischen 1 und 100.

13.3.5 Performance Index

Frage: Wie erfüllen Service-Klasse ihre Ziele (Goals) und wie verhalten sie sich im Vergleich zu anderen Service-Klassen.

Workload-Manager-Algorithmen lösen dieses Problem durch die Definition eines Performance Index (PI).

Die Definition des Performance Index (PI)besagt:

PI < 1:	Ziel wurde übererfüllt. Die Service Class benötigt vermutlich nicht alle ihr zugeteilten Resourcen
PI = 1:	Ziel wurde exakt erreicht. Alles ist ok
PI > 1:	Ziel wurde nicht erreicht. Die Service Class benötigt zusätzliche Resourcen

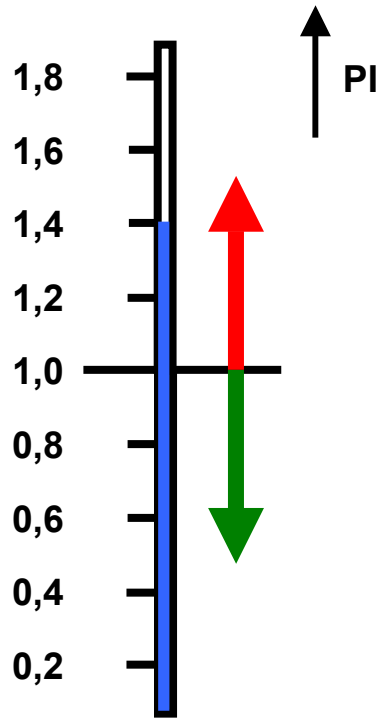


Abb. 13.3.1
Performance Index

13.3.6 Performance Index Definitionen

Die Definition des Performance Index (PI) erfolgt unterschiedlich für die Execution Velocity Goals und die Response Time Goals, um die gezeigte einfache Darstellung zu ermöglichen.

Definition des Performance Index für das Execution Velocity Goal:

$$PI = \frac{\text{Execution Velocity Goal}}{\text{Aktuell erreichte Execution Velocity}}$$

Definition des Performance Index für das Response Time Goal:

$$PI = \frac{\text{Aktuelle Response Time}}{\text{Response Time Goal}}$$

Ein PI < 1bedeutet, Ziele wurden erreicht
Ein PI > 1bedeutet, Ziele wurden nicht erreicht

Ein PI = 1 signalisiert Zielerfüllung.

13.3.7 Importance (Wichtigkeit)

In einem gut ausgelasteten System werden nicht notwendigerweise alle Ziele erreicht !

Neben der Zielvorgabe wird jeder Service Class (Dienstklasse) eine Wichtigkeit (Importance) zugeordnet, die festlegt, welche Klassen bevorzugt bzw. benachteiligt werden sollen, wenn benötigte Resourcen in dem Mainframe Server nicht mehr ausreichend zur Verfügung stehen.

Der wichtigste Punkt bei der Definition einer Service-Klasse ist es festzulegen, wie wichtig sie ist, um die übergeordneten Geschäftsziele zu erreichen. Die Wichtigkeit wird dargestellt, indem eine Goal Importance für die Service-Klasse definiert wird. Diese Goal Importance ist später für das System der wesentliche Entscheidungsfaktor, wenn der Zugang zu den Resourcen geregelt werden muss. Sie wird normalerweise durch eine Ziffer zwischen 1 und 5 angegeben, wobei 1 die höchste, und 5 die niedrigste Wichtigkeit darstellt.

Dies wird an Hand eines Beispiels erläutert. Die gesamte Workload für den Rechner wird in 5 Serviceklassen aufgeteilt. Hierbei werden die Stapelverarbeitung-Work-Units in zwei Gruppen mit unterschiedlichen Zielsetzungen aufgeteilt.

CICS Work Units
TSO Work Units
Stapel 1
Stapel 2
Internet

Abb. 13.3.2 zeigt ein Beispiel für die Classification Rules.

13.3.8 Classification Rules Beispiel

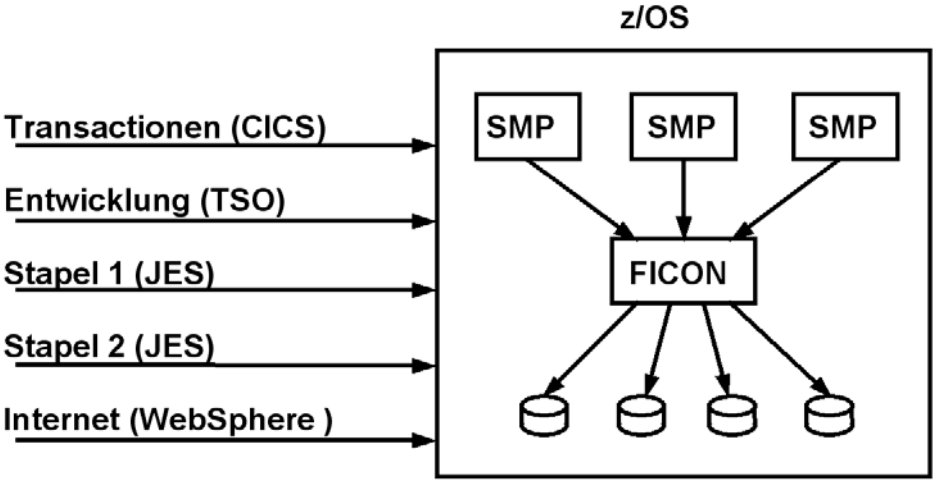


Abb. 13.3.2
Aufteilung der Workload in 5 Service Klassen

	Unterschiedliche Ziele für Service Classes	Wichtigkeit
Transaktionen, (CICS)	90 % Antwortzeit < 0,35 s	1
Stapel 1	90 % complete < 3 Stunden	4
Stapel 2	niedrige Priorität	5
Internet (DB2) *)	90 % Antwortzeit < 0,3 s	3
Entwicklung (TSO)	90 % Antwortzeit < 0,57 s	2

*) DB2 Stored Procedures werden in WLM-managed Adressenräumen ausgeführt.

Workload	Service Class	Per	Dur	Imp	Goal Type	Goal Pct	Goal Value	# Rules	Goal	Comment
BATCH	BATJESME	1		5	ExVel	20		1	Execution velocity of 20	BATCH JES WORKLOAD MEDIUM
BATCH	BATWLMHI	1		4	ExVel	30		2	Execution velocity of 30	BATCH WLM WORKLOAD HIGH
BATCH	BATWLMLO	1		4	Disc			1	Discretionary	BATCH WLM WORKLOAD LOW
BATCH	BATWLMME	1		5	ExVel	20		1	Execution velocity of 20	BATCH WLM WORKLOAD MEDIUM
DATABASE	DB2DDF	1	500	3	RspTime	80	0.5	1	80% complete within 00:00:00.500	DB2 DDF REQUESTS
DATABASE	DB2DDF	2		4	ExVel	10		1	Execution velocity of 10	DB2 DDF REQUESTS
DATABASE	DB2STC	1		1	ExVel	60		24	Execution velocity of 60	DB2 REGION SERVICE CLASS
ONLINE	CICSIMP	1		2	RspTime	90	0.6	5	90% complete within 00:00:00.600	CICS Important Transactions
ONLINE	CICSLOW	1		3	RspTime	80	1	1	80% complete within 00:00:01.000	Other CICS Transactions
STCTASKS	OMVS	1	5000	2	RspTime	80	1	1	80% complete within 00:00:01.000	UNIX System Services
STCTASKS	OMVS	2		3	ExVel	30		1	Execution velocity of 30	UNIX System Services
STCTASKS	STCDFLT	1		4	ExVel	30		1	Execution velocity of 30	Started tasks other/low
STCTASKS	STCHI	1		2	ExVel	50		11	Execution velocity of 50	Started tasks high
STCTASKS	STCMED	1		3	ExVel	30		25	Execution velocity of 30	Started tasks medium
TSO	TSO	1	3000	3	RspTime	90	0.5	1	90% complete within 00:00:00.500	TSO Users
TSO	TSO	2		4	ExVel	20		1	Execution velocity of 20	TSO Users long running

1. The most important work are the DB2 address spaces.
   - The work running within the DB2 subsystem will be managed according to the transactions
2. At importance level 2 we have the high important started task, including the CICS regions, and the important CICS transactions (CICSIMP)
3. Some service classes have a second period defined. The second periods have a much less aggressive goal and a lower importance
4. Batch runs at the lowest importance: 4, 5 and DISCRETIONARY

Abb. 13.3.3
Beispiel einer mit der „WLM Administrative Application“ erstellten Service Definition

13.3.10 WLM Regelmechanismus

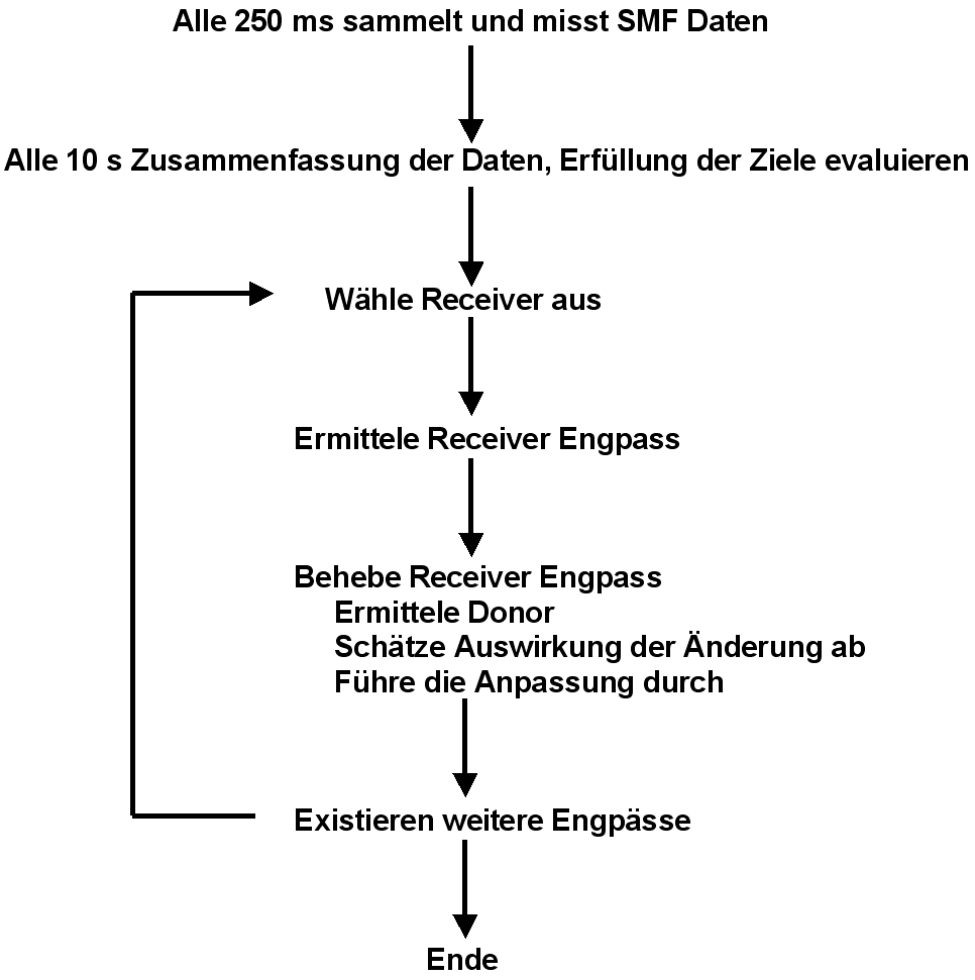


Abb. 13.3.4
Ermittelung von Donor und Receiver

WLM benutzt einen Regelmechanismus, um zur Laufzeit den Zugang zu den Resourcen zu steuern. Dazu werden kontinuierlich Daten aus dem z/OS-System gesammelt (mit Hilfe der System Management Facility, SMF). Dies sind Informationen über die Wartezustände der Work Units auf die Resourcen, die Anzahl der laufenden Work Units und deren Abarbeitungszeiten. Die Informationen werden für die einzelnen Service Classes (Dienstklassen) ermittelt und zusammengefasst, entsprechend der Klassifizierung, die durch den Systemverwalter vorgenommen wurde. Dann wird auf der Basis dieser Informationen die Zielerfüllung für jede Klasse berechnet und, falls notwendig, der Zugang zu den Resourcen angepasst.

Die Anpassung erfolgt immer in Abhängigkeit von der Wichtigkeit (Importance) der Klassen, und dem Grad, in dem das Ziel verfehlt wird. Das heißt, die wichtigste Klasse, die am weitesten ihr vorgegebenes Ziel verfehlt hat, wird als erste betrachtet und die Klassen mit der geringsten Wichtigkeit sind die potenziellen Kandidaten, um Resourcen abzugeben. Die wichtigere Klasse wird der Empfänger (Receiver) zusätzlicher System Resourcen. Diese müssen natürlich einer anderen Dienstklasse weggenommen werden. Dabei wird allerdings berücksichtigt, ob ein potenzieller Spender (Donor) auch tatsächlich das benötigte Resourcen verwendet.

Hieraus werden vom Goal Oriented Work Load Manager Einstellparameter für den System Resources Manager (SRM) abgeleitet. Der SRM nimmt dann die entsprechenden Anpassungen vor.

Dieser Regelmechanismus läuft typischerweise alle 10 Sekunden im z/OS ab. In der Zwischenzeit werden die Daten für das nächste Berechnungsintervall von SRM gesammelt. Ein Berechnungsintervall endet, wenn eine Anpassung zugunsten einer Dienstklasse durchgeführt werden kann.

Der in jedem Zeitintervall ablaufende Regelmechanismus besteht aus den folgenden Schritten:

- **Auslastung messen**
- **Verhalten mit den Zielen (Goals) vergleichen**
- **Anpassungen vornehmen (Donor und Receiver ermitteln.**

13.3.11 Simulationsmodell

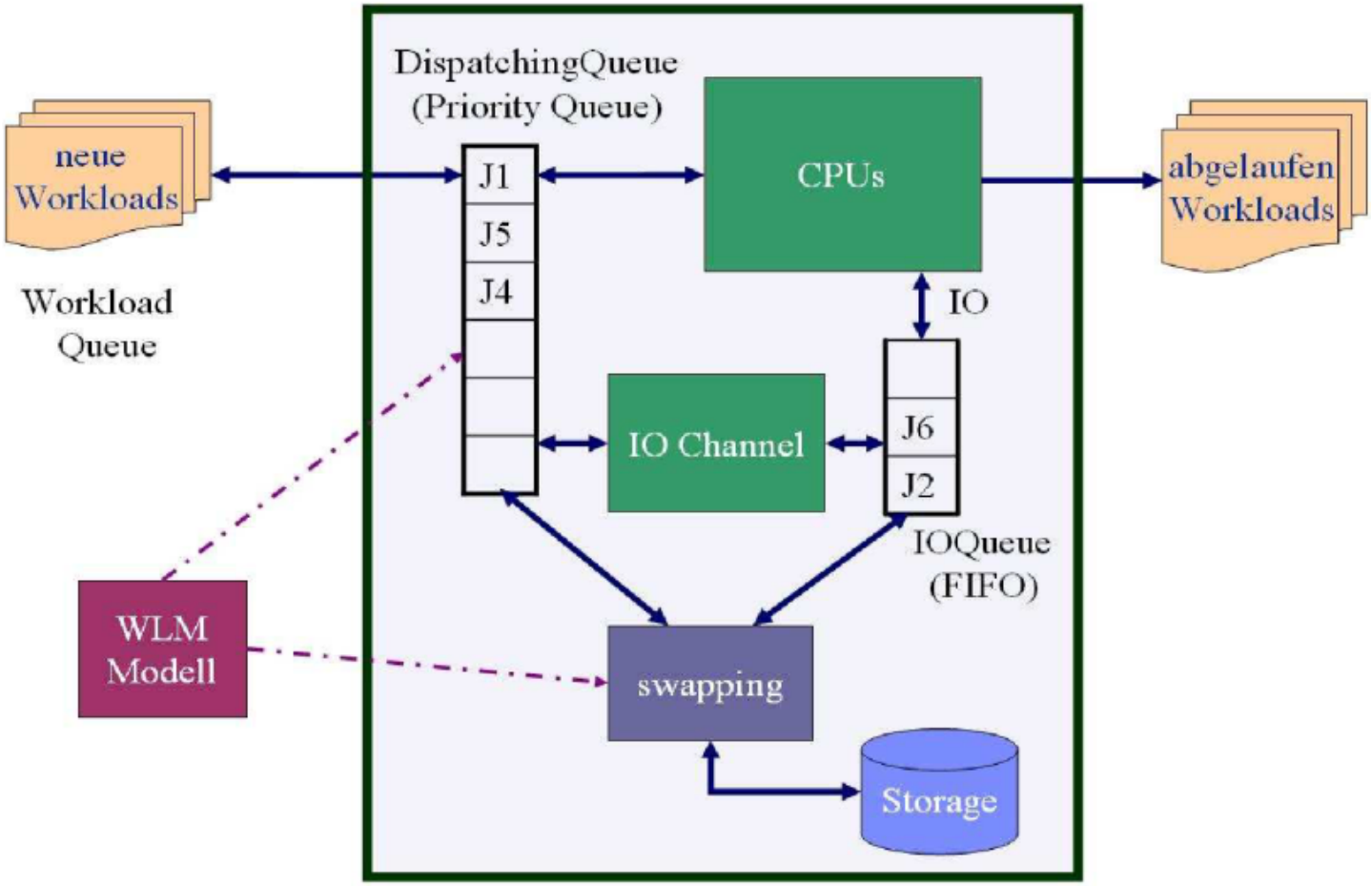


Abb. 13.3.5
Vereinfachte Konfiguration für das Simulationsmodell

Wir beobachten diese Vorgänge an Hand eines einfachen Simulationsmodells, welcher in der folgenden Abbildung dargestellt ist.

Das simulierte System besteht aus mehren CPUs, welche gleichzeitig ausführbare Prozesse bearbeiten. Die wartenden Prozesse befinden sich in der I/O Queue. Nach Ende der I/O Verarbeitung werden die jetzt ausführbaren Prozesse der Dispatching Queue übergeben.

Der Zugang zu den CPUs wird mittels der Dispatching Queue gesteuert. Jeder Prozess, d.h. jede im Hauptspeicher befindliche Work Unit, erhält eine Dispatch-Priorität und ist innerhalb dieser Dispatch-Queue nach seiner Priorität geordnet. Der Prozess Scheduler ordnet immer dem ersten Prozess in der Dispatch-Queue mit der höchsten Priorität eine verfügbaren CPU zu.

Unser Modell enthält weiterhin einen Ein-/Ausgabe Kanal (I/O-Channel). In z/OS ist dieser Betriebsmittelzugang durch die Festlegung von Prioritäten für den Ein-/Ausgabe-Kanal geregelt. Um das in der Simulation verwendete Szenario einfach und anschaulich zu halten, wurde statt dessen eine FIFO-Queue (First In First Out) als Ein/Ausgabe-Queue (I/O-Queue) verwendet. Hierbei werden alle Prozesse in der Reihenfolge ihres Eingangs bearbeitet.

Für die Speicherverwaltung ist ein Swapping modelliert. Wir nehmen an, dass sich der Prozess, der in der Dispatching-Queue oder Ein/Ausgabe-Queue liegt, im Hauptspeicher befindet. Wenn die Workload ungesteuert abläuft, kann es zu Engpässen bei den Betriebsmitteln kommen. Um das zu vermeiden, wird für jede Service-Klasse festgelegt, wie viele Adressräume gleichzeitig im System sein dürfen und wie viele auf jeden Fall im System bleiben sollen. Die Zahl der Adressräume, die gleichzeitige im System sind, wird als Multiprogramming Level (MPL) bezeichnet. Der MPL ist Teil eines wesentlichen Steuerungsmechanismus des Systems. Die Unter- und Obergrenze für den MPL darf nur durch den WLM dynamisch geändert werden. Wenn das System feststellt, dass Engpässe auftreten, kann der Workload-Manager die MPL der Service-Klasse reduzieren, damit Adressräume auslagert werden und die verbleibenden Adressräume effizienter (weniger Fehlseiten Unterbrechungen) arbeiten können.

13.3.12 Service Definition für das Experiment

Für das Simulationsmodell werden insgesamt 5 Service-Klassen modelliert:

Name	Importance	Goal
scCICS1	1 (hoch)	0,35 s Response
scTSO1	2	0,57 s Response
scDB21	3	0,3 s Response
scJES1	4	15 Velocity
scJES2	5 (niedrig)	15 Velocity

In der folgenden Graphik ist die Verteilung der Work Units in den einzelnen Service-Klassen für die Zeit von 13:00:00 bis 13:05:00 wiedergegeben. Während der ganzen Zeit liegt die Anzahl Work Units in jeder Klasse unter 20. Mit einer Ausnahme:

Um 13:01:00 wird eine große Anzahl von Work Units in der Service Klasse ‚scDB21' gestartet. Der Wert steigt plötzlich von unter 20 auf über 100. Un 13:02:00 fällt der Wert wieder auf unter 20 zurück. Dies ist in Abb. 3.3.6 dargestellt.

Wir sehen uns an, wie WLM den PI (Performance Index) managed.

13.3.13 Belastung durch die Service Klassen

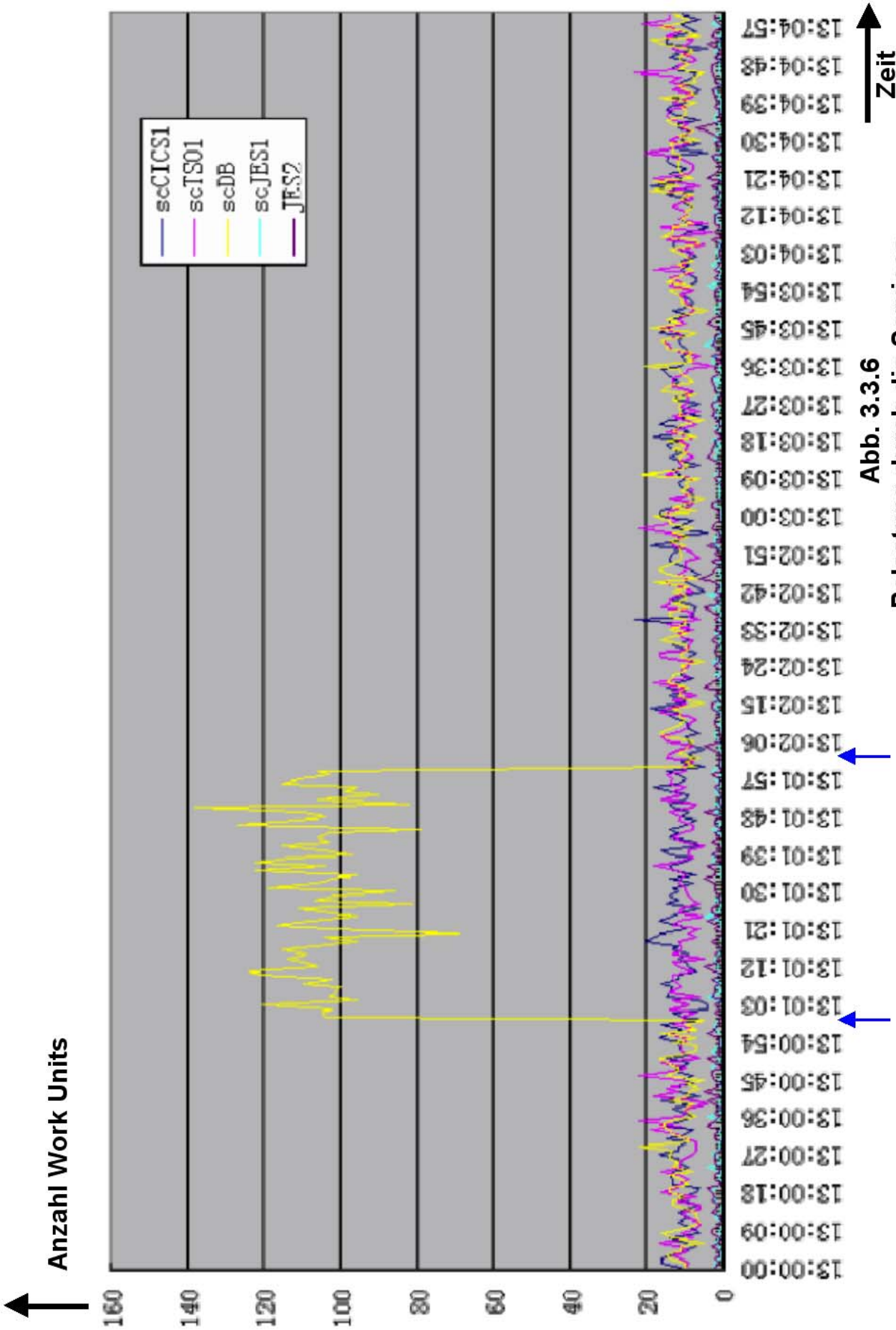


Abb. 3.3.6
Belastung durch die Services

13.3.14 Reaktion des Work Load Managers

In Abb. 3.3.7 ist der zeitliche Verlauf der PIs (Performance Indizes) für die 5 Service Klassen scCICS1, scTSO1, scDB21, scJES1 und scJES2 dargestellt. WLM arbeitet typischerweise mit 10 Sekunden Intervallen. In diesem Beispiel sind es 9 Sekunden.

Da nur wenige Work Units bis zu 13:01:00 in das System eintreten, und die CPU-Anforderungen gering sind, werde die Ziele aller Service-Klassen übererfüllt (PI <1). Die Ziele für scJES1 und scJES2 sind nicht so eng gefasst, deshalb ist es auch möglich, alle Ziele deutlich über zu erfüllen. Da seit 13:01:02 die Anforderungen durch die zusätzlichen scDB21-Work Units sehr hoch sind, werden zu diesem Zeitpunkt die Ziele für scDB21 nicht mehr erreicht (PI >1). WLM beginnt jetzt zu steuern.

<u>Um 13:01:02</u> war die Service Klasse scDB21 der Receiver zusätzlicher Resourcen, die von dem Donor scJES2, der Service Class mit der niedrigsten Importance, abgegeben wurden. Wir sehen, dass etwa 10 Sekunden später (um 13:01:10) der PI für scDB2 wieder unterhalb von 1 ist und der PI für scJES2 deutlich oberhalb von 1 steigt, da die MPL-Werte der scJES2 Service Klasse reduziert werden.

<u>Um 13:01:10</u> hatte die Performance der Service Klasse scJES1 einen PI deutlich kleiner als 1, während der PI Wert von scJES2 deutlich über 1 liegt. Obwohl die Importance von scJES1 höher als die von scJES2 ist, wird scJES1 zum Donor und gibt Resourcen an scJES2 (Receiver) ab. Danach ist bis um 13:01:40 der PI für scJES1 gestiegen und der PI für scJES2 besser geworden, obwohl die Ziele von scJES2 immer noch nicht erfüllt werden (der PI ist > 1).

<u>Um 13:01:41</u> gibt es zwei Service-Klasse scJES1 und scJES2 mit einen PI größer als 1. Der PI von scDB2 liegt unter 1. Service Klasse scDB2 gibt deshalb Resourcen an Service Klasse scJES1 ab. Das Ergebnis ist wenig erfreulich, denn

<u>um 13:01:48</u> haben die Service-Klassen scDB21, scJES1 und scJES2 alle ihre Ziele nicht erreicht. Die Service Klasse scDB2 wird deshalb ein Receiver zusätzlicher Resourcen und die Service Klasse scTSO1 wird s der Donor. Die PI von scTSO1, steigt, bleibt aber unter dem Wert 1.

<u>Ab 13:02:00</u> fällt die Anzahl von Work Units in der Service Klasse scDB21 wieder auf den ursprünglichen Wert < 20. Die PI Werte von scDB2 und scJES1 sind deshalb besser geworden. Da die PI der scJES2 immer noch größer als 1 ist, führt WLM

<u>ab 13:02:04</u> bis 13:04:00 führt WLM weitere Anpassungen durch. Dies verursacht wilde Oszillationen zwischen dem PI von scJES1 und dem PI von scJES2. Um 13:05:00 endet das Test.

Der Work Load Manager des Simulationsmodells verwendet einen sehr einfachen Regelalgorithmus. Offensichtlich sind Verbesserungen im Regelalgorithmus notwendig, z.B. um die Oszillationen zwischen <u>13:02:04</u> und <u>13:04:00</u> zu beseitigen. Der Algorithmus des z/OS Work Load Managers ist dagegen ein sehr komplexes Gebilde, in das vieljährige Erfahrungen eingeflossen sind. Nach allgemeiner Erfahrung liefert WLM deshalb auch in komplexen Installationen sehr befriedigende Ergebnisse.

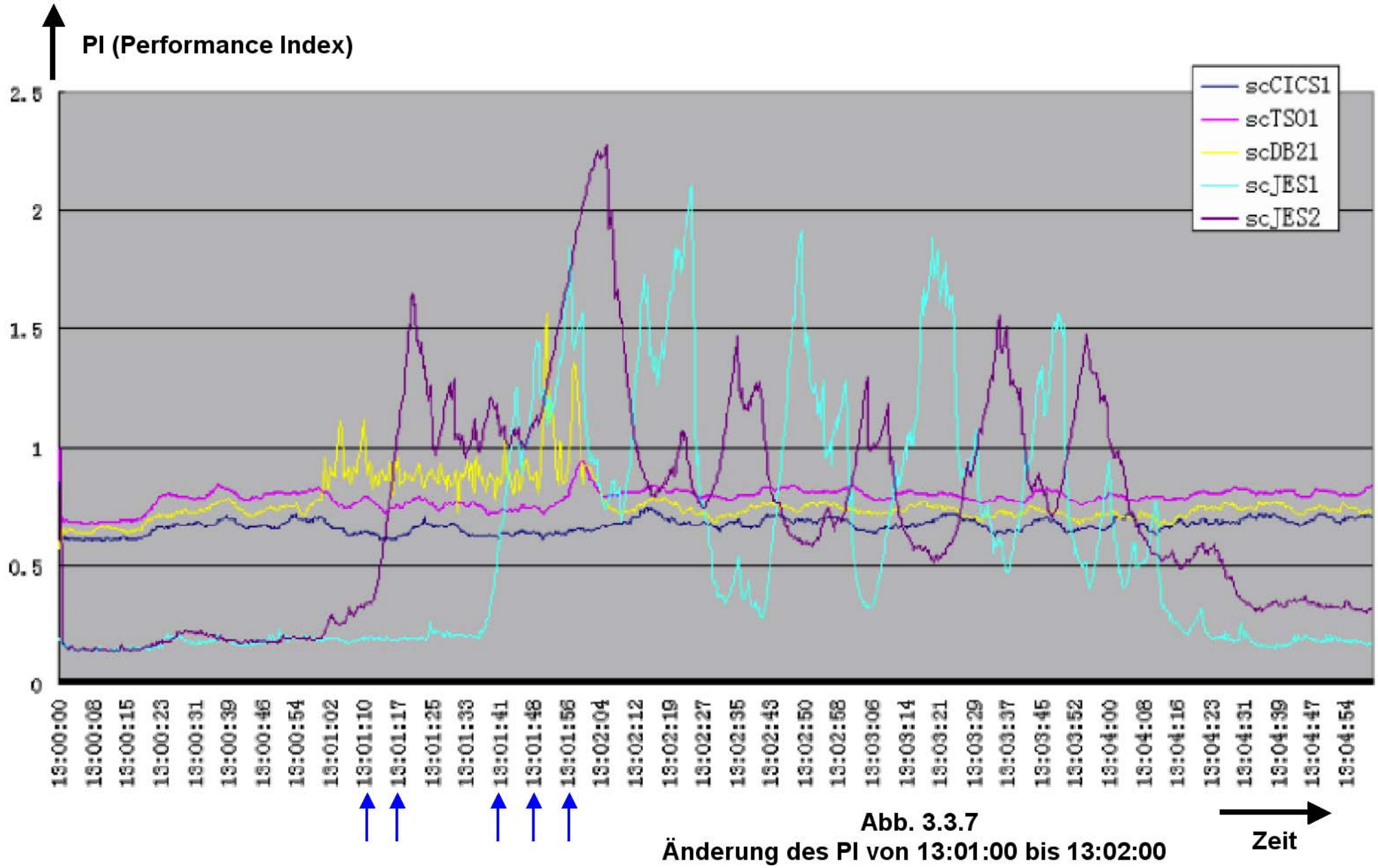


Abb. 3.3.7
Änderung des PI von 13:01:00 bis 13:02:00

13.3.15 WLM und Sysplex

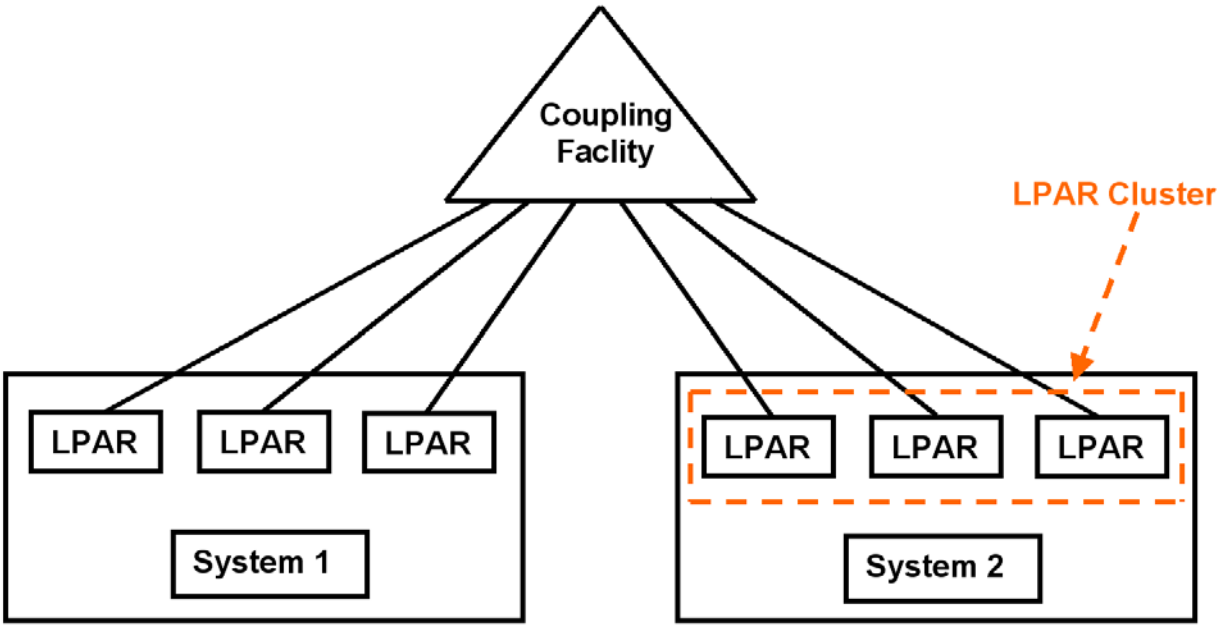


Abb. 13.3.8
Die LPARs eines LPAR Clusters können von WLM gemeinsam gesteuert werden

Eine große Sysplex Installation besteht in der Regel aus mehreren physischen Rechnern (Systeme in der IBM Terminologie), von denen jeder mehrere LPARs mit je einem SMP (z/OS Instanz) pro LPAR enthält. Jede z/OS Instanz in jeder LPAR verfügt über ihren eigenen WLM, und jede z/OS Instanz ist über ein Coupling Link mit der Coupling Facility verbunden.

Alle LPARs auf dem gleichen System werden als LPAR Cluster bezeichnet, und können von WLM unter Benutzung der IRD Funktionen gemanaged werden

Die WLM Instanzen auf unterschiedlichen Systemen tauschen Information über die Coupling Facility aus. Alle LPARs haben die gleiche Service Klassen Definitionen.

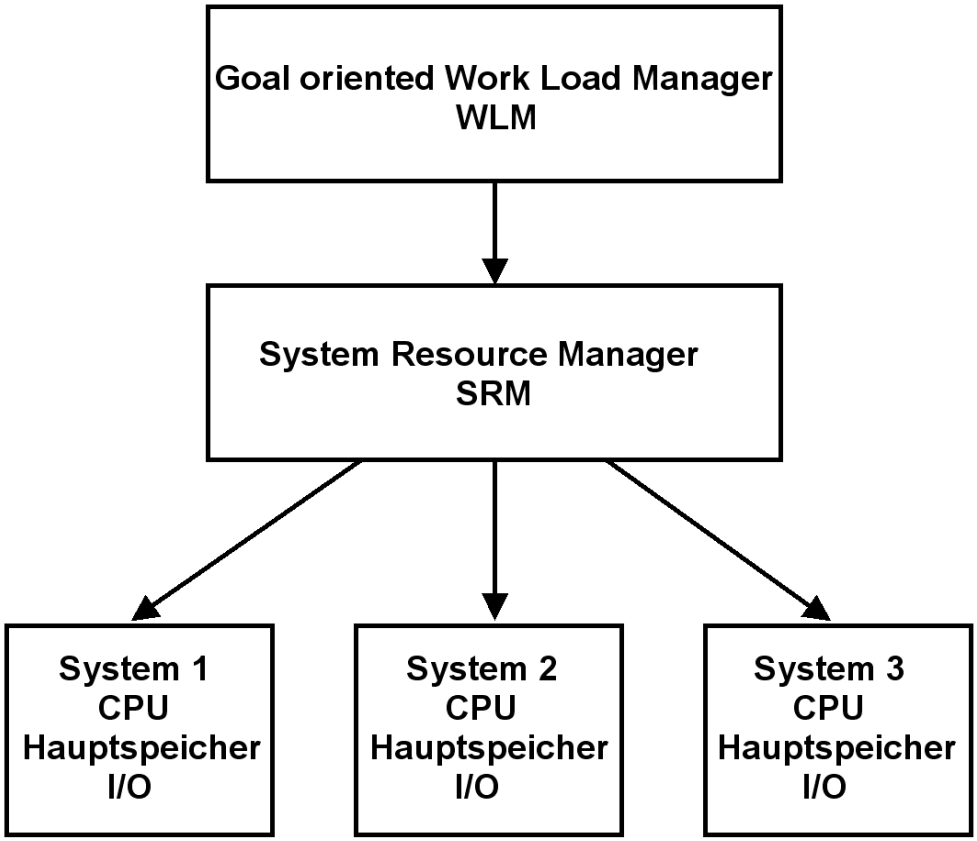


Abb. 13.3.9
Die Systeme (Knoten) eines Sysplex Clusters werden von WLM gemeinsam gesteuert

Der z/OS Work Load Manager kann nicht nur einen einzelnen Rechner, sondern auch einen ganzen Sysplex steuern.

Angenommen ein Sysplex mit drei Systemen.

Die System Resource Manager Komponente des Work Load Managers beobachtet für alle angeschlossenen Systeme:

- **CPU Auslastung**
- **Hauptspeicher Nutzung**
- **I/O Belastung**

Der Goal oriented Work Load Manager steuert die optimale Auslastung der Systeme des Sysplex.

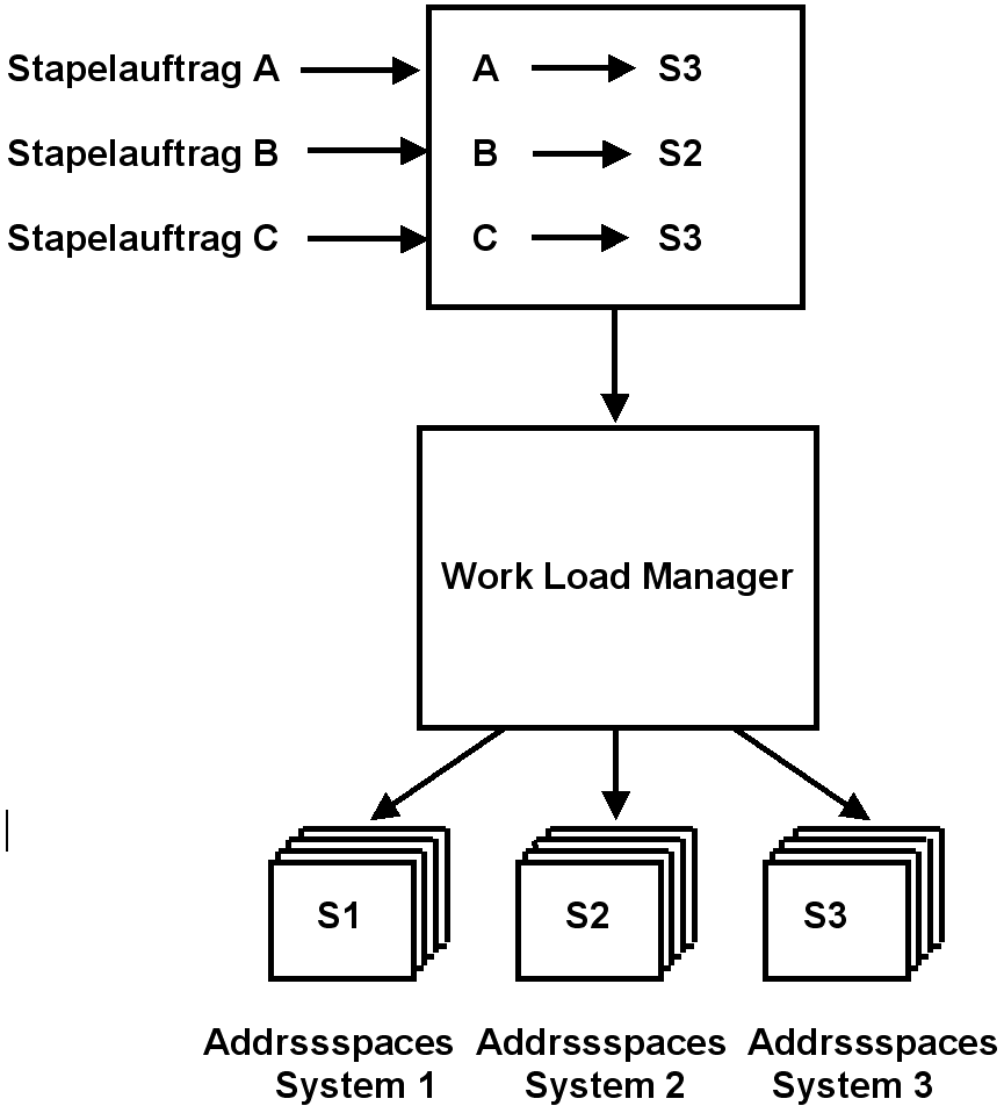


Abb. 13.3.10
Zuordnung von Aufträgen zu Systemen eines Sysplex

Eine Aufgabe des Work Load Managers ist es, Einstellungen der einzelnen Systeme eines Sysplex dynamisch an sich ändernde Belastungen anpassen und automatisch justieren.

Bei widersprüchlichen Anforderungen (Regelfall) verfügt der WLM über Algorithmen, einen möglichst optimalen Kompromiss zu erreichen.

Gezeigt ist, wie die gerade neu eintreffenden Jobs A, B und C auf die Systeme S1, S2 und S3 verteilt werden. Der Work Load Manager entscheidet (auf Grund seiner Einschätzung der derzeitigen Auslastung) Job B an System S2 sowie Jobs A und C an System S3 zu vergeben, während System S1 keine zusätzliche Belastung bekommt.

13.3.16 WLM-managed JES Initiators

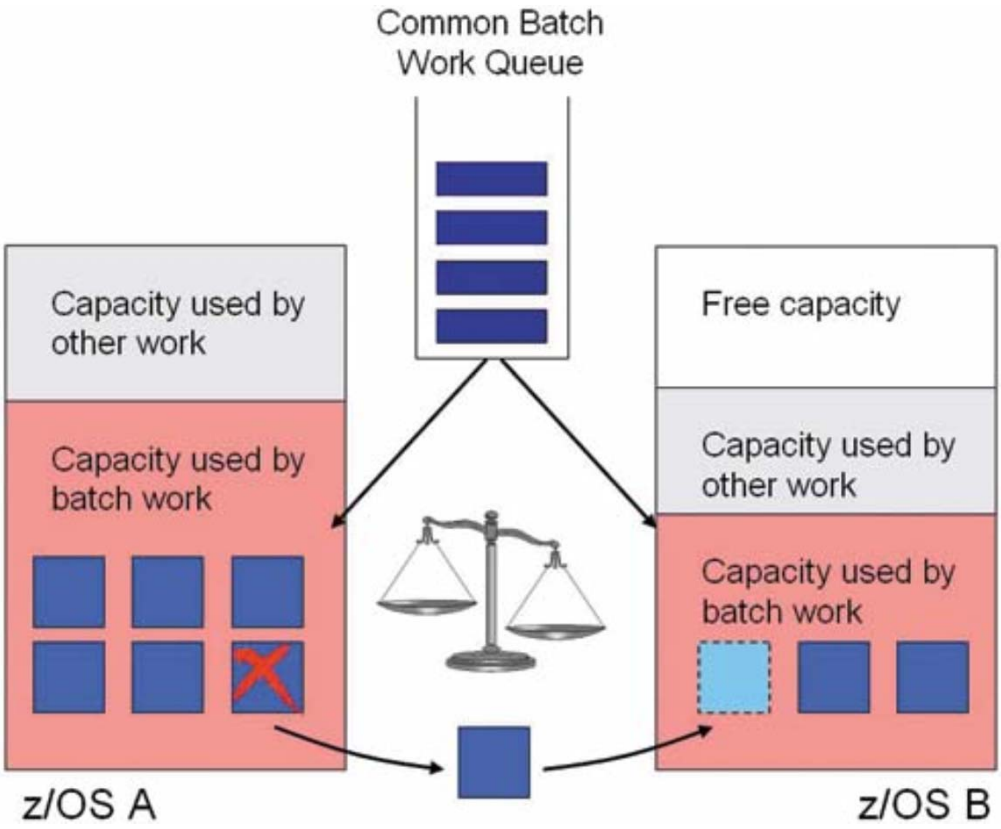


Abb. 13.3.11

Band 1, Abschnitt 3.2.4 erläutert die Funktion der Initiators eines Job Entry Subsystems (JES).

Für JES entscheidet der WLM (Workload Manager) , wann wie viele Initiators auf welchen Systemen des Parallel Sysplex-Verbundes gestartet und gestoppt werden. In dem Beispiel wird ein Initiator in System A gestoppt und ein neuer Initiator in System B gestartet.

Der WLM stützt sich dabei auf die Vorgaben der Performance Goals in der WLM-Policy.

13.4 Weiterführende Information

13.4.1 Die wichtigste Literatur

Eine sehr gute Übersicht über den Work Load Manager ist zu finden in Kapitel 6 des Lehrbuches

M. Teuffel, R. Vaupel: „Das Betriebssystem z/OS und die zSeries“. Oldenbourg 2004., ISBN 3-486-27528-3.

Die wichtigsten Redbooks sind

z/OS Intelligent Resource Director, August 2001, SG24-5952-00,
http://www.redbooks.ibm.com/abstracts/sg245952.html

System Programmer’s Guide to: Workload Manager, January 2006, SG24-6472-02
http://www-ti.informatik.uni-tuebingen.de/~spruth/edumirror/xx044.pdf

http://www.redbooks.ibm.com/redbooks/pdfs/sg246472.pdf

http://h18004.www1.hp.com/products/quickspecs/11726_div/11726_div.HTML

13.4.2 Videos

Ein Youtube Video über die neue Administrator Interface zu dem z/OS finden Sie unter
http://www.youtube.com/watch?v=8AaDxordGJM

http://www.mdug.org/Presentations/Ed%20Woods%20-%20DB2%20&%20WLM.pdf
enthält einen leicht verständlichen Übersichtsvortrag zum Thema WLM.

http://www.slideshare.net/Tess98/managing-your-db2-for-zos-workloads-using-wlm

13.4.3 Weitere Literatur

WLM ist ein komplexes Thema. Weitere Weiterführende Literatur finden Sie unter:

- **z/OS V1.6 Workload Management, Server Limits für DB2 Application Environments:** http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IEA2W150/13.3?DT=20040709143734
- **DB2 for z/OS Stored Procedures: Through the CALL and Beyond:** http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg247083.html
- **USING WORKLOAD (WLM) and DB2 STORED PROCEDURES (SPAS) - Some clarifying Questions and Answers:** http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD103105
- **DB2 stored procedure performance considerations, part two:** http://www-306.ibm.com/software/tivoli/features/ccr2/ccr2-2004-04/features-db2stored.html
- **Setting up the WLM application environment for interpreted Java routines:** http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2.doc.java/bjnkmstr128.htm
- **Workload management (WLM) tuning tips for z/OS Questions and Answers:** http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1//index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rprf_tunezwlm.html

- **z/OS V1.6 Workload Management, Server Limits für DB2 Application Environments:** http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IEA2W150/13.3?DT=20040709143734
- **DB2 for z/OS Stored Procedures: Through the CALL and Beyond:** http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg247083.html

- **Workload Manager** http://de.wikipedia.org/wiki/Workload_Manager
- **ABCs of z/OS System Programming Volume 12** http://www.redbooks.ibm.com/abstracts/sg247621.html
- **M. Bensch, D. Brugger, P. Baeuerle, W. Rosenstiel, M. Bogdan, W. Spruth: "Self-Learning Prediciton System for Optimisation of Workload Managememt in a Mainframe Operating System"**
  International Conference on Enterprise Information Systems, pp. 212-218, Funchal, Portugal, 12.-26. Juni 2007
  http://www-ti.informatik.uni-tuebingen.de/~spruth/Mirror/iceis.pdf

13.4.4 Tivoli

Unter dem Oberbegriff Tivoli bietet IBM eine umfangreich Software Suite zur Verwaltung von Informationssystemen an. Sie dienen dazu, Rechner zu überwachen, Software zu verteilen, Systeme zu inventarisieren oder Daten zu sichern. Die Tivoli Software deckt die folgenden Bereiche ab:

- **Operational Management**
- **Service Management**
- **Process Management**

13.4.4.1 Operational Management

Die im Bereich Operational Management zusammengefassten Produkte stellen die Verfügbarkeit der Systeme sicher. Es wird eine Verwaltung, Überwachung und Optimierung der gesamten IT-Landschaft ermöglicht. Genauer werden die Produkte in vier Kategorien unterschieden:

Business Automation

- **System Automation**
- **Provisioning Manager**
- **Intelligent Orchestrator**
- **Monitoring**
- **Workload Scheduling**
- **Business Systems Manager**
- **Service Level Advisor**

Storage Management

- **Tivoli Storage Manager**
- **Total Storage Productivity Center**
- **SAN Volume Controller**

Security

- **Identity Manager**
- **Access Manager for eBusiness**
- **Access Manager for Enterprise Single Sign-On**
- **Security Operations Manager**
- **Security Compliance Manager**
- **Compliance Insight Manager**

13.4.4.2 Service Management

Die IBM Service Management Platform ermöglicht es, automatisch alle über ein Netzwerk erreichbaren IT-Assets (i.a. Geräte der Informationstechnik) eines Unternehmens aufzufinden, zu inventarisieren und zu katalogisieren. Über einen festzulegenden IP-Adress-Bereich wird die Konfiguration eines jeden Konfiguration Items aufgedeckt und gespeichert. In der CCMDB werden diese Informationen abgelegt. Das Change Management ist mit eingeschlossen. Die Produkte sind

- **Tivoli Application Dependency Discovery Manager (ITADDM) und**
- **Tivoli Change and Configuration Management Database (CCMDB).**

13.4.4.3 IT Process Management

Das IT Process Management umfasst vordefinierte Prozesse die auf der Plattform des Websphere Process Managers zur Anwendung gebracht werden können. Basis dieser Prozesse ist die IT Infrastructure Library (ITIL). Die darin etablierten Workflows werden praktisch umgesetzt, um Unternehmen bei der Einführung von ITIL Practices zu unterstützen. Angebotene Process Manager sind

- **Storage Process Manager,**
- **Release Process Manager und**
- **der Unified Process Composer, um weitere Prozesse entwerfen zu können.**

http://de.wikipedia.org/wiki/Tivoli_%28IBM%29

13.4.4.4 Tivoli für z/OS

Die Tivoli Software Suite deckt das gesamte IT Spektrum ab. Nur ein Teil der Produkte läuft unter z/OS oder hat einen unmittelbaren Bezug zu z/OS. Dies gilt besonders für:

- **Tivoli System Automation für z/OS ist eine Policy-basierte, selbstheilende und hochverfügbare Lösung, um die Effizienz und Verfügbarkeit von kritischen Systemen und Anwendungen zu maximieren.**

- **Tivoli NetView für z/OS bietet Automatisierung, Netzwerk- und Systemmanagement.**

- **Tivoli Event Pump für z/OS ermöglicht die automatische Erfassung und Weiterleitung von Zustands-und Statusereignissen von z/OS-Rechnern und Subsystemen, einschließlich CICS, IMS, DB2 sowie Drittanbieterprodukten.**

- **Tivoli Advanced Backup und Recovery für z/OS stellt eine zuverlässige und genaue Sicherung und Wiederherstellung von Anwendungsdaten auf z/OS-Rechnern bereit, und verfolgt Sicherungen für Prüfprotokolle.**

- **Tivoli OMEGAMON XE für z/OS bietet detailliertes Überwachungs- und Problem-Management für IBM System z und IBM zEnterprise-Systeme. Hiermit kann die Sichtbarkeit, Benutzerfreundlichkeit und Leistung verbessert werden.**

- **Tivoli System Automation für z/OS ist eine Policy-basierte, selbstheilende und hochverfügbare Lösung, um die Effizienz und Verfügbarkeit von kritischen Systemen und Anwendungen zu maximieren.**

- **Mit Tivoli Workload Scheduler für z/OS können Sie die Verarbeitung von IBM System z Workloads automatisieren, planen und steuern. Es fungiert als eine virtueller Kontrollpunkt, der für unternehmensweite Produktion-Workloads die Geschwindigkeit zu maximieren und Ressourcen zu optimieren ermöglicht.**

Jedes dieser Produkte braucht Experten-Wissen für den Einsatz und eine umfangreiche Einarbeitungszeit.

Während IBM Für Software Produkte wie CICS, VSAM und MQSeries ein de Facto Monopol besitzt, gilt das nicht für die Tivoli Produkte (und andere Software wie z.B. RACF). Eine Reihe von anderen Software Unternehmen (Independent Software Vendor, ISV) wie Computer Accociates, BMC und Hewlett Packard bieten viele vergleichbare Software Produkte an, die als Alternative zu den Tivoli Produkten eingesetzt werden können.

14. Hybrid Computing

14.1 Installationsbeispiele

14.1.1 Geographically Dispersed Parallel Sysplex

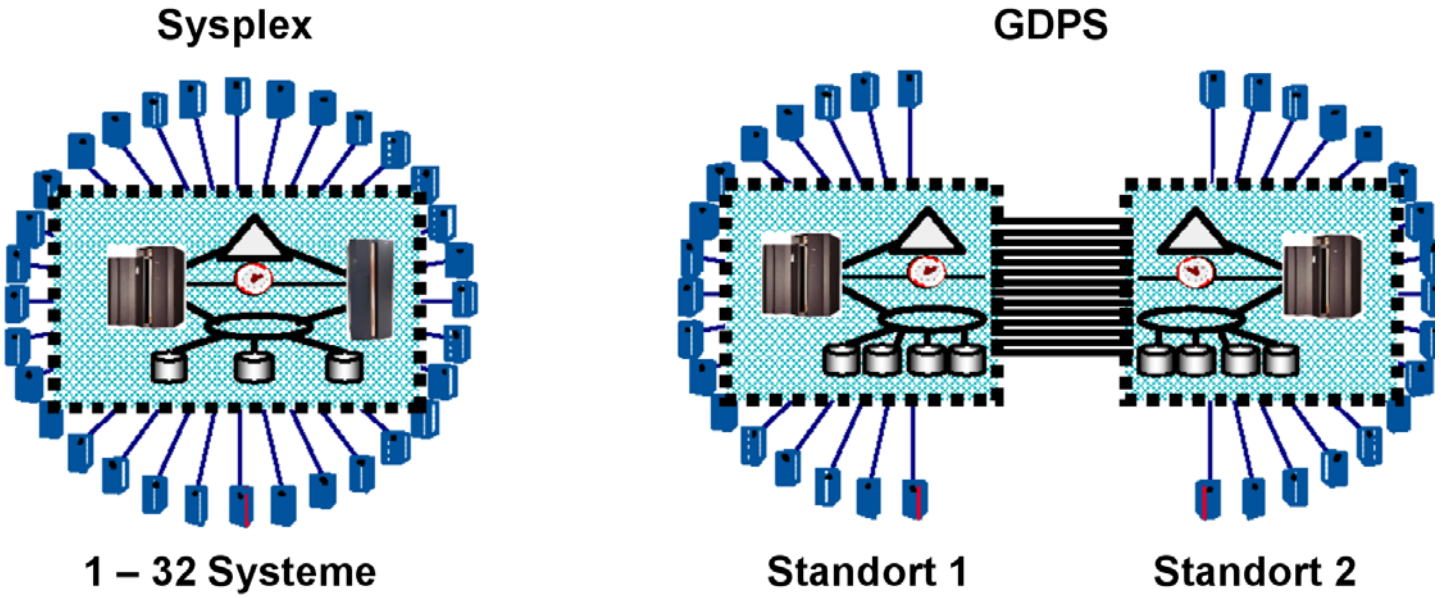


Abb. 14.1.1
Aufteilung des Rechenzentrums auf zwei Standorte

Ein (Parallel) Sysplex ist eine Konfiguration, bei der mehrere Mainframe Rechner zu einem Cluster (closely coupled) zusammengeschlossen sind. Dabei ist angenommen, dass die Rechner relativ nahe (Dutzende von Metern) zueinander untergebracht sind.

Ein „Geographically Dispersed Parallel Sysplex“ (GDPS) ist eine Konfiguration bei der die Rechner des Sysplex in zwei Rechenzentren untergebracht sind, die typischerweise 5 – 10 km voneinander entfernt sind.

14.1.2 Mainframe Installation

Abb. 14.1.2 zeigt eine typische Großrechner Konfiguration.

Die Konfiguration besteht aus zwei Rechenzentren (RZ 1 und RZ 2), die je einen (oder mehrere) Mainframe Rechner enthalten, und in einem Abstand von wenigen km untergebracht sind. Wenn ein Rechenzentrum ausfällt (Beispiel: ein Flugzeug stürzt auf dem Gebäude ab und setzt es in Brand), kann der Betrieb des Unternehmens mit dem 2. Rechenzentrum aufrecht erhalten bleiben. Alle größeren Unternehmen unterhalten aus Gründen der Ausfallsicherheit zwei Rechenzentren.

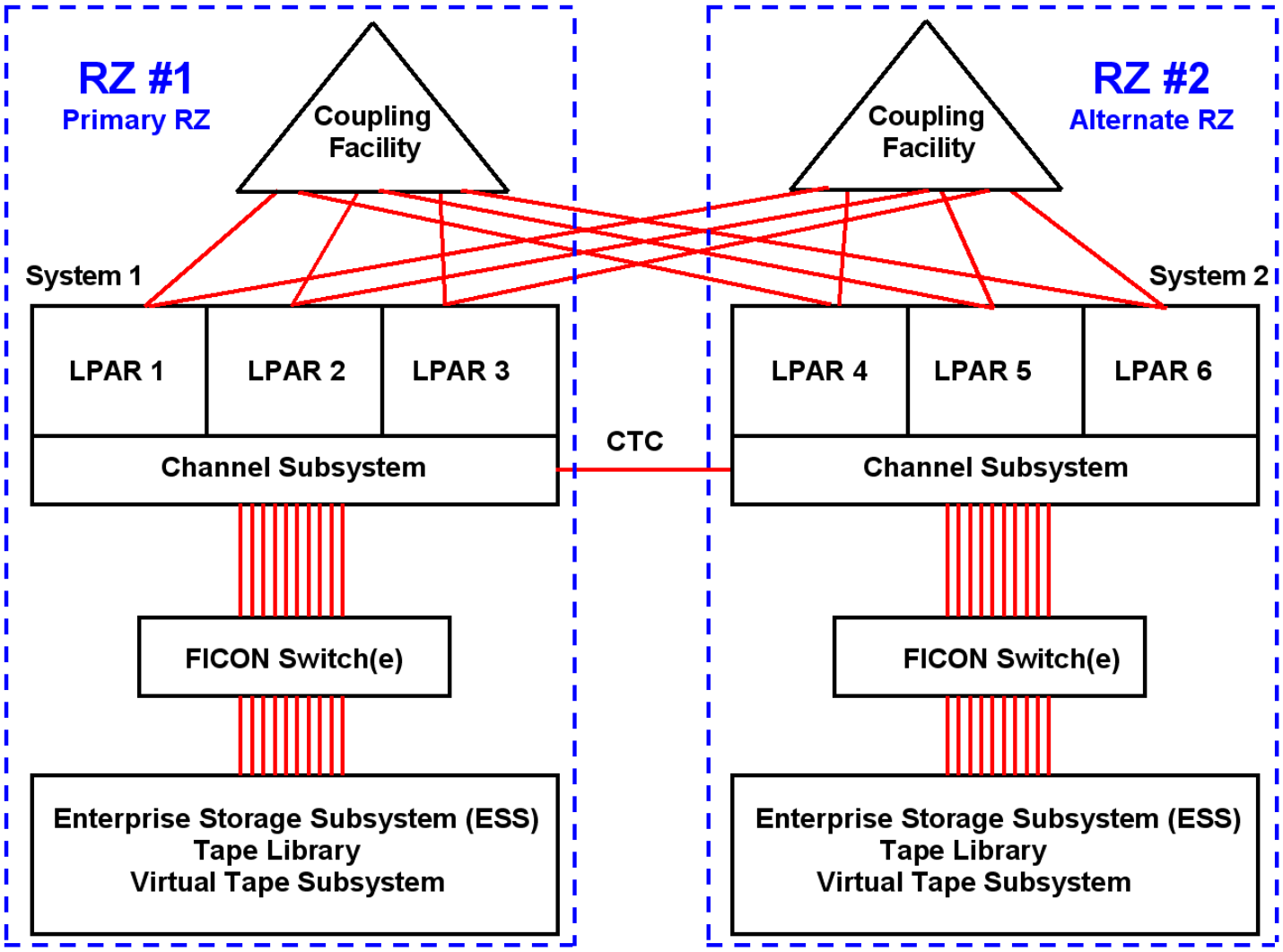


Abb. 14.1.2
Mehrfache LPARs in jedem Standort

Die Entfernung der beiden Rechenzentren untereinander beträgt typischerweise 5 – 10 km, in Ausnahmefällen 15 km. Man wünscht für den Katastrophenfall genügend Abstand. Auf der anderen Seite bewirkt die Lichtgeschwindigkeit (etwa 20 µs für 5 km) eine unerwünschte Latenz für die Datenübertragung zwischen den beiden Rechenzentren. Abstände > 10 – 15 km erfordern eine andere Struktur als kleinere Abstände.

Ebenfalls aus Gründen der Ausfallsicherheit sind (mindestens) zwei Coupling Facilities vorhanden (je eine pro Rechenzentrum). Die Mainframes in den beiden Rechenzentren sind mit beiden Coupling Facilities verbunden.

Weiterhin sind die Mainframes in den beiden Rechenzentren über „Channel to Channel“ (CTC) Glasfaserkabel miteinander verbunden (Band 1, Abschnitt 5.3.6). In ganz großen Installationen befinden sich mehrere Mainframes in jedem der beiden Rechenzentren, z.B. 2 x 3 z196 Rechner bei Volkswagen und der UBS (Union Bank of Switzerland). In mittelgroßen Unternehmen existiert in jedem Rechenzentrum nur ein Mainframe Rechner mit weniger als maximaler Ausrüstung, z.B. mit nur 16 statt maximal 101 CPUs.

14.1.3 Datensicherheit

Alle kritischen Daten sind in beiden Rechenzentren auf getrennten Enterprise Storage Subsystemen und Tape Libraries dupliziert. Zahlreiche FICON Glasfaserverbindungen und mehrere FICON Switche (Director) stellen die Verbindung zu den Mainframes her. Modern ist der zusätzliche Einsatz eines „Virtual Tape Servers“ (VTS, Band 1 Abschnitt 5.4.6). Er enthält einen Plattenspeicher, der eine Art Cache für die Tape Library implementiert.

Typischerweise werden z.B. bei der Transaktionsverarbeitung die Datenbestände in beiden Rechenzentren synchron dupliziert. Eine Update Funktion erfolgt in beiden Rechenzentren (fast) gleichzeitig. Ein Commit erfolgt erst dann, wenn die Datenänderungen in beiden Rechenzentren erfolgreich abgeschlossen sind. Wegen der Latency der Signalübertragung benötigt dieses als „Metro Mirror“ bezeichnete Verfahren etwas mehr Verarbeitungszeit.

Eine als „Global Mirror“ bezeichnete asynchrone Alternative kann bei größeren Distanzen zwischen den beiden Rechenzentren eingesetzt werden, ist dafür aber nicht ganz so Katastrophensicher.

Eine Sysplex Konfiguration umfasst in jedem RZ mindestens eine LPAR, oft aber mehr. Es ist möglich und üblich, dass die LPARs in den beiden RZ als mehrere unabhängige und parallel laufende Sysplexe konfiguriert werden.

14.1.4 Mainframe Installation der Union Bank of Switzerland

Die Union Bank of Switzerland (UBS) ist die größte Bank in der Schweiz mit Hauptsitz in Zürich.

Die beiden Rechenzentren der UBS sind in 10 km Entfernung voneinander untergebracht. In jedem Rechenzentrum befinden sich (Stand 2008) 5 Mainframes plus zusätzlich drei weitere Mainframes, die als Coupling Facilities eingesetzt werden. Jedes Rechenzentrum enthält 12 FICON Switsche (Director) für den Anschluss der Plattenspeicher (Enterprise Storage Server) und Magnetband-Libraries.

Es ist durchaus üblich, dass Enterprise Storage Server und Magnetband-Libraries von anderen Firmen als IBM stammen. Bei der UBS stammen die Enterprise Storage Server teilweise von der Firma Hitachi. Die Magnetband Library stammt von der Firma Storage Tek, und kann insgesamt 26 400 Magnetbandkassetten aufnehmen. Zusätzlich werden Virtual Tape Server eingesetzt.

Anmerkung: In 2011 wurden die vorhandenen 2 x 5 älteren Mainframes durch 2 x 3 z196 Rechner mit insgesamt wesentlich höherer Leistung ersetzt. Vier weitere Mainframe Rechner dienen als Coupling Facilities.

Mainframe CPU Leistung wird traditionell in MIPS (Million Instructions per Second) gemessen. Hierzu dient ein IBM proprietäres Benchmark, welches in Assembler geschrieben ist und deshalb nicht auf andere Plattformen portiert werden kann. Mainframe Performance Vergleiche mit anderen Architekturen sind fast unmöglich. Als Faustformel kann dienen, dass ein moderner zEC12 Mikroprozessor marginal schneller als ein Hochleistungs- x86 oder PowerPC Mikroprozessor ist.

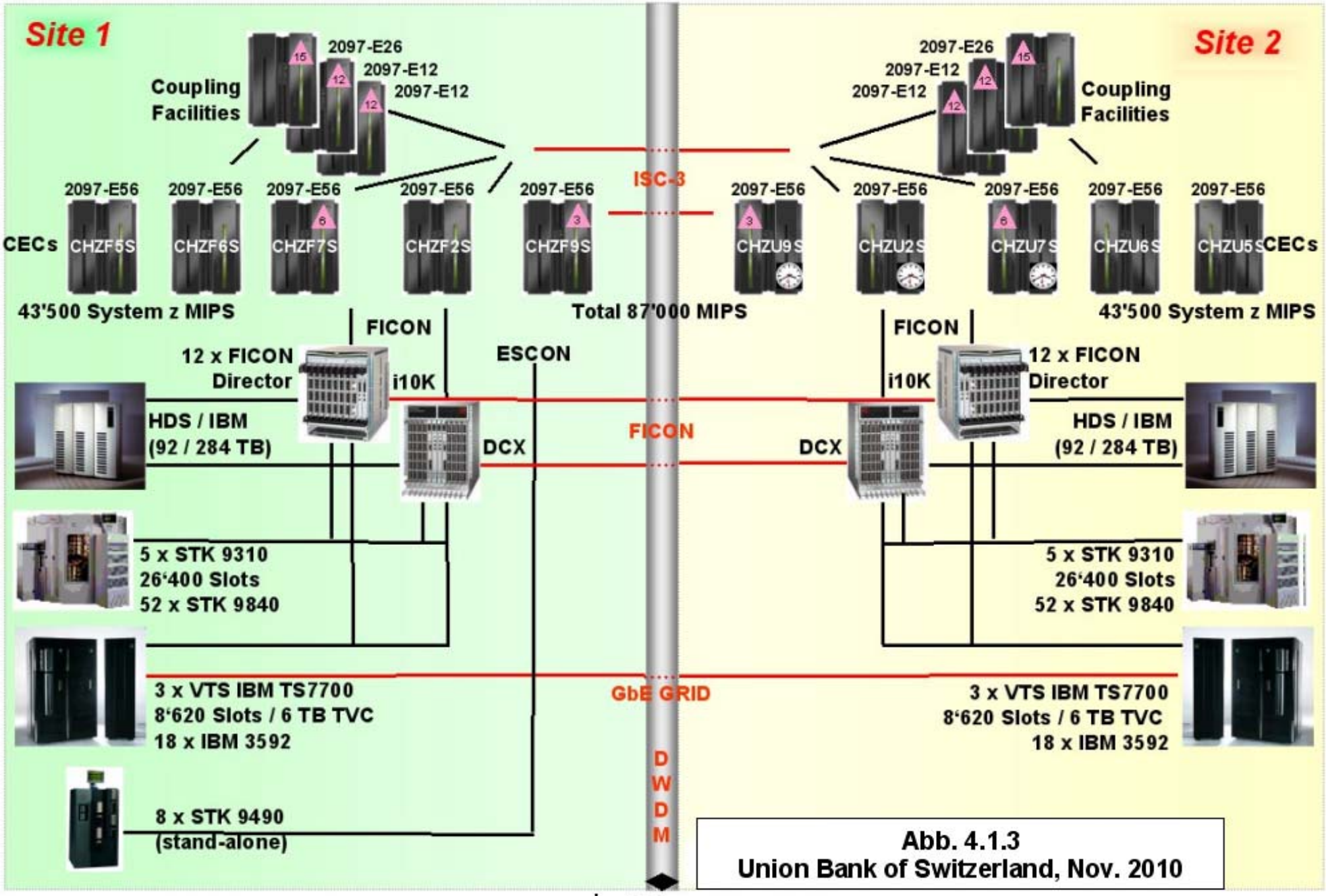


Abb. 4.1.3
Union Bank of Switzerland, Nov. 2010

STK 9310, 9490 StorageTek Cartridge Subsystem, HDS Hitachi Data Systems

14.1.5 Multiple Sysplexes

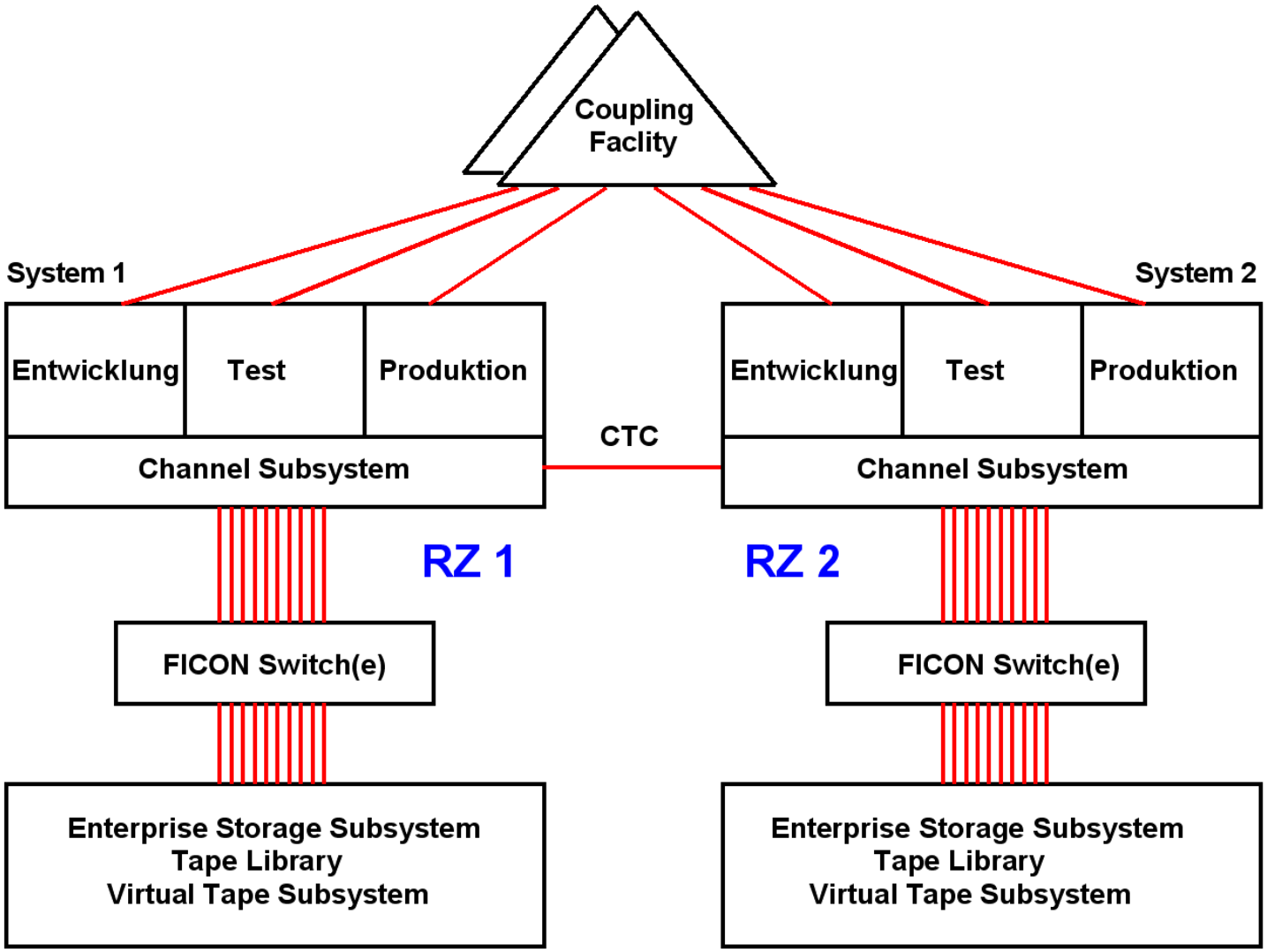


Abb. 14.1.4
Multiple Sysplex Configuration

Der eigentliche Haupteinsatz eines z/OS Rechners für Aufgaben wie Transaktionsverarbeitung, Stapelverarbeitung und Datenhaltung wird als “Produktion” bezeichnet.

Neben einem Sysplex für die Produktion existiert typischerweise ein getrennter Sysplex für die Entwicklung neuer Anwendungen. Ein weiterer Sysplex wird für das Austesten neu entwickelter Anwendungen benötigt, ehe sie für die Produktion frei gegeben werden.

Häufig existieren weitere Sysplexe für Sonderaufgaben.

Es ist ohne weiteres möglich (und üblich), mehrere logische Sysplexe auf eine physische GDPS Infrastruktur abzubilden. Dabei verfügt in jedem Rechenzentrum ein Sysplex über mindestens eine LPAR, wobei auch mehrere LPARs denkbar sind.

14.1.6 CreditSuisse

Die *Credit Suisse* Group ist die zweitgrößte Bank in der Schweiz mit Hauptsitz in Zürich.

Die beiden Rechenzentren in Zürich und Horgen sind etwa 15 km voneinander entfernt.

In beiden Rechenzentren stehen jeweils zwei physische Mainframe Rechner (PC10 und PC20 in Horgen) sowie PC60 und PC70 (in Zürich). Auf jedem der 4 Rechner befinden sich mehrere LPARs, z.B. die LPARs mit den Namen S11, GA1, S21, GB1, R21 usw. auf dem Rechner PC10 am Standort Horgen.

Unterhalten werden 3 logische Sysplexe für Entwicklung (RZ1), Test (RR2) und Produktion (RZ2). Hierbei erhält der Produktions-Sysplex den größten Teil der verfügbaren CPU Kapazität (40 00 MIPS) im Vergleich zu jeweils 5 000 MIPS für die beiden anderen Sysplexe, siehe Abbildung auf der folgenden Seite. Diese drei Sysplexe verfügen über LPARs auf allen 4 Rechnern.

Daneben existieren 6 weitere logische Sysplexe (RQ2, RZ4, RZ0, RZ8, RZZ und ZVM), die für Spezialaufgaben eingesetzt werden. Sie verfügen über weit weniger CPU Leistung (100 – 700 MIPS), und sind teilweise auf einen der beiden Standorte begrenzt. Auf z/VM laufen eine Reihe von zLinux virtuellen Maschinen für Entwicklungs- und Testzwecke.

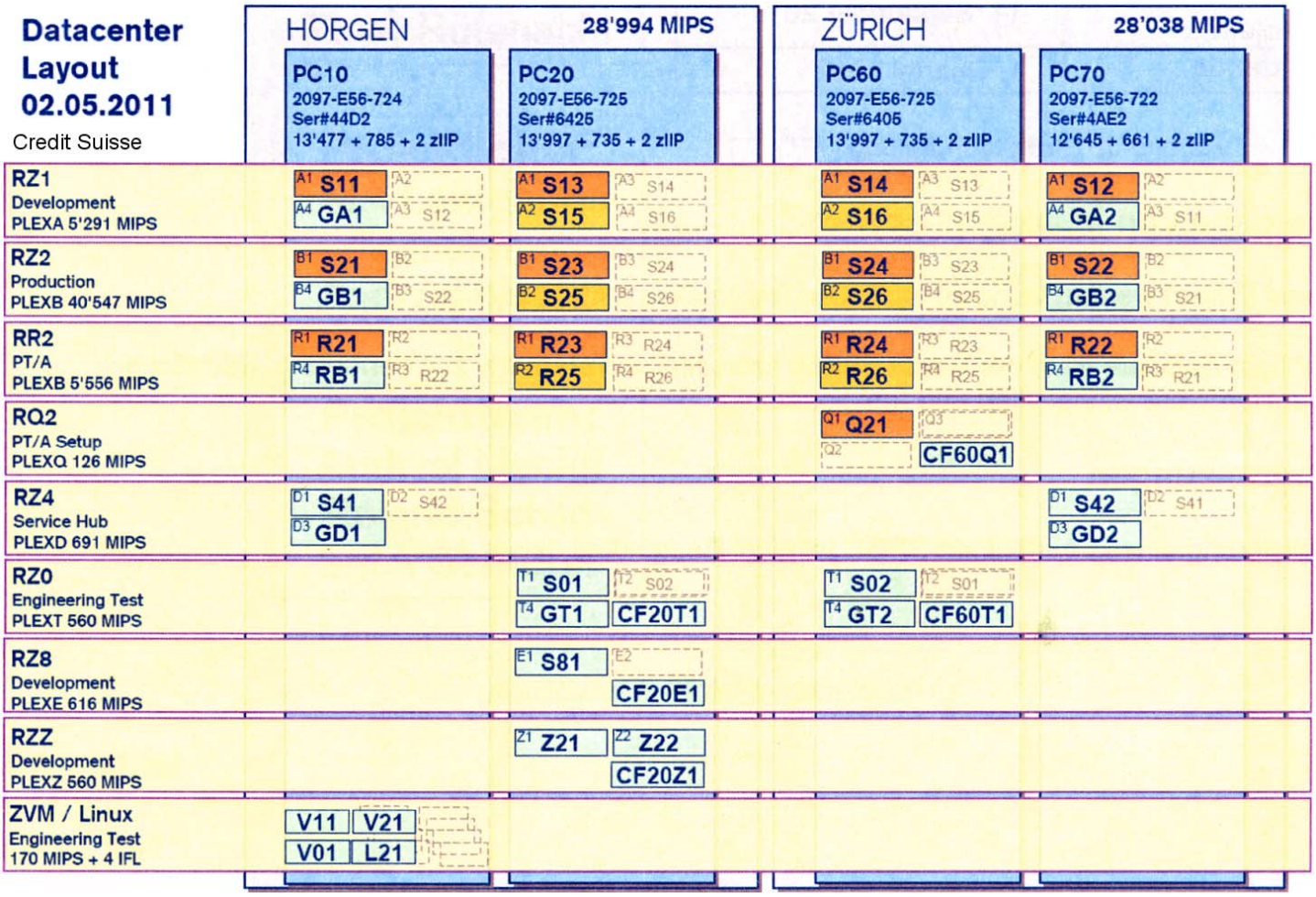


Abb. 14.1.5
Data Center Layout der Firma Credit Suisse, 2011

Kenndaten des Credit Suisse Data Centers:

Server	**6 751**
• Windows Application Servers	2 726
• Unix Servers	3 131
• File- & Print-Server	894
• Storage Capacity	4,600 PByte
Mainframes	
• Mainframe Systems (plus weitere in anderen Standorten)	4
• Installed MIPS	70 000
• Storage Capacity (DASD)	1 008 PByte
• Anzahl Datenbanken	54 500
• Anzahl IMS Transactions	27 Million pro Tag
• Anzahl Batch Jobs	86 000 pro Tag
• Tape Library Storage (plus weitere 13 PByte Shelf storage)	4.6 PByte

Kenndaten des Production-Sysplex:

- 100 Million MQ-Gets und 0.5 TByte Daten verarbeitet pro Tag
- 12 Million Corba Calls für IMS oder CICS pro Tag
- 20 Million CICS Transactions pro Tag (peak)
- 37.6 Million IMS Transactions pro Tag (peak)
- 220 IMS Databases
- 76 200 DB2 Tables
- 4.9 Billion SQL Calls pro Tag (peak)

50 000 unterstützte Benutzer in 550 Locationen
97 000 Workstations / Laptops
75 000 email Accounts
339 Millionen Emails/Jahr

100 Million MQ-Gets und 0.5 TB data processed per day besagt, dass WebSphere MQ in großem Umfang eingesetzt wird.

12 Million Corba Calls for IMS oder CICS per day besagt, dass RMI/IIOP in großem Umfang eingesetzt wird. RMI/IIOP wird später in Kapitel 6 behandelt.

Credit Suisse setzt sowohl DB2 wie die nicht relationalen IMS Datenbanken ein. Dies ist nicht untypisch. Weltweit sind auf allen in der Wirtschaft und in der Verwaltung eingesetzten Mainframe Rechnern etwa 60 % des Datenvolumens in DB2 Datenbanken, 30 % in IMS Datenbanken und 5 % in Adabas Datenbanken gespeichert. (Das VSAM Datenvolumen ist hierbei nicht berücksichtigt). Dabei existieren große Abweichungen von Unternehmen zu Unternehmen. So existieren zahlreiche Fälle, in denen nur DB2 oder nur IMS eingesetzt wird.

Für IMS existiert ein eigener Transaktionsmonitor, der an Stelle von CICS eingesetzt wird. Im Falle der Credit Suisse übersteigt das IMS Transaktionsvolumen das CICS Transaktionsvolumen, was allerdings nicht sehr typisch ist.

Credit Suisse Storage Management:

Plex	Data-Class	Storage-Class	Management-Class	Storage-Group
RZ0	130	24	90	32 (davon 5 tape, 1 VIO)
RZ1	136	26	96	30 (davon 2 tape, 1 VIO)
RZ8*	133	18	83	23 (davon 2 tape, 1 VIO)
PLEXZ	106	26	93	25 (davon 2 tape, 1 VIO)
RR2	108	28	100	37 (davon 2 tape, 1 VIO)
Q21	108	28	100	37 (davon 2 tape, 1 VIO)
RZ4	103	24	93	24 (davon 1 tape, 1 VIO)
RZ2	108	28	100	37 (davon 2 tape, 1 VIO)
Restliche*	(n/a)	(n/a)	(n/a)	(n/a)

Mai 2011

Abb. 14.1.6
Von der Credit Suisse verwendete DFSMS Konstrukte

In Band 1, Abschnitt 3.3.9 wurde dargestellt, wie mit Hilfe der z/OS DFSMS Komponente Data Sets allocated und jeweils einer Data Class, einer Storage Class und einer Management Class zugeordnet werden müssen.

Plattenspeicher (Volumes) mit identischer Data Class, Storage Class und Management Class werden zu einer Storage Group zusammengefasst.

Abb. 14.1.6 stellt dar, wie viele Data Classes, Storage Classes und Management Classes sowie Storage Groups in den einzelnen Sysplexes bei der Credit Suisse vorhanden sind. Die große Anzahl verdeutlicht die Komplexität der Datenverwaltung in einem modernen Rechenzentrum.

VIO (Virtual I/O) ist eine andere Bezeichnung für Virtual-Tape-Server, Band 1, Abschnitt 5.4.6.

Im Vergleich dazu verfügt unser Rechner am Lehrstuhl Technische Informatik der Uni Leipzig über je 1 Data Class, Storage Class, Management Class und Storage Group.

14.1.7 FIDUCIA IT AG , Karlsruhe

Die Fiducia ist ein führender IT-Dienstleister. Sie betreibt IT-Outsourcing für 790 Volksbanken und Raiffeisenbanken, Privatbanken sowie Zentralinstitute. Installierte Endgeräte bei den angeschlossenen Banken und Instituten:

- **100.998 PC-Arbeitsplätze in den Banken**
- **6.611 Server – 7 x 24 Stunden im Online-Betrieb, davon 3.885 Solaris Server, u.a. distributed WebSphere**
- **10.971 Geldautomaten**
- **10.757 Kontoauszugsdrucker**

Installation:

- **2 Rechenzentren, 7 x 24 Stunden im Online-Betrieb**
- **5 x z10 Rechner (59.005 MIPS), je 512 GByte, 5 LPAR/z10, bis zu 25 CPU/LPAR**
- **612 TB Plattenspeicherkapazität HDS, EMC**
- **7.492 TB Kassettenspeicher-Roboter**
- **40 % IMS, 60 % DB2**

Verarbeitet über 16,22 Milliarden Transaktionen pro Jahr.

Spitzenvolumen am 2.1.2009, IMS/DC

- **70 Millionen Host-Transaktionen/Tag**
- **3.062 Host-Transaktionen/Sek.**

30 Mil. LOC (Lines of Code) in Cobol; weitere 30 Mill. LOC in Java

14.1.8 Generali Versicherung, Aachen

Als weiteres Beispiel sei die Generali Information Services erwähnt. Installiert sind (Mai 2010)

- **15 000 MIPS mit 15 % MIPS Wachstum / Jahr**
- **ca. 100 TByte Disk, 30 000 aktive Benutzer**
- **Print Output: 270 Mill. Seiten / Jahr, 50 Mill. Briefe / Jahr.**
- **1,2 Mill. € Energiekosten / Jahr**

Generali Information Services beschäftigt 650 Mitarbeiter in der Anwendungsentwicklung

Websphere wurde offloaded nach AIX auf System p; Domino auf System x Blade Server

Generali benutzt eine typische GSPS Konfiguration mit drei Sysplexes für Entwicklung, Produktion und Test, sowie weiteren Sysplexes für die Tochtergesellschaften in Belgien, Österreich und Spanien.

Ein Virtual Tape Server (VTS, anderer Name Virtual I/O, VIO) ist ein Speicher auf Basis eines Festplatten Arrays, der nach außen hin eine Tape-Library emuliert. Siehe Band 1, Abschnitt 5.4.6.

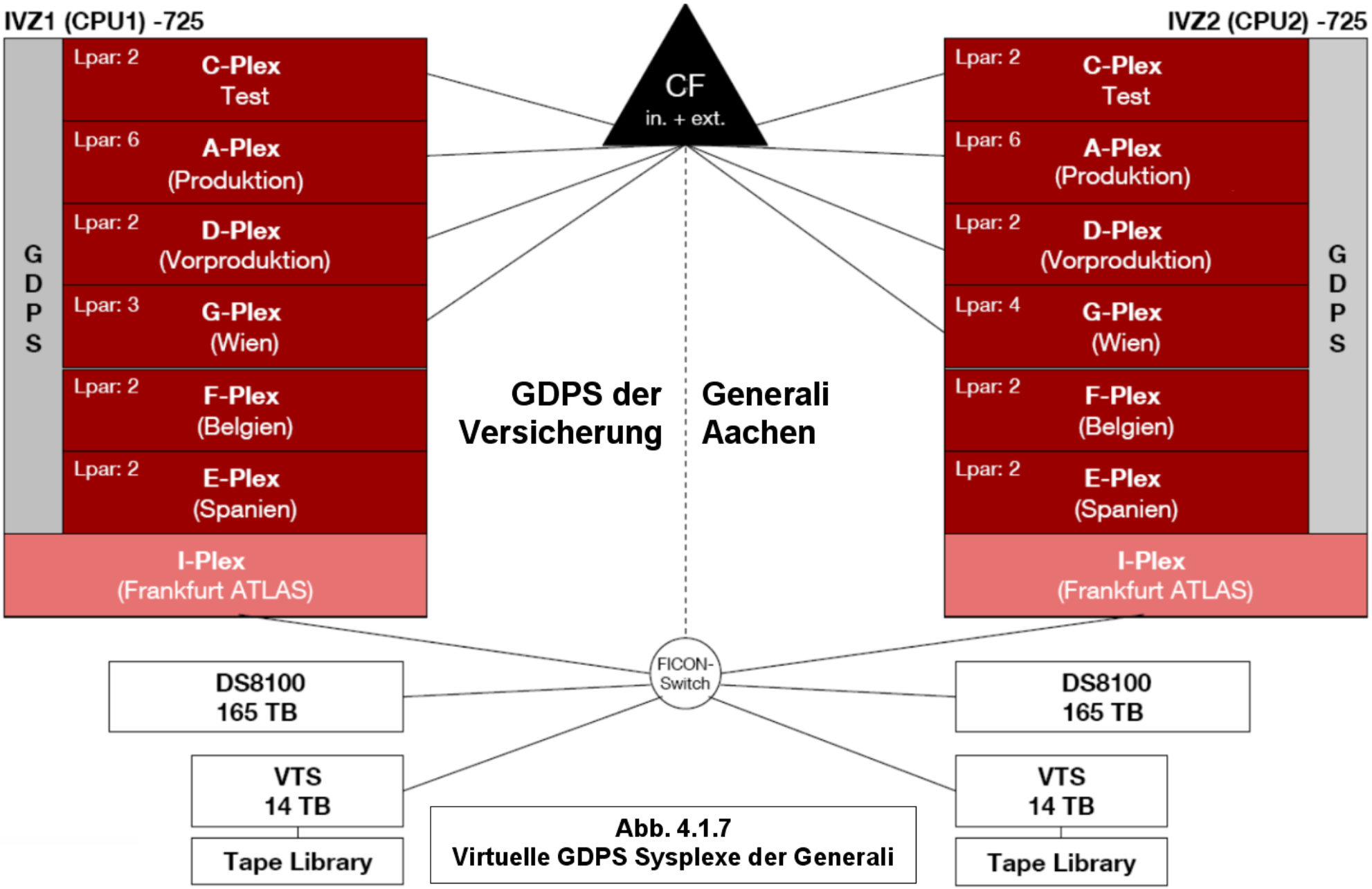


Abb. 4.1.7
Virtuelle GDPS Sysplexe der Generali

14.1.9 Rechenzentrum der gkv Informatik AG in Wuppertal

Die gkv Informatik ist ein IT-Dienstleister für gesetzliche Krankenversicherungen im Verbund von AOKs und BARMER GEK. Die Unterhält die IT Infrastruktur für 37 000 Mitarbeiter. Bewältigt den zentralen Beitragseinzug für 16 Millionen Versicherte. Stand Dezember 2010:

- **zentrales Rechenzentrum mit 1.600 qm Nutzfläche,**
- **20.000 Glasfaser Kabel mit einer Gesamtkabellänge > 100 km**
- **4 Diesel- Notstromaggregate mit 4,4 MWatt Leistung,**
- **4 IBM z10-Mainframes mit 40.000 MIPS,**
- **400 TByte Festplatten Kapazität**
- **sechs Bandrobotersysteme mit 3.500 TByte Kapazität ,**
- **400 UNIX-Server (AIX, Solaris, Linux)**
- **3.000 Windows-Server, davon 1.500 in Citrix Server-Farmen.**
- **Serverfarm mit 1.000 Bladesystemen für simultanen-Zugriff von > 12.000 Mitarbeitern**
- **120.000 Clients (PCs, Notebooks, Thin-Clients, Multifunktionsdruckers)**
- **700 Mitarbeiter der gkv Informatik sichern die kontinuierliche Erbringung der IT-Dienstleistungen**
- **VoIP-Netz mit 44.000 Endgeräten an 1.000 Orten**

14.1.10 Kommunales Rechenzentrum St. Gallen

Das Verwaltungsrechenzentrum AG (VRSG) in St. Gallen ist ein Dienstleistungsunternehmen für Städte und Gemeinden in der östlichen Schweiz. Es bedient 260 Kunden (Kantone/Gemeinden/öffentliche Verwaltungen), 7500 Benutzer, 50 Mio. Franken Umsatz (2010), 220 Mitarbeiterinnen und Mitarbeiter. VSRG betreibt 2 Rechenzentren in der St.Leonardstr. in St. Gallen sowie im St. Gallen Stadtteil Winkeln. Die Entfernung zwischen beiden Standorten beträgt etwa 7 km. Es handelt sich um eine weniger große Installation.

An beiden Standorten ist je ein (kleinerer) z10 Rechner installiert (Sept. 2011). Die beiden Standorte implementieren zwei logische „Geographically Dispersed Parallel Sysplex“ (GDPS). Jeder Rechner verfügt über 14 Prozessoren plus SAPs. (Ein z10 Rechner kann maximal 64 Prozessoren plus 12 SAPs enthalten). Bei den Prozessoren (PUs in IBM Terminology) handelt es sich um 6 normale CPUs (CPs in IBM Terminology), sowie 8 Spezialprozessoren, spezifisch

- **2 zIIP (System z Integrated Information Processor) für DB2 Verarbeitung**
- **2 zAAP (System z Application Assist Processor) für Java und XML Verarbeitung**
- **3 IFL (Integrated Facility for Linux) für zLinux Verarbeitung**
- **1 ICF (Integrated Coupling Facility). Dies ist ein Prozessor für eine LPAR, welche die Aufgabe einer Coupling Facility übernimmt.**

zIIPs, zAAPs, IFLs und ICFs benutzen die gleiche Mikroprozessor Hardware wie CPUs (und SAPs), werden jedoch bei der System Installation für ihre Spezialaufgaben „charakterisiert“, was Konsequenzen in Bezug auf Performance, Verkaufspreis und Software Lizenz Gebühren hat.

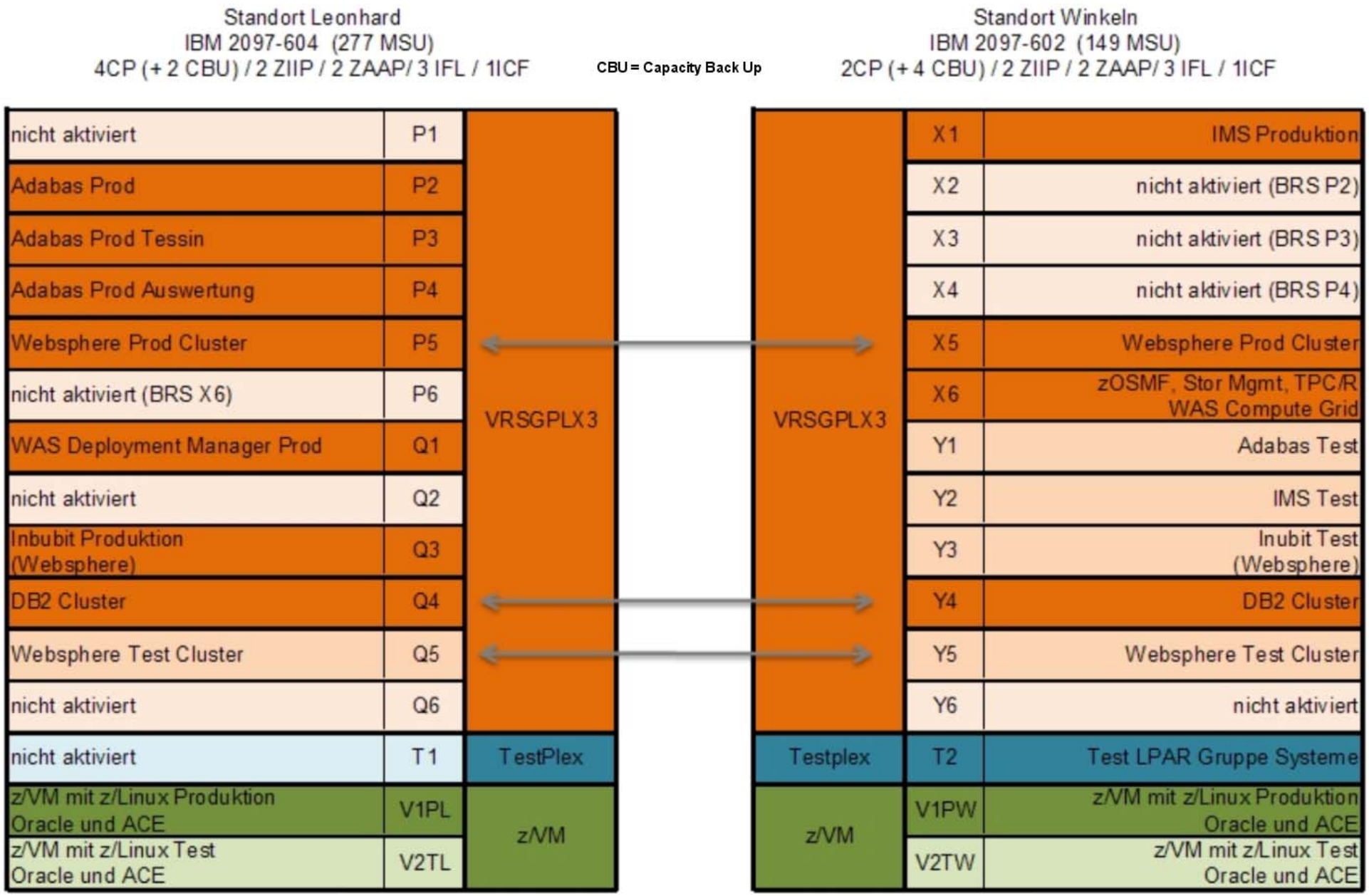


Abb. 4.1.8
Kommunales Rechenzentrum St. Gallen, Sept. 2011

14.2 Integrated Coupling Facility

14.2.1 Alternative zu einer Freestanding Coupling Facility

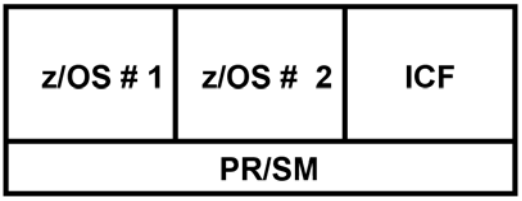


Abb. 14.2.1
Coupling Facility in einer LPAR eines Mainframe Rechners

Die Coupling Facility (CF) kann statt in einem getrennten Rechner auch in einer LPAR eines zSeries Knotens (System) untergebracht werden, wobei in anderen LPARs zwei oder mehr z/OS Images laufen. Diese Art der CF wird als Integrated Coupling Facility (ICF) bezeichnet. Die ICF läuft auf einer oder mehreren hierfür dedizierten CPUs eines SMP. ICFs haben in den letzten Jahren zunehmend an Popularität gewonnen. Das Kommunale Rechenzentrum St. Gallen benutzt Integrated Coupling Facilities.

Die ICF stellt CF Funktionalität ohne Coupling Links zu Verfügung. Letztere werden durch den PR/SM Hypervisor emuliert.

Aus Zuverlässigkeitsgründen sollten immer mindestens 2 physische oder integrierte CFs vorhanden sein.

Die entsprechende GDPS Konfiguration ist in Abb. 14.2.2 dargestellt.

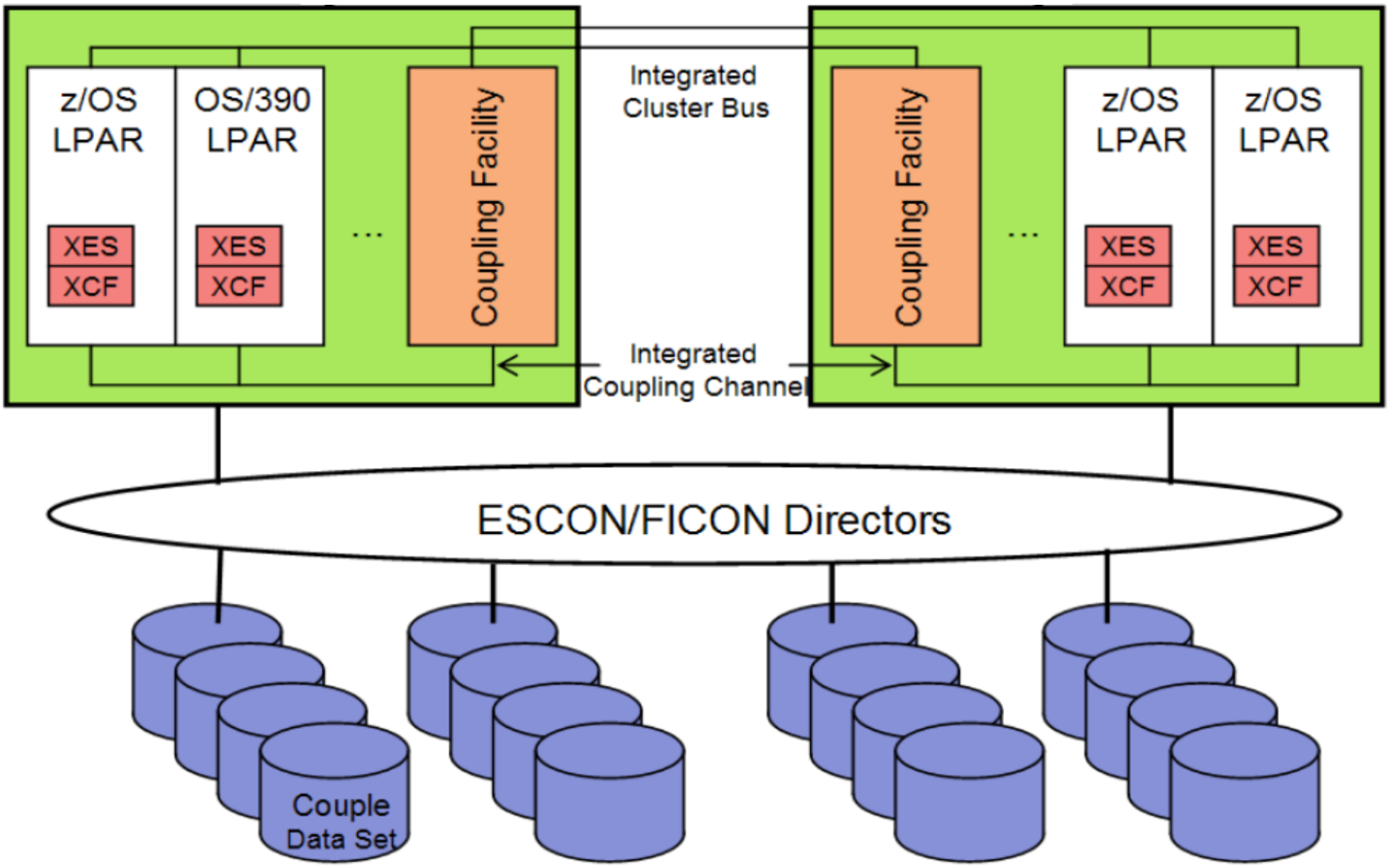


Abb. 14.2.2
Zwei Systeme mit integrierter Coupling Facility

Der Integrated Coupling Channel ist eine logische Struktur, welche die Aufgabe eines Coupling Links übernimmt, siehe Abschnitt 11.2.2.

14.2.2 Firma Endress+Hauser, Weil am Rhein

Ein faszinierendes Beispiel für eine ICF Installation ist die Firma Endress + Hauser. Die Endress+Hauser Gruppe ist einer der international führenden Anbieter von Messgeräten, Dienstleistungen und Lösungen für die industrielle Verfahrenstechnik. Das Unternehmen liefert Sensoren, Geräte, Systeme und Dienstleistungen für Füllstand-, Durchfluss-, Druck- und Temperaturmessung sowie Flüssigkeitsanalyse und Messwertregistrierung.

Zwei z10 EC (Modell E64) Systeme im Parallel-Sysplex-Betrieb. Pro System:

- **17 CPUs für z/OS Datenbankanwendungen**
- **2 CPUs für ICF**
- **32 CPUs für zLinux,**
- **1,5 TB Hauptspeicher**
- **Taktfrequenz: 4,2 GHz**
- **Stellfläche: je 2 m² (!) pro System.**

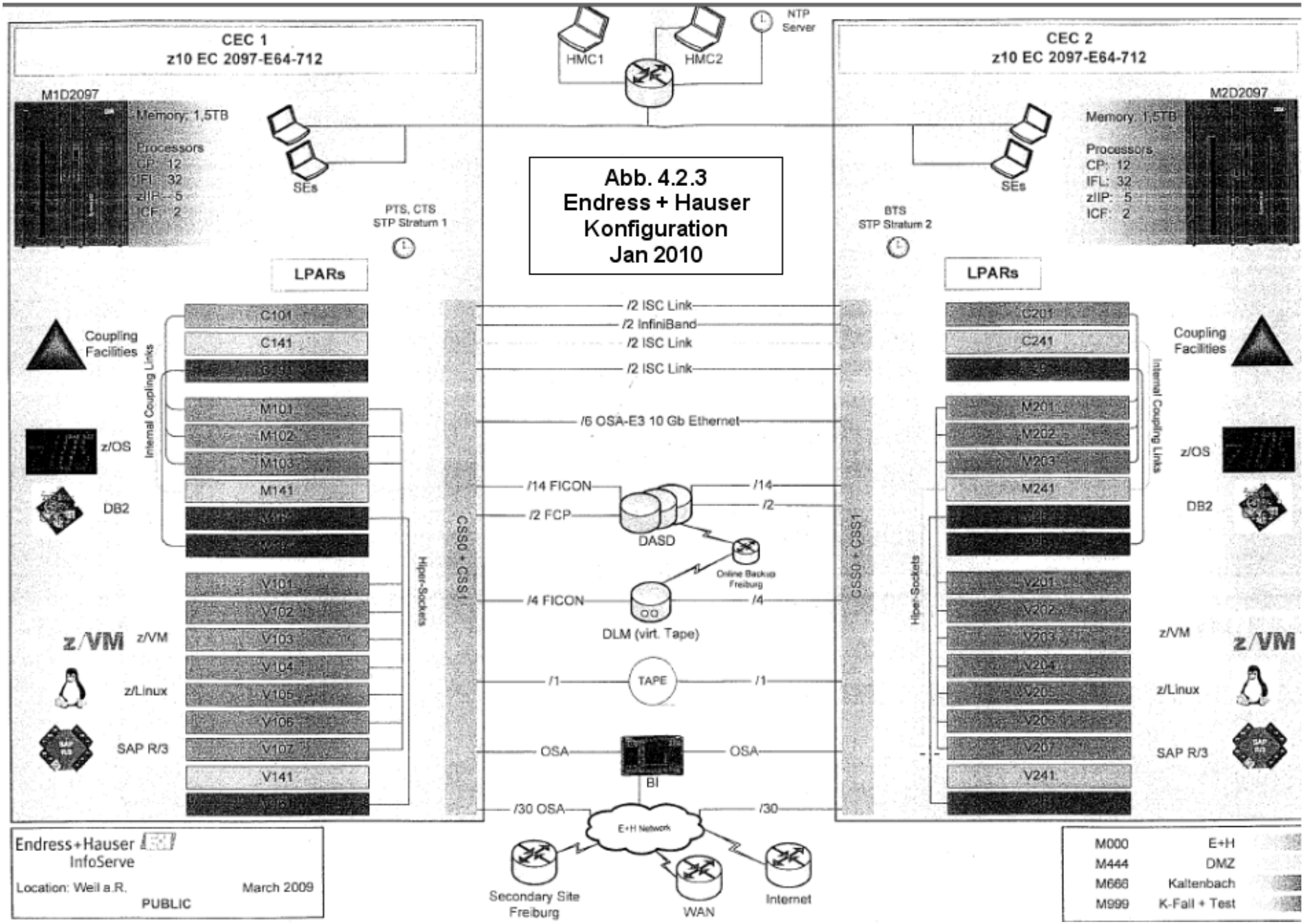
Unter zLinux laufen vor allem SAP Anwendungen, unter z/OS die dazugehörigen Datenbankanwendungen. Die HiperSockets-Technologie (Abschnitt 12.4.7) bietet extrem schnelle und sichere Konnektivität zwischen den SAP Anwendungs- und Datenbankservern.

19 SAP R/3 Produktionssysteme und 20 SAP R/3 Test-Systeme. verteilt auf 14 logische Partitionen (LPARs), laufen unter zLinux. SAP-R/3 Datenbanken auf der Basis von z/OS DB2 sind auf 6 logische Partitionen verteilt.

Unterstützt werden 3.700 registrierte Benutzer (drunter bis zu 2.500 gleichzeitige Benutzer). Diese Benutzer arbeiten in 35 Tochtergesellschaften an 71 Standorten weltweit und erhalten eine durchschnittliche Reaktionszeit von 0,5 Sekunden für SAP Anwendungen.

Endress + Hauser unterhält die vermutlich größte zLinux Produktionsinstallation in Deutschland.

Die folgende Abbildung 14.2.3 ist leider nur schlecht lesbar. Mit maximaler Vergrößerung werden jedoch viele Einzelheiten sichtbar.


CEC 1
z10 EC 2097-E64-712
M1D2097
Memory: 1,5TB
Processors
CP: 12
IFL: 32
zIIP: 5
ICF: 2
SEs
PTS, CTS
STP Stratum 1
LPARs
C101
C141
M101
M102
M103
M141
V101
V102
V103
V104
V105
V106
V107
V141
Coupling Facilities
Internal Coupling Links
z/OS
DB2
z/VM
z/Linux
SAP R/3
Hiper-Sockets
CSS0 + CSS1
HMC1
HMC2
NTP Server
Abb. 4.2.3
Endress + Hauser
Konfiguration
Jan 2010
/2 ISC Link
/2 InfiniBand
/2 ISC Link
/2 ISC Link
/6 OSA-E3 10 Gb Ethernet
/14 FICON
/2 FCP
DASD
Online Backup Freiburg
/4 FICON
DLM (virt. Tape)
/1
TAPE
OSA
BI
/30 OSA
E+H Network
Secondary Site Freiburg
WAN
Internet
CEC 2
z10 EC 2097-E64-712
M2D2097
Memory: 1,5TB
Processors
CP: 12
IFL: 32
zIIP: 5
ICF: 2
SEs
BTS
STP Stratum 2
LPARs
C201
C241
M201
M202
M203
M241
V201
V202
V203
V204
V205
V206
V207
V241
Coupling Facilities
Internal Coupling Links
z/OS
DB2
z/VM
z/Linux
SAP R/3
Hiper-Sockets
CSS0 + CSS1
Endress+Hauser
InfoServe
Location: Weil a.R.
March 2009
PUBLIC
M000 E+H
M444 DMZ
M666 Kaltenbach
M999 K-Fall + Test

14.2.3 Hardware Management Console

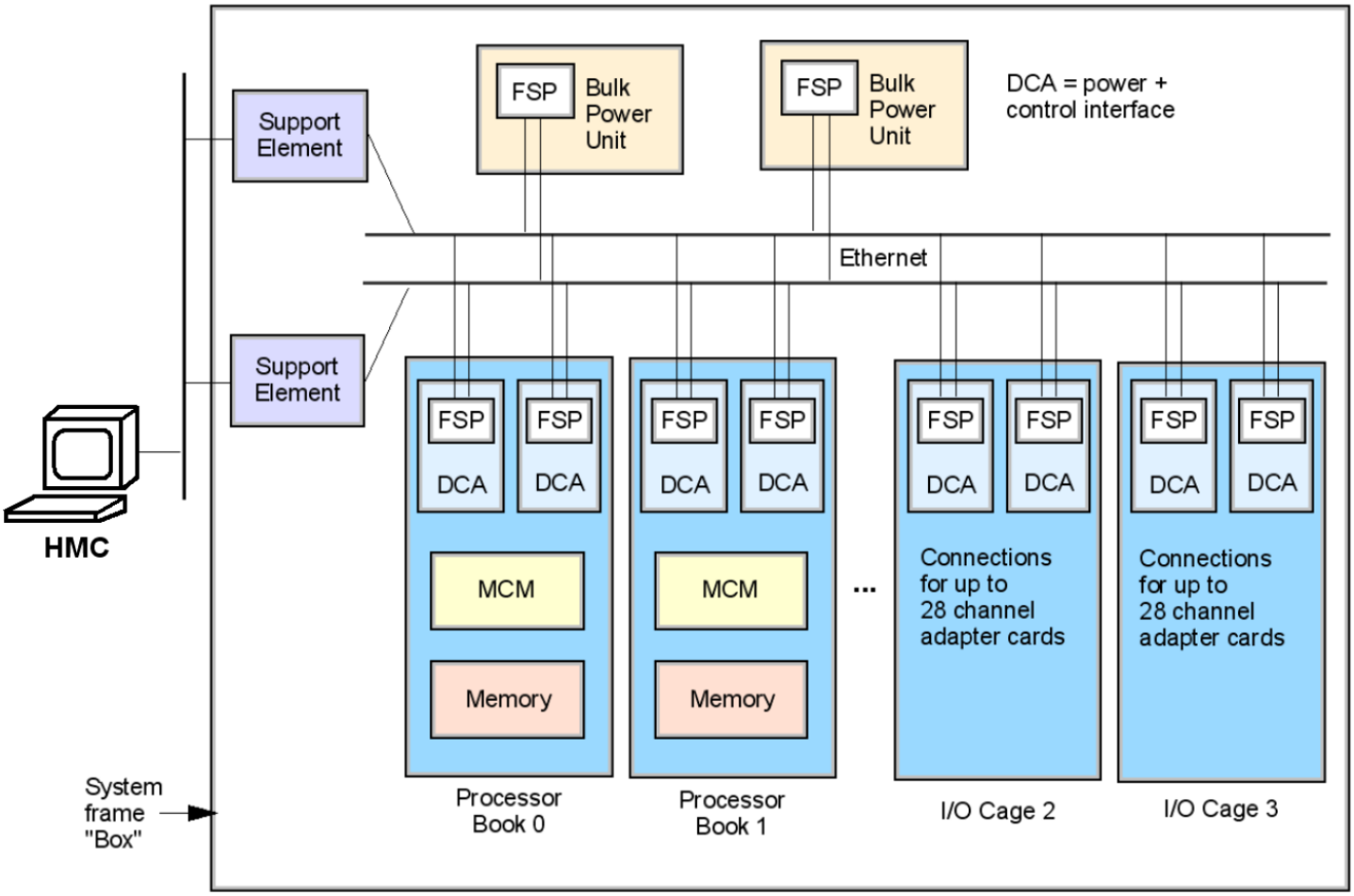


Abb. 14.2.4
Anschlüsse für das Support Element

Abbildung 14.2.4 zeigt das Innere eines System z Rechners. Für die Administration ist das Support Element (eingebauter Laptop, andere Bezeichnung Service Element) wichtig.

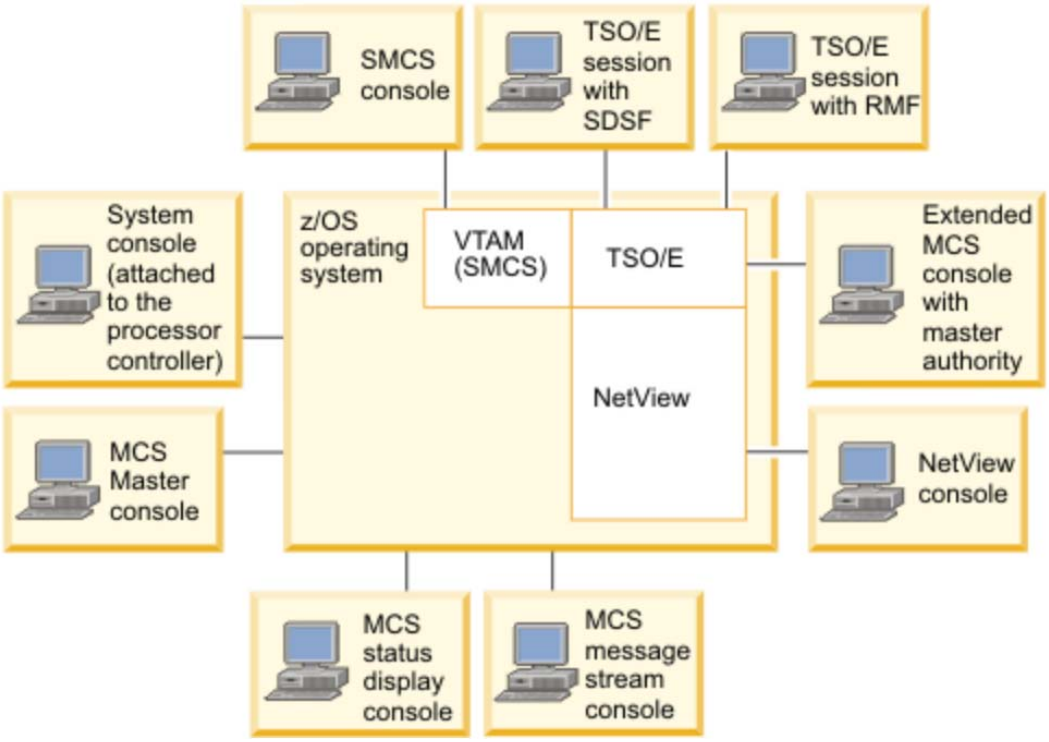
Das Support Element ist über eine interne Ethernet Verbindung mit einer ganzen Reihe von „Flexible Service Processoren“ (FSB) verbunden. FSBs sind kleine Baugruppen, die in die Books, I/O Cages und Power Supply Module integriert sind, siehe Band 1, Abschnitt 4.3.3. Sie erstellen kontinuierlich diagnostische Daten, die an das Support Element weitergegeben werden. Über das Support Element kann der Rechner auch gesteuert werde, z.B. konfiguriert oder hochgefahren werden.

Aus Gründen der Zuverlässigkeit sind das Support Element, die dazugehörigen FSBs und Ethernet Verbindungen doppelt vorhanden. Wenn ein Support Element ausfällt, kann der Betrieb mit dem zweiten Support Element fortgesetzt werden.

Die beiden Support Elemente sind in den Mainframe Rahmen fest eingebaut, und können nicht entfernt werden. Wenn ein Administrator das Support Element benutzen will, muss er die Tür des Mainframe Rahmens öffnen, um sich Zutritt zu verschaffen. Da dies lästig ist, existiert eine zusätzliche „Hardware Management Console“ (HMC), siehe Band 1, Abschnitt 1.3.11. Dies ist ein externer PC, der über ein unabhängiges Ethernetkabel mit den beiden Support Elementen verbunden ist.

Die HMC ermöglicht alle Funktionen, die auch mit dem Support Element durchführbar sind. Sie steht normalerweise in einem geschlossenen Raum mit kontrollierten Zutrittsrechten. Die HMC ist eine geschlossene Box, die bei einer gewaltsamen Öffnung selfdestructs. Die Kommunikation zwischen HMC und Support Element erfolgt kryptografisch verschlüsselt.

In einer Mainframe Installation existieren neben der HMC noch weitere Konsolen (Bildschirme) für die Administration des z/OS Betriebssystems, und für die Administration von Subsystemen wie z.B. DB2 oder WebSphere. Außerdem existieren in der Regel weitere zahlreiche Windows, Unix und Linux Rechner, die ebenfalls administriert werden müssen. Diese zahlreichen Bildschirm-Konsolen eines Unternehmens werden üblicherweise im gleichen Raum untergebracht und zu einem „Leitstand“ zusammengefasst.



Beispiel einer Console Konfiguration für eine Mainframe Installation

Die System Console ist eine Teilfunktion der Hardware Management Console.

14.2.4 Leitstand der Fiducia IT AG in Karlsruhe

Die Abbildungen 14.2.5 – 14.2.8 zeigen den Leitstand der Fiducia IT AG in Karlsruhe. Kennzeichnend ist neben den zahlreichen Computer-Bildschirmen eine zentrale Bildschirmwand, auf der selektiv die Inhalte einzelner Administrator-Arbeitsplätze wiedergegeben werden.

Der Leitstand steuert den Normalbetrieb und in einer Krise den Wiederanlauf aller Services.

Der Leitstand ist an 24 Stunden/Tag und 7 Tagen pro Woche besetzt. Die Systemadministratoren haben neben dem Platz im Leitstand noch einen normalen Büroarbeitsplatz für Ihre laufenden Tätigkeiten. Der normale Betrieb der IT Infrastruktur ist hochgradig automatisiert. Viele Funktionen sind nur für einen Störfall vorhanden, der hoffentlich nie auftritt.

Die Kosten, die bei einer Störung des IT Betriebes auftreten sind jedoch so astronomisch, dass alle Voraussetzungen geschaffen werden, um solche Störfälle möglichst rasch beenden zu können.

Bei (sehr seltenen) Störfällen versammeln sich der Krisenstab und die technischen Spezialisten in einem Konferenzraum, der durch eine Glaswand einen unmittelbaren Blick auf die Bildschirme des Leitstandes ermöglicht.

Leitstände, wie sie in der IT Industrie üblich sind, werden auch in anderen Industriezweigen eingesetzt, z.B. Kraftwerke, Raffinerien, Chemische Industrieanlagen, Raumfahrt, andere ...

Abb. 4.2.5
Fiducia Leitstand (1)

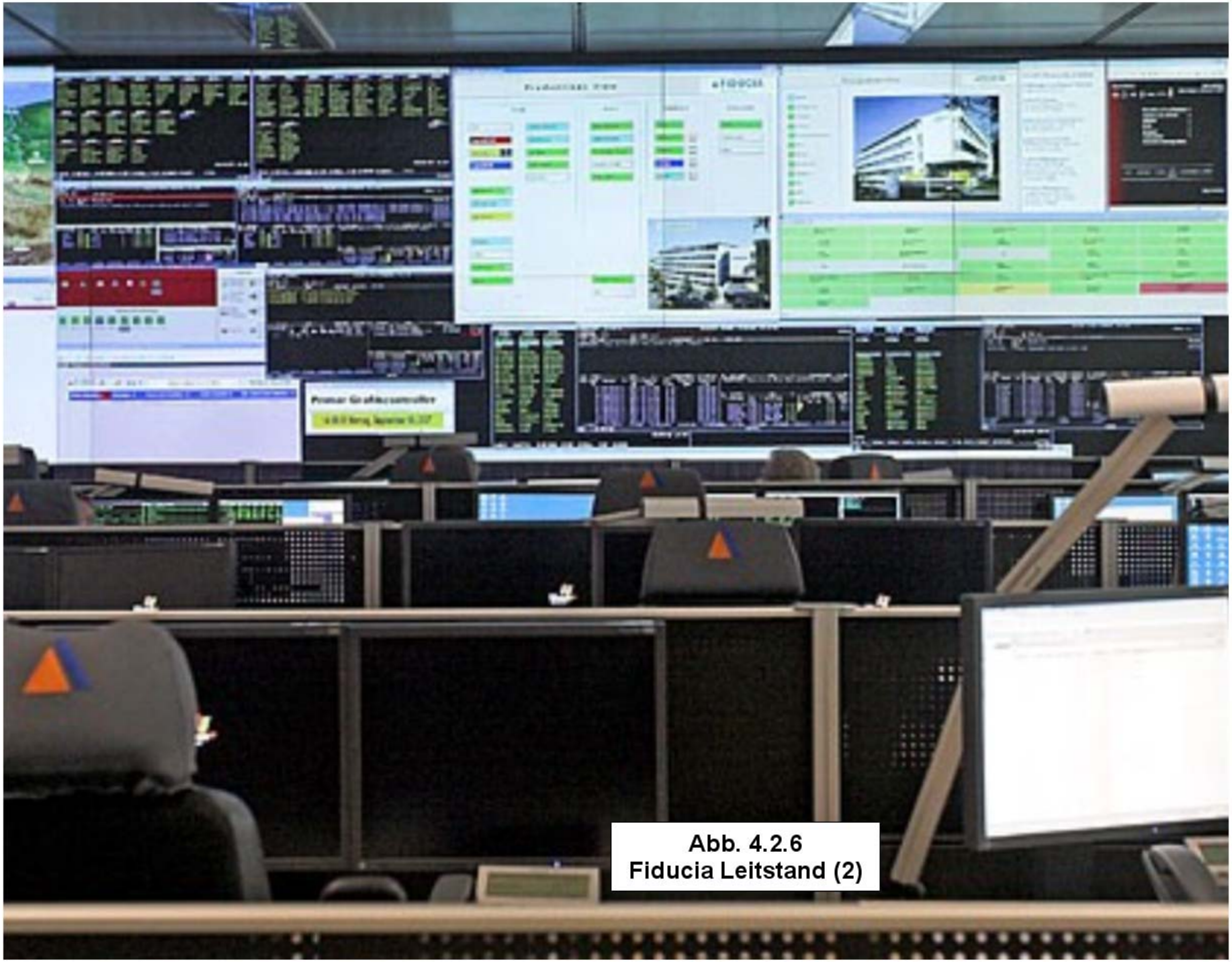


Abb. 4.2.6
Fiducia Leitstand (2)

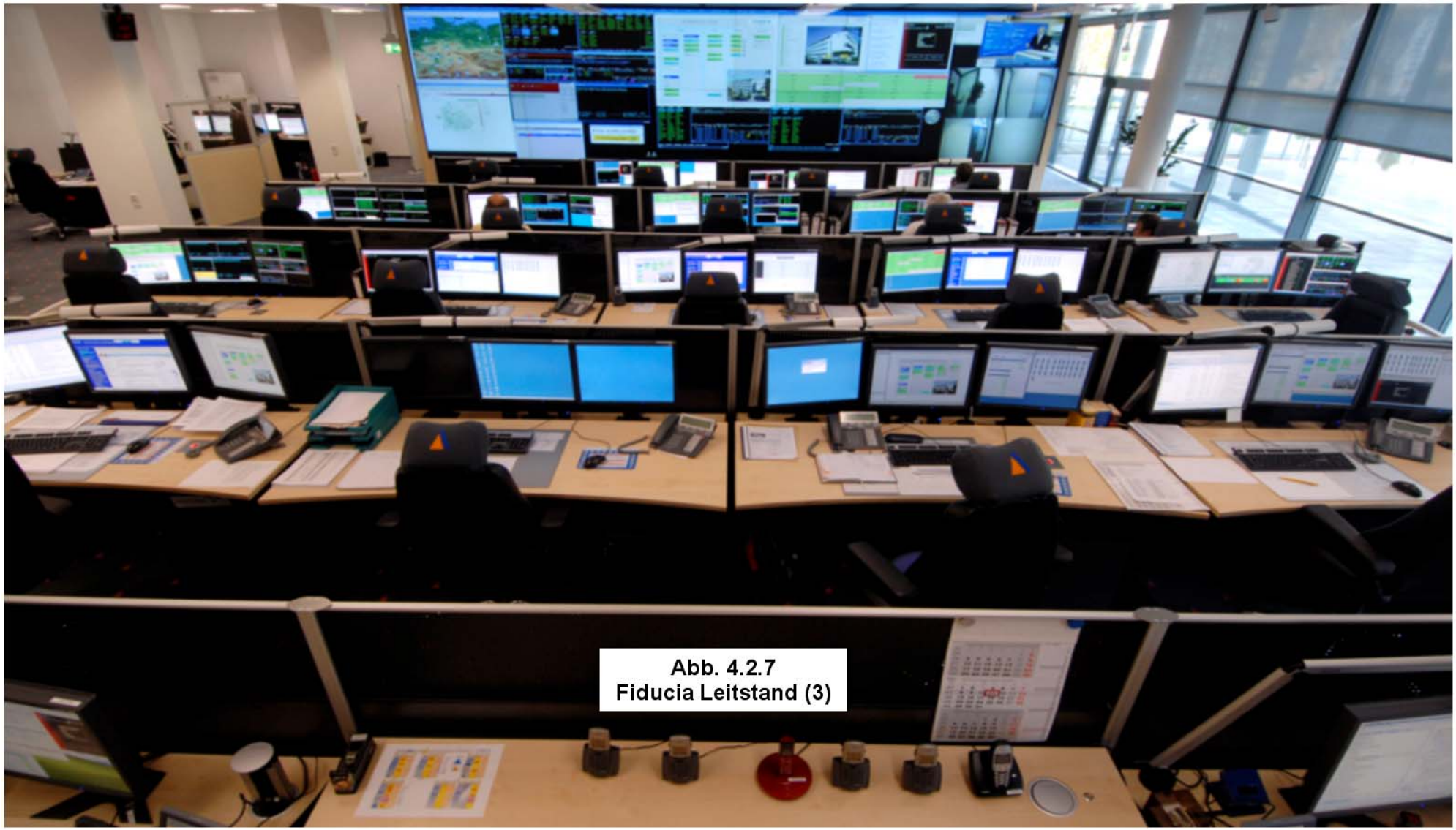


Abb. 4.2.7
Fiducia Leitstand (3)

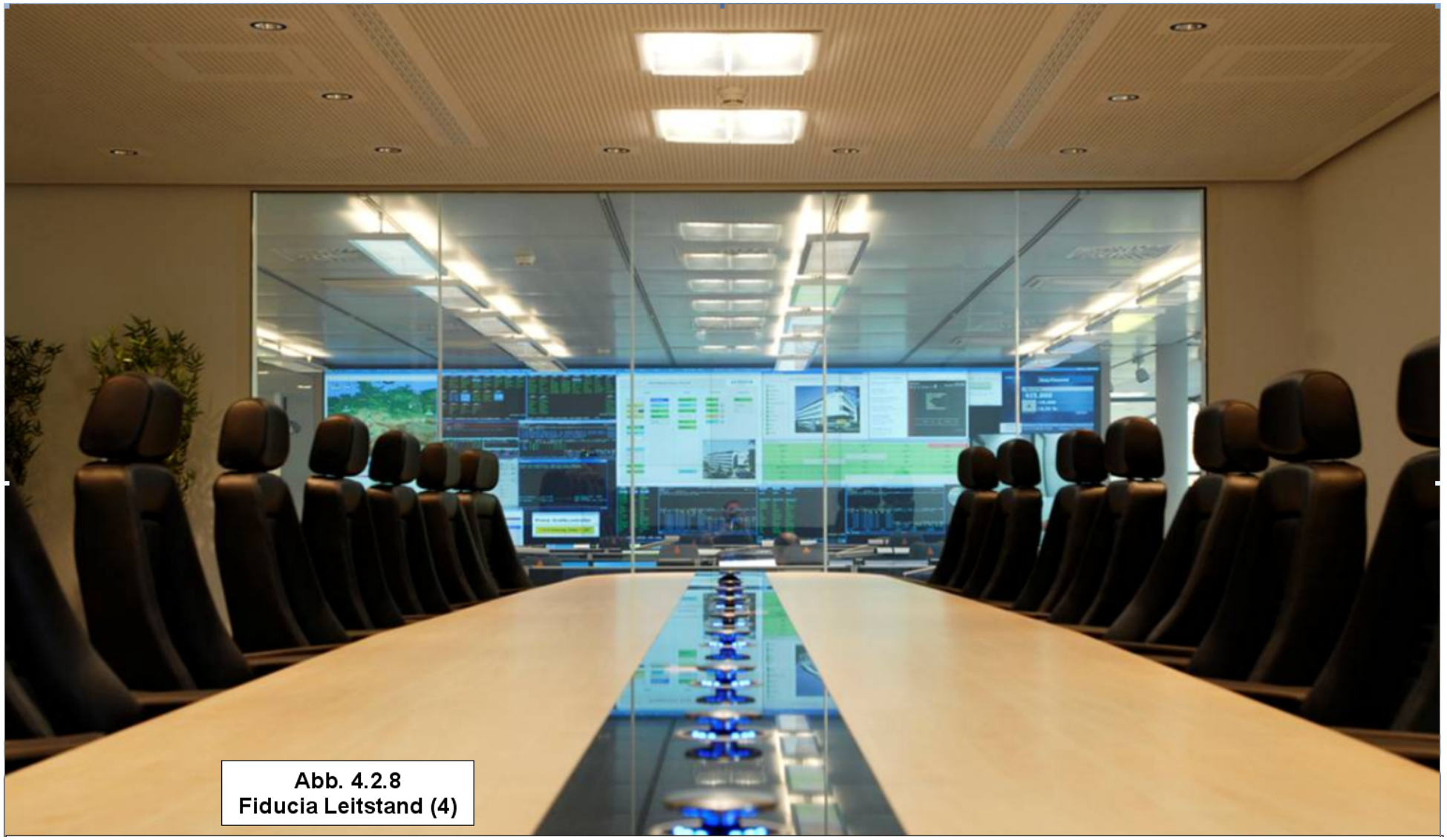


Abb. 4.2.8
Fiducia Leitstand (4)

14.2.9 Firma Jungmann Systemtechnik

Die Firma Jungmann Systemtechnik in Buxtehude, Landkreis Stade in Niedersachsen, ist der führende Hersteller von Leitständen in Deutschland.

Die folgenden Abbildungen 14.2.9 und 14.2.10 zeigen zwei Demonstrationsleitstände der Firma Jungmann.

Das Unternehmen liefert nicht nur die Computerindustrie, sondern auch andere Industriezweige wie z.B. Kraftwerke, Raffinerien, Chemische Industrieanlagen, Raumfahrt und andere.

Jungmann liefert auch ergonomisch besonders bequeme Sessel, die Ermüdungserscheinungen bei den Systemadministratoren verringern sollen.

Abb. 4.2.9
Demonstations-Leitstand der Firma Jungmann Systemtechnik, Buxtehude

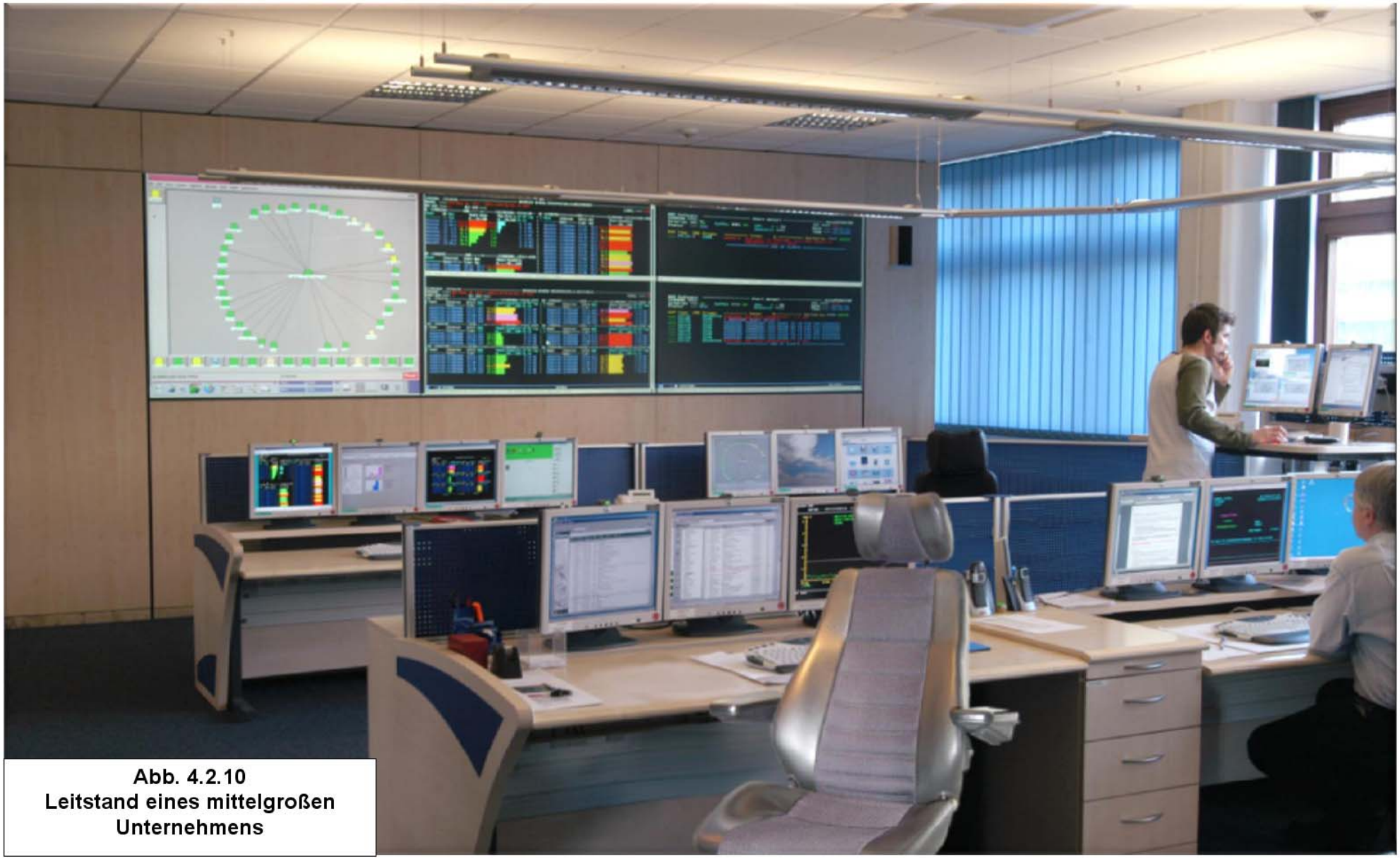


Abb. 4.2.10
Leitstand eines mittelgroßen Unternehmens

14.3 zBX und der Unified Resource Manager

14.3.1 IT Infrastruktur in einem Großunternehmen

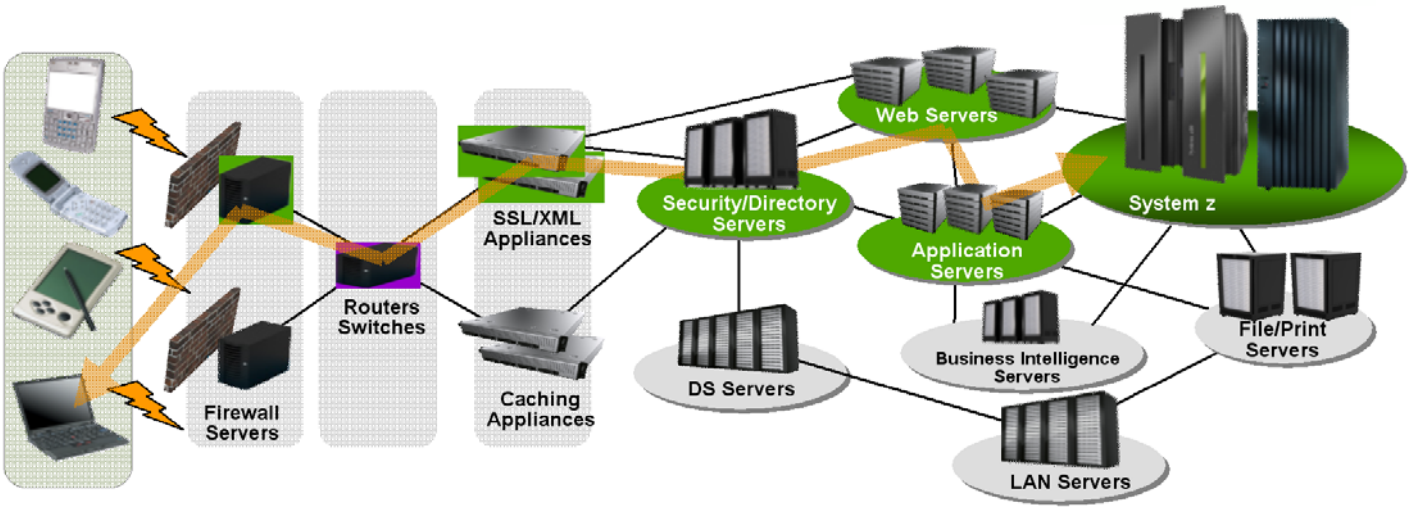


Abb. 14.3.1
Große Unternehmen benutzen Hunderte oder Tausende distributed Server

Die IT Infrastruktur eines Großunternehmens besteht neben den Mainframes typischerweise aus einer großen Anzahl weiterer nicht-Mainframe Server. Dies können viele tausend Server sein, die spezielle Aufgaben wahrnehmen, wie Kryptografie, Secure Socket Layer Verarbeitung, LDAP Directory Services, Routing, Netzwerk Management, Business Intelligence, Präsentation Logik, SAP Server, Web Application Server und vieles anderes.

Häufig werden von Spezialfirmen Anwendungen entwickelt, die nur unter einem spezifischen Betriebssystem (Windows, Linux, Solaris, MacOS, andere) und/oder einer spezifischen Hardware (x86, Sparc, PowerPC) laufen.

Im Gegensatz zum Mainframe wird dieser Teil der IT Landschaft als „distributed“ bezeichnet.

Siehe auch Abb. 14.3.2 , Das mittelgroße VRSG, Verwaltungsrechenzentrum AG St.Gallen, Dienstleistungsunternehmen für Städte und Gemeinden in der Ost-Schweiz, zeigt die Komplexität der distributed IT Infrastruktur (DMZ = Demilitarized Zone).

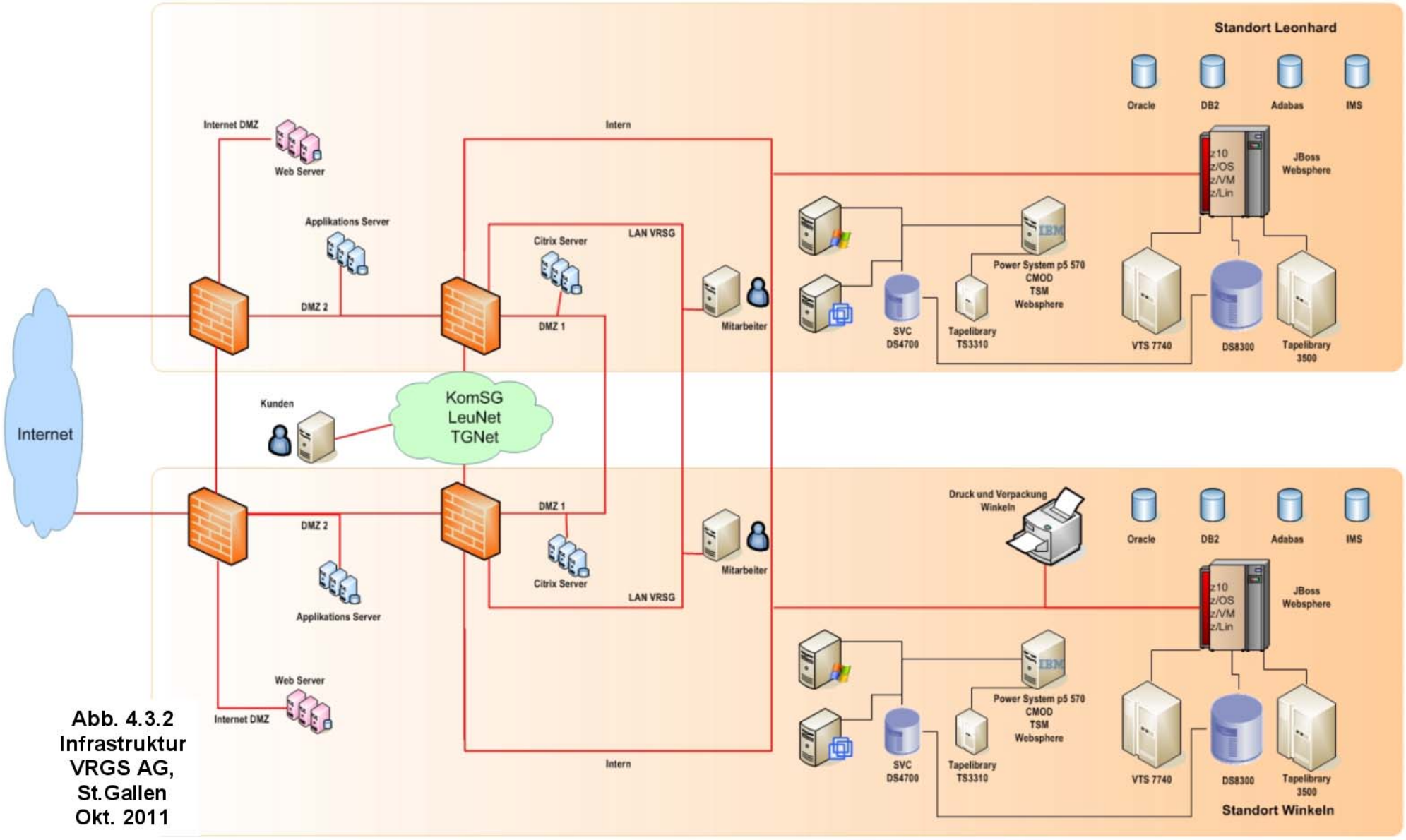


Abb. 4.3.2
Infrastruktur
VRGS AG,
St.Gallen
Okt. 2011

Der distributed Teil der IT Infrastruktur stellt fast immer einen sehr unorganischen Wildwuchs dar. In kleinen Unternehmen ist eine einheitliche IT Strategie relativ leicht zu etablieren und einzuhalten, z.B. „wir benutzen ausschließlich Microsoft Server Software“ (Microsoft Server, MS Directory Server, MS Transaction Server, MS SQL Server). Beim Focus Verlag in München werden seit Jahrzehnten nahezu ausschließlich Mac OS Server eingesetzt.

In großen Unternehmen fehlt oft eine einheitliche Strategie, oder wird nicht konsequent befolgt, ändert sich bei einem Wechsel in der Unternehmensführung oder scheitert an Sachzwängen.

Die distributed Infrastruktur hat eine hohe Komplexität und einen großen und aufwendigen Administrationsaufwand zur Folge. Spezifisch haben die zahlreichen Plattformen unterschiedliche und inkompatible Management Interfaces.

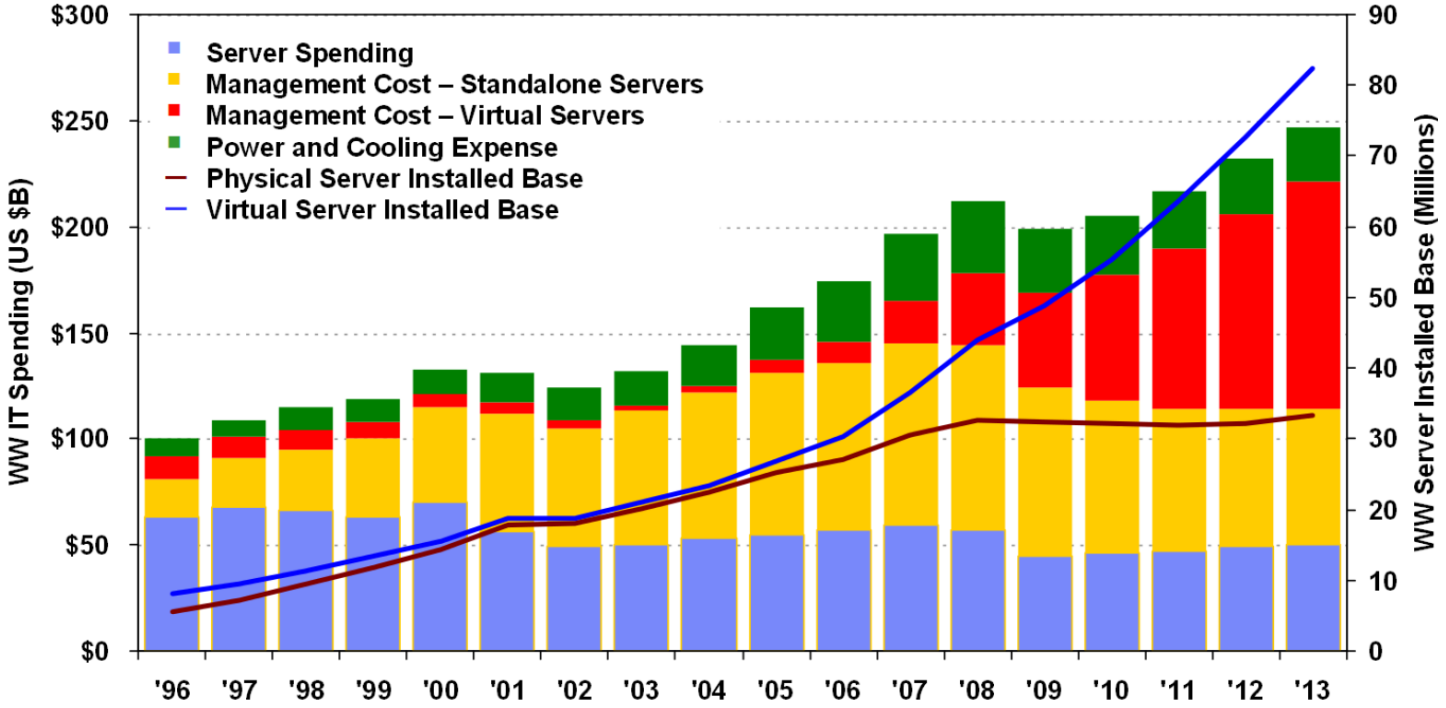


Abb. 14.3.3
Management Kosten übertreffen bei Weitem die Kosten für Hardware und Software

Source: IDC – “Three Data Centers – One Vision?”, March 2010

www.cedix.de/VorlesMirror/Band2/RayJones02.pdf

In vielen Unternehmen sind die distributed Server in den einzelnen Fachbereichen installiert und über das Unternehmen verstreut. Häufig wird versucht, die Server im Rechenzentrum zu konzentrieren um einen Teil der Kosten und des Administrationsaufwandes einzusparen. So sind z.B. bei der Fiducia AG, Karlsruhe neben den 5 Mainframes mit 60 000 MIPS weitere 6 611 Server vorhanden, davon heute (2011) 3.885 zentralisiert im Rechenzentrum.

Abb. 14.3.4
Server Farm im Fiducia Rechenzentrum

Die Fiducia IT AG Karlsruhe hat einen Teil der dezentral in den Fachabteilungen stehenden Server zentralisiert und einheitlich auf Sun Solaris umgestellt. Im Rechenzentrum stehen neben den 5 Mainframes weitere 3.885 Unix Solaris Server. die als Blades implementiert sind Dies verringert deutlich den Administrations-Aufwand, der aber dennoch wesentlich höher als der Mainframe Administrations-Aufwand bleibt.

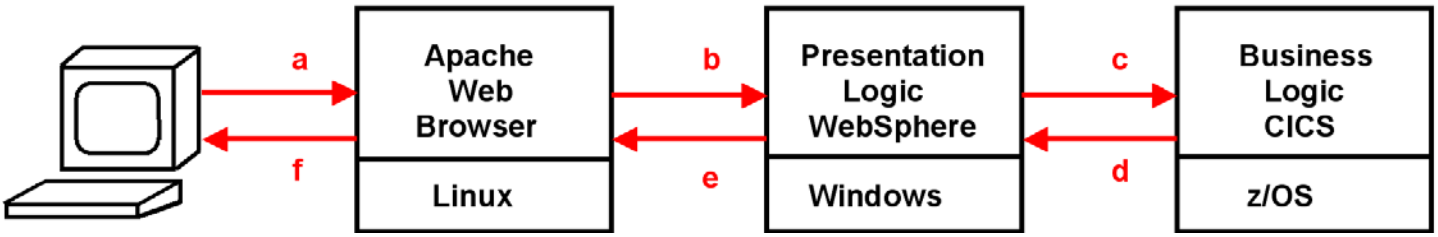


Abb. 14.3.5
Reichweite des Work Load Managers

Die Komplexität sei an Hand eines Beispiels demonstriert.

Gezeigt ist eine typische Konfiguration, bei der die Business Logic unter z/OS und CICS läuft, und die Presentation Logik unter einem „distributed“ WebSphere auf einem Windows Rechner.

Der z/OS Work Load Manager kann die Response Time zwischen den Punkten c und d messen, nicht aber zwischen den Punkten a und f, also dem, was der Benutzer an seinem Bildschirm sieht. (Wäre der Browser und WebSphere auch unter z/OS installiert, wäre das kein Problem). Wenn die Response Time ungenügend ist, ist es schwierig herauszufinden, ob das Problem an dem Linux oder dem Windows Server liegt. Der Administrator tappt hier im Dunkeln. Dies gilt besonders, wenn auf Grund der hohen Work Load nicht ein sondern 10 oder 20 Apache und WebSphere Server installiert sind.

Was passiert: Der Administrator rät, dass das Problem bei dem WebSphere Server liegt, und installiert zwei zusätzliche Server. Wenn er Pech hat, wird die Response Time dadurch noch schlechter.

Die CPU Utilization (andere Bezeichnung System Utilization) in distributed Servern ist häufig nur 20 %, während Mainframes routinemäßig eine CPU Utilization von 90 % oder besser erreichen.

Hierzu Scott McNealy, Chairman, President, und CEO der Firma Sun Microsystems:

- **A recent study shows that about 10% of IT costs are hardware, 10% software, and the rest administration and training.**
- **System utilization is around 15%; it should be 80%.**
- **Today, a system administrator can manage between 15 and 30 systems; it should be 500.**

www.cedix.de/VorlesMirror/Band2/DistrCost1.pdf

Das war im Januar 2003. Seitdem hat sich wenig geändert. Ebenfalls von Jonathan Schwartz, Executive Vice President, Sun Software:

- **IT infrastructure utilization rates are at an all-time low. The rapid spread of new services across a multiplicity of distributed systems has resulted in a gross underutilization of hardware.**
- **Management costs and complexity are at an all-time high. The proliferation of individual, low-level software and hardware components that deliver just one service requires too much management.**

www.cedix.de/VorlesMirror/Band2/DistrCost2.pdf

14.3.2 Blades

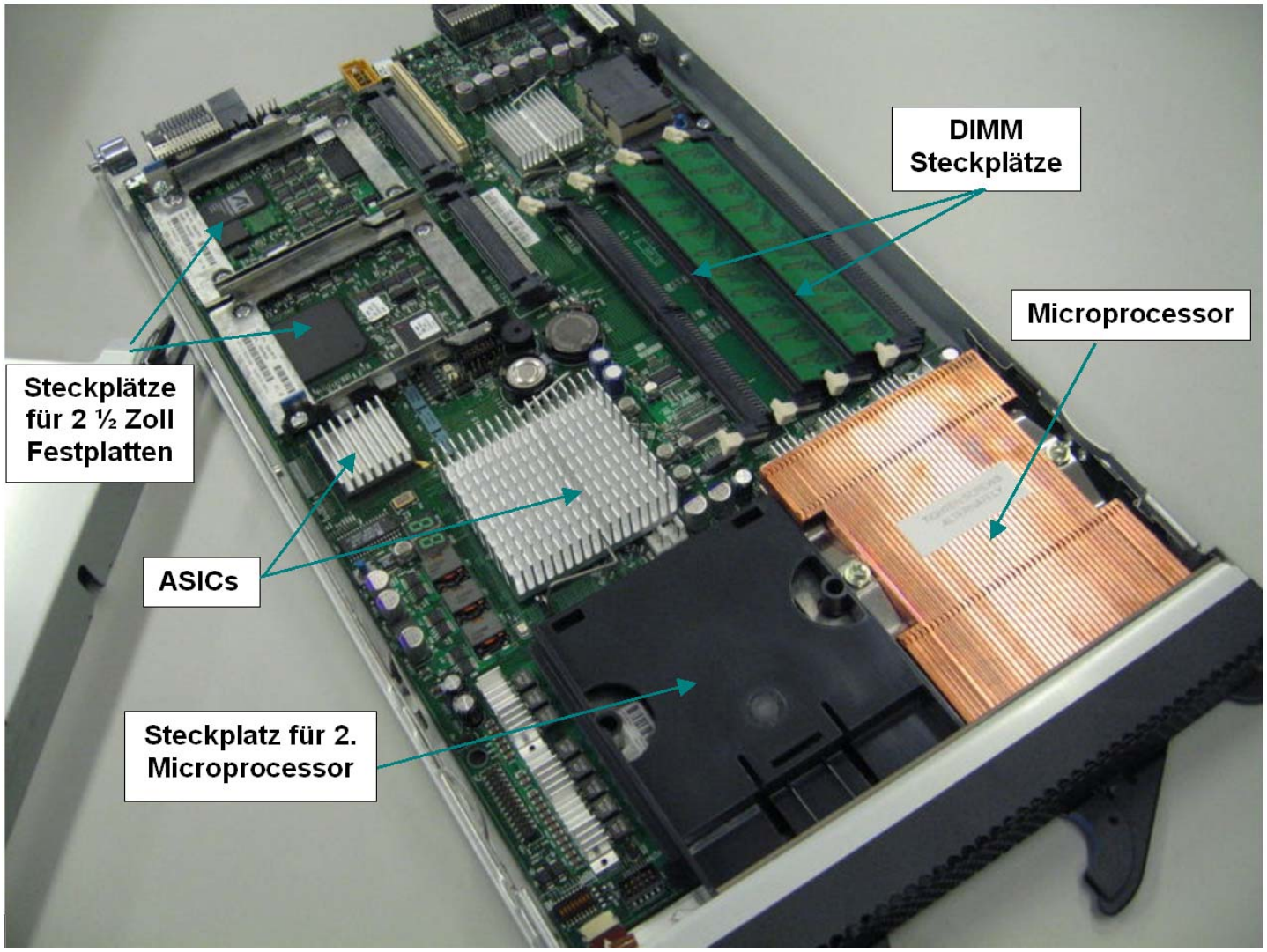


Abb. 14.3.6
IBM Blade für x86 und PowerPC Mikroprozessoren

Distributed Server bestehen in vielen Fällen aus Baugruppen, die als "Blades" bezeichnet werden. Eine Blade ist ein Printed Circuit Board (ähnlich einem PC Mainboard), auf dem sich typischerweise zwei Mikroprozessor Chips befinden, normalerweise mit 4 oder 8 CPU Cores/Chip. Weiterhin enthält eine Blade Steckplätze für Hauptspeicher DIMMs (Dual Inline Memory Module), ASIC Chips für die Integration, für Ethernet und Serial SCSI oder Fibre Channel SCSI Anschlüsse und eventuell Steckplätze für ein bis zwei 2 ½ Zoll Festplatten, auf denen das Betriebssystem untergebracht werden kann.

Blades werden in sehr ähnlicher Ausführung von zahlreichen Herstellern produziert, darunter Dell, HP, IBM und Sun, normalerweise mit einer ganzen Reihe von unterschiedlichen Ausstattungsvarianten. Die Blades werden mit x86 Prozessoren der Firmen Intel (Xeon) und AMD (Opteron) bestückt, oder alternativ mit proprietären Chips der Hersteller (HP Itanium, IBM PowerPC sowie Sun Sparc). Auf den Blades laufen Betriebssysteme wie Windows, Linux, Solaris, HP-UX, IBM AIX und Sun Solaris.

Abb. 14.3.6 zeigt eine IBM Blade.

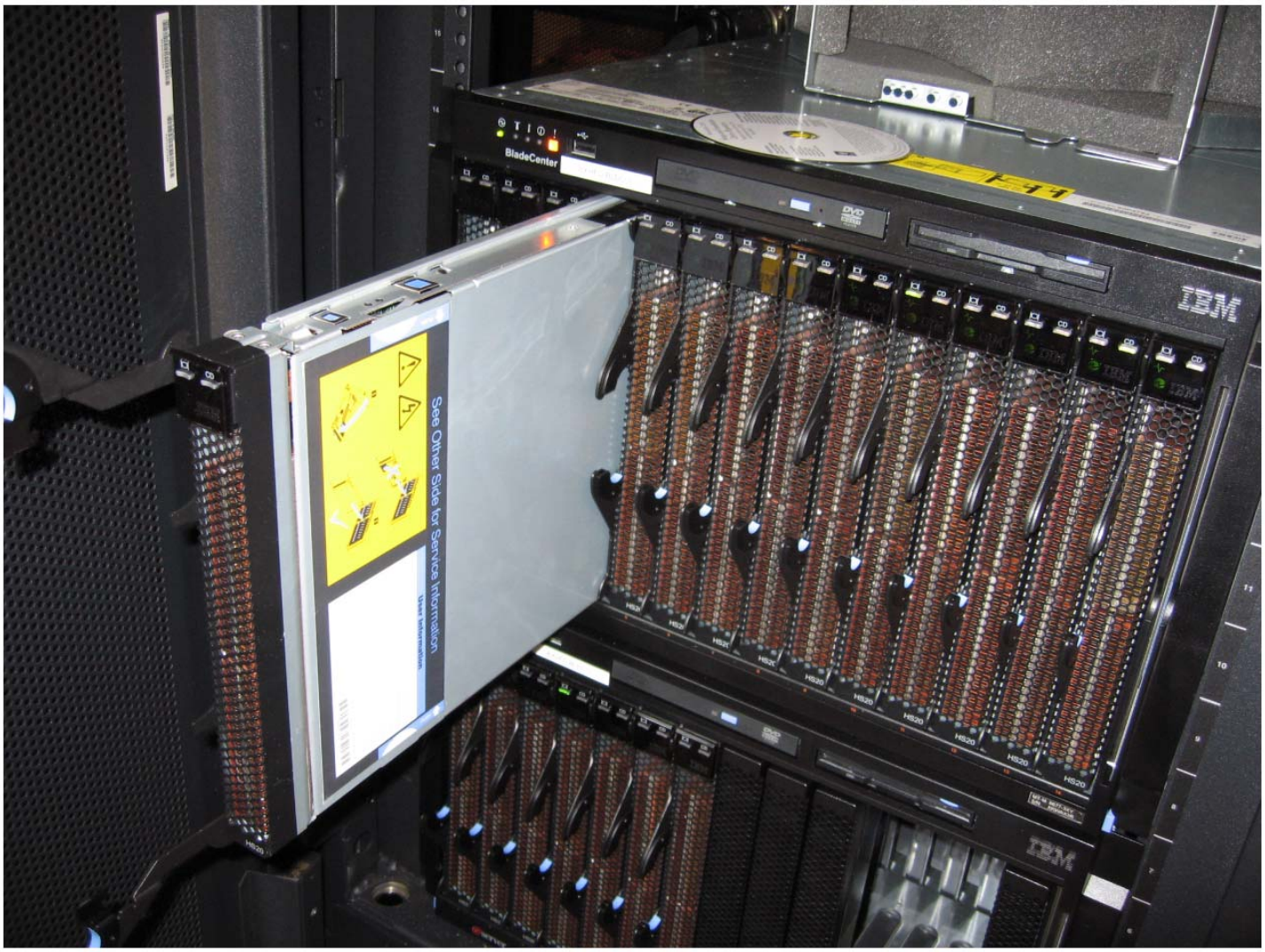


Abb. 14.3.7
IBM Blade Center mit 14 Blades

Blades werden in einem Blade Gehäuse (Blade Enclosure, IBM Bezeichnung „Blade Center“) untergebracht. Eine Blade Enclosure enthält eine Anzahl von Blades, wobei auf jeder Blade typischerweise ein eigenes Betriebssystem läuft. Weiterhin stellt die Blade Enclosure zusätzliche Funktionen wie Stromversorgung, Kühlung, Ethernet Netzwerkfunktionen, Plattenspeicheranschlüsse (häufig über Fibre Channel SCSI, siehe Band 1, Abschnitt 5.2.3) und Management Funktionen zur Verfügung.

Abbildung 14.3.7 zeigt ein IBM Blade Center, welches 14 IBM Blades aufnehmen kann. Es existieren unterschiedliche Blade Center Konfigurationen, die sich z.B. bezüglich der Netzwerk Konfigurationen unterscheiden. Es kann ein recht komplexer Ethernet Switch eingebaut sein, welcher eine direkte interne Verbindung der Blades untereinander, und/oder zu einem Storage Area Netzwerk (SAN) ermöglicht. Mehrere Blade Center können in einem 19 Zoll Rack untergebracht werden.

Blades können einen sehr unterschiedlichen Funktionsumfang haben.

Blade Enclosures werden von zahlreichen Herstellern vertrieben und häufig für distributed Server eingesetzt. Sun Microsystems bietet die "Sun Blade 6000" Blade-Server-Platform an, die sowohl mit Sparc- als auch x86-Blades bestückt werden kann. Auf ihnen laufen die Solaris, Linux und Windows Betriebssysteme. Die Blades können mit jeweils zwei UltraSPARC T1 Chips, zwei Quad-Core-Xeon Chips oder zwei Opteron Chips bestückt werden.

Die Basis stellt ein Blade-Gehäuse dar, das bis zu zehn Blades aufnimmt, bei bis zu vier Gehäusen pro Rack. Ähnliche Produkte sind von den Firmen Cisco, Dell und HP erhältlich.

Blade Center der Firma Sun

Abb. 14.3.8
Sun Blade Center

In großen Unix Konfigurationen wird eine ähnliche Technologie eingesetzt. Die System Boards eines Sun 25K (Abschnitt 11.1.4) oder M9000 Rechners sind eine evolutionäre Erweiterung der Sun Blade 6000 Blades. Ähnliches gilt für die „Cell Boards“ des HP Superdomes. Sie unterscheiden sich durch zusätzliche Funktionen, z.B.:

- **Direkte Verbindung über einen zentralen Switch (ohne Ethernet Protokoll)**
- **NUMA Fähigkeit**
- **Partitionsmöglichkeiten (Harte oder virtuelle Partitionierung)**
- **Zusätzliche unterstützende Hardware für die HP-UX oder Solaris Betriebssysteme**
- **Einrichtungen für eine zentrale Administration**
- **Verbesserte I/O Einrichtungen, höhere I/O Kapazität**
- **Zusätzliche Verbesserungen für Ausfallsicherheit, Zuverlässigkeit und Verfügbarkeit**
- **.....**

14.3.3 System z Blade Center Extension

Die „System z Blade Center Extension“ (zBX) ist eine Gruppierung von einem oder mehreren speziell hierfür optimierten IBM Blade Centern, die über ein sicheres privates Hochleistungsnetzwerk (IEDN), mit einem zEnterprise-Rechner - entweder zEC12, z196 oder z114 – verbunden sind. Eine zBX kann speziell angepasste PowerPC oder x86 Blades aufnehmen, auf denen die AIX, Linux oder Windows Betriebssysteme und deren Anwendungen laufen können. Weiterhin sind Blades mit leistungsfähigen Spezialprozessoren für bestimmte Workloads (z.B. XML oder Krypto Verarbeitung) verfügbar. Sie werden als „Accelerator“ bezeichnet.

Die zBX ist eine neue Infrastruktur für die Erweiterung von System z Servicequalität- und Verwaltungsfunktionen auf integrierte POWER7- und x86 Komponenten. Die Kombination eines oder mehrerer zEC12, z196 oder z114 Rechner mit einer oder mehrerer zBXs wird als zEnterprise-System bezeichnet. Die Administration eines zEnterprise-Systems erfolgt über eine neue Komponente, dem zEnterprise Unified Resource Manager mit System z Wartungsstandards.

Für eine höhere Verfügbarkeit wurde in der zBX auf verschiedenen Ebenen Hardwareredundanz vorgesehen. Dies umfasst die Power-Infrastruktur, Rack-einbaufähige Netzwerkswitches, Netzteile und Switcheinheiten im BladeCenter-Gehäuse, die redundante Verkabelung zu Supportzwecken sowie Datenverbindungen zum zEnterprise Rechner.

Eine zBX besteht aus einem Gehäuse, in dem 1 oder 2 Blade Center , Stromversorgung, Verkabelung in der Form von mehreren Ethernet Switche und weiteren Komponenten untergebracht sind. Die zBX wird in der Regel unmittelbar neben dem zEC12, z196 oder z114 Rechner aufgestellt. Die Datenverbindungen zwischen den beiden Einheiten sind von außen nicht zugreifbar.

Eine zBX kann zunächst Blades mit Standard Microprozessoren aufnehmen, spezifisch x86 und PowerPC. Auf ihnen sind alle unterstützten Betriebssystem lauffähig, besonders Windows und Linux, aber in Zukunft auch andere Betriebssysteme wie z.B. Solaris, Mac OS, Xenix, Unixware, SCO Unix oder FreeBSD.

Alternativ zu Blades mit Standard Microprozessoren sind „Spezial Blades“ verfügbar. Auf Ihnen laufen bestimmte Anwendungen besonders performant dank spezieller Hardware und Software. Ein Beispiel hierfür sind die WebSphere DataPower Appliances wie

- **WebSphere DataPower Integration Appliance XI50**
- **WebSphere DataPower XML Accelerator Appliance XA35**
- **WebSphere DataPower Message-Routing Appliance XM70**
- **WebSphere DataPower XML Security Gateway Appliance XS40**
- **WebSphere DataPower (B2B Business to Business) Gateway Appliance XB60**

http://www.cedix.de/VorlesMirror/Band2/SpecBlade.pdf

Weitere Special Blades werden in Zukunft erwartet. Ein Beispiel ist, besonders Performance-kritische Algorithmen nicht durch einen Mikroprozessor, sondern durch einen ASIC oder ein FPGA verarbeiten zu lassen. Gameframe ist ein weiteres Beispiel (http://www.ti.uni-tuebingen.de/index.php?id=381&L=1).

Abb. 14.3.9
zBX Konfiguration

Eine zBX wird über eine interne, von außen nicht zugängliche Verkabelung direkt an ein Modell zEC12, z196 oder Modell 114 Mainframe angeschlossen.

Es ist eine zentrale Switch Steuerung vorhanden, in mancher Beziehung vergleichbar mit den zentralen Switches der Sun M9000 oder HP Superdome Rechner.

IBM bezeichnet Konfigurationen mit einer zBX als „zEnterprise“.

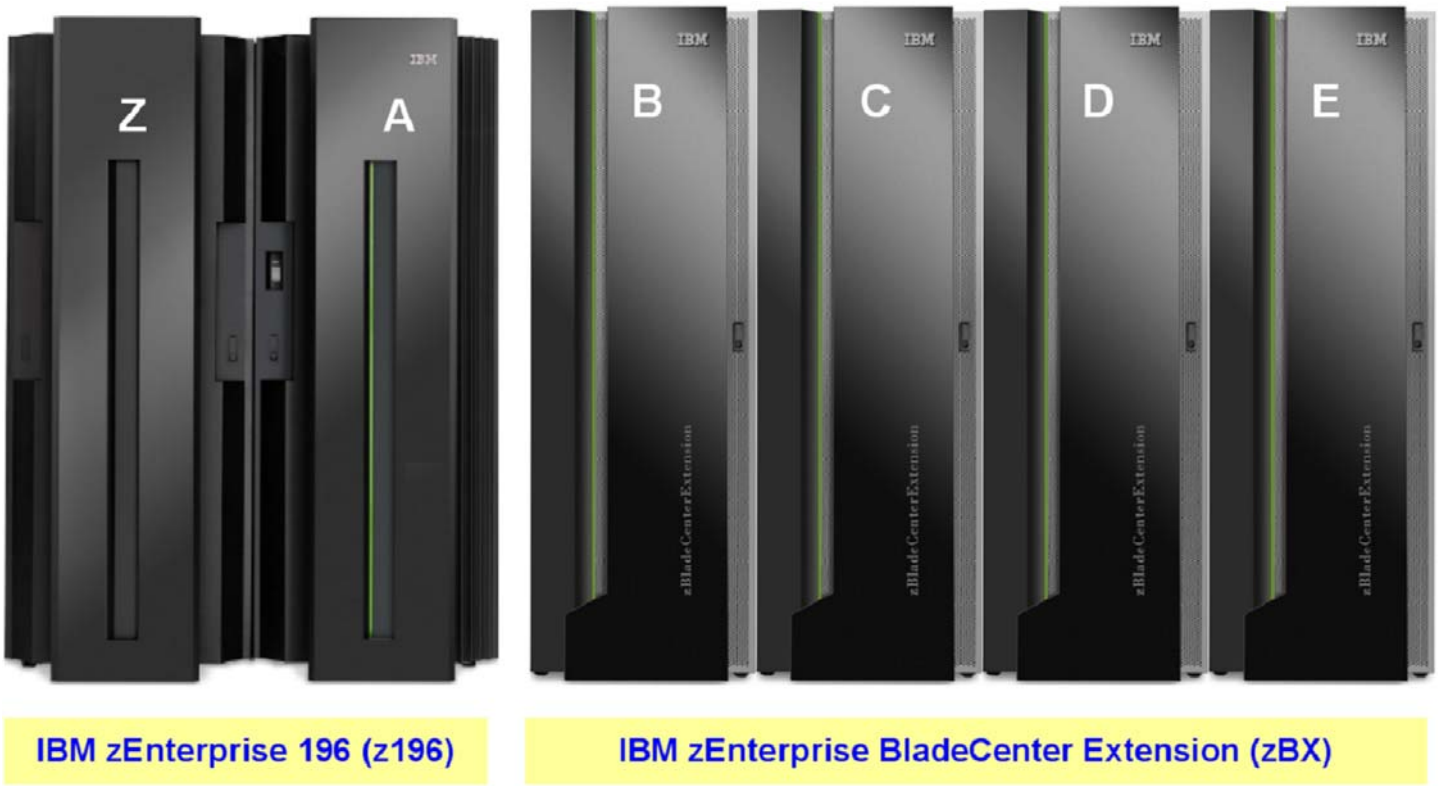


Abb. 14.3.10
Konfiguration mit der maximal zulässigen Anzahl von 4 zBX Einheiten

Es ist möglich, an einen zEnterprise Rechner mehrere (bis zu 4) zBX Einheiten anzuschließen.

Ein Blade Center nimmt bis zu 14 Blades auf und ein zBX Frame entweder 1 oder 2 Blade Center, also insgesamt max. 28 Blades. Eine Maximalkonfiguration mit 4 angeschlossenen zBX kann somit 112 Blades enthalten-

Mit 2 CPU Chips pro Blade und 8 CPU Cores pro CPU Chip sind maximal 1 792 Blade CPU Cores möglich.

14.3.4 Private Netzwerke

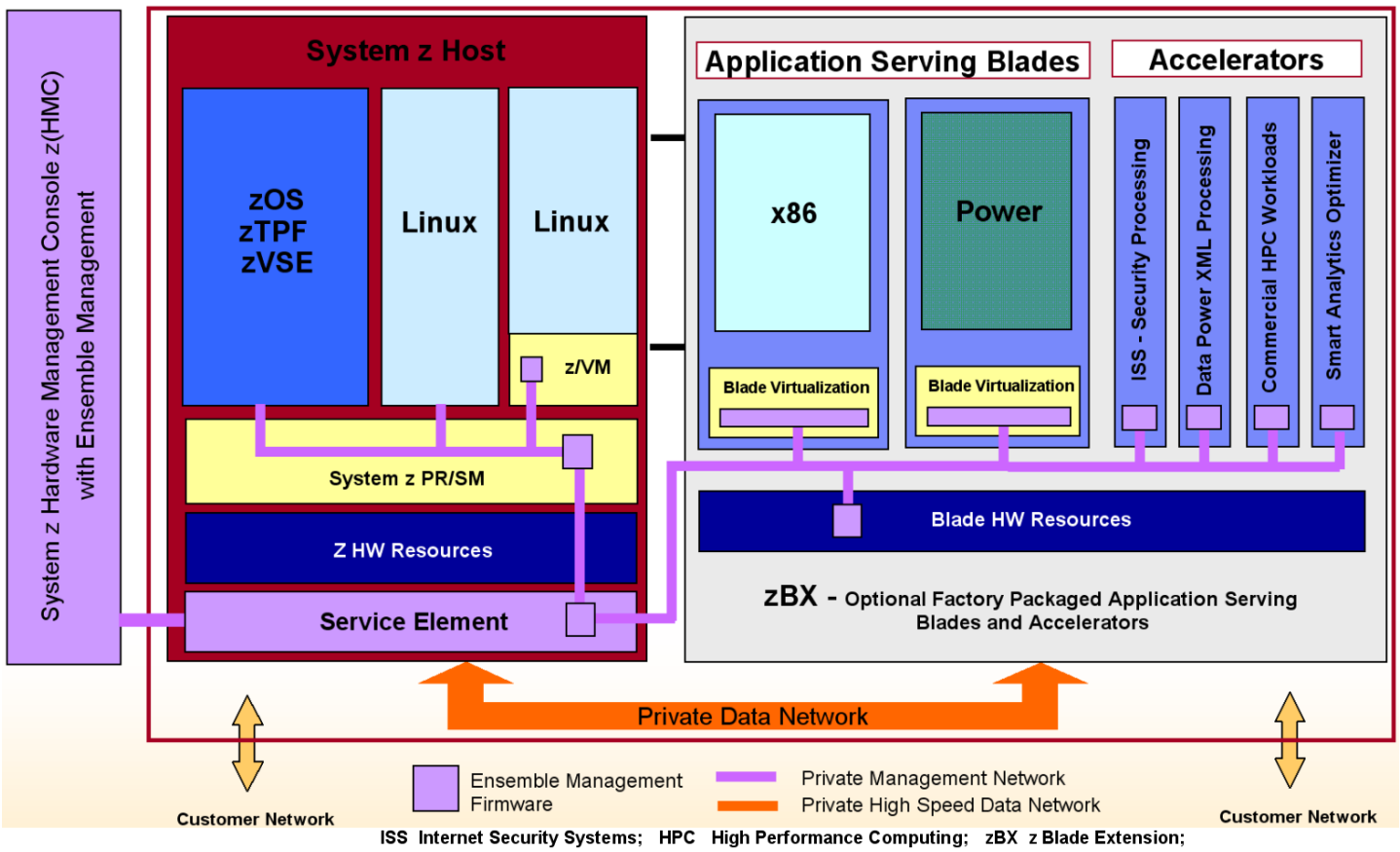


Abb. 14.3.11
Netzwerk Anbindung einer zBX

Die zBX ist sehr eng an den zEnterprise Rechner gekoppelt. Hierzu dienen zwei „Private Networks“. Private bedeutet, dass die Netzwerke nur zwischen der zBX und dem zEnterprise Rechner kommunizieren, und von außen nicht zugänglich sind. Ein Zugang von außen erfolgt nur über den Mainframe Rechner oder eine Blade in der zBX. Bei den beiden Netzwerken handelt es sich einmal um ein (10 Gbit Ethernet) Private Data Network sowie ein (1000BASE-T Ethernet) Private Management Network.

Das Private Data Network wird als „Inter Ensemble Data Network“ (IEDN) bezeichnet. Es wird benutzt, um Daten zwischen den Mainframe Prozessoren und den Blade Prozessoren (bzw. deren Anwendungen) auszutauschen. Hierfür existieren innerhalb des zBX Frames mehrere sehr leistungsfähige Ethernet Switche und der dazugehörige Support.

Das Private Management Network wird benutzt, um die für ein Mainframe bereits vorhandenen Management Funktionen auf die Blades der zBX auszudehnen. Die einzelnen Blades, die Switche und die Stromversorgungen einer zBX sind hierfür mit zusätzlichen Support Prozessoren und/oder Firmware ausgestattet, die eine zentrale Administration ermöglichen.

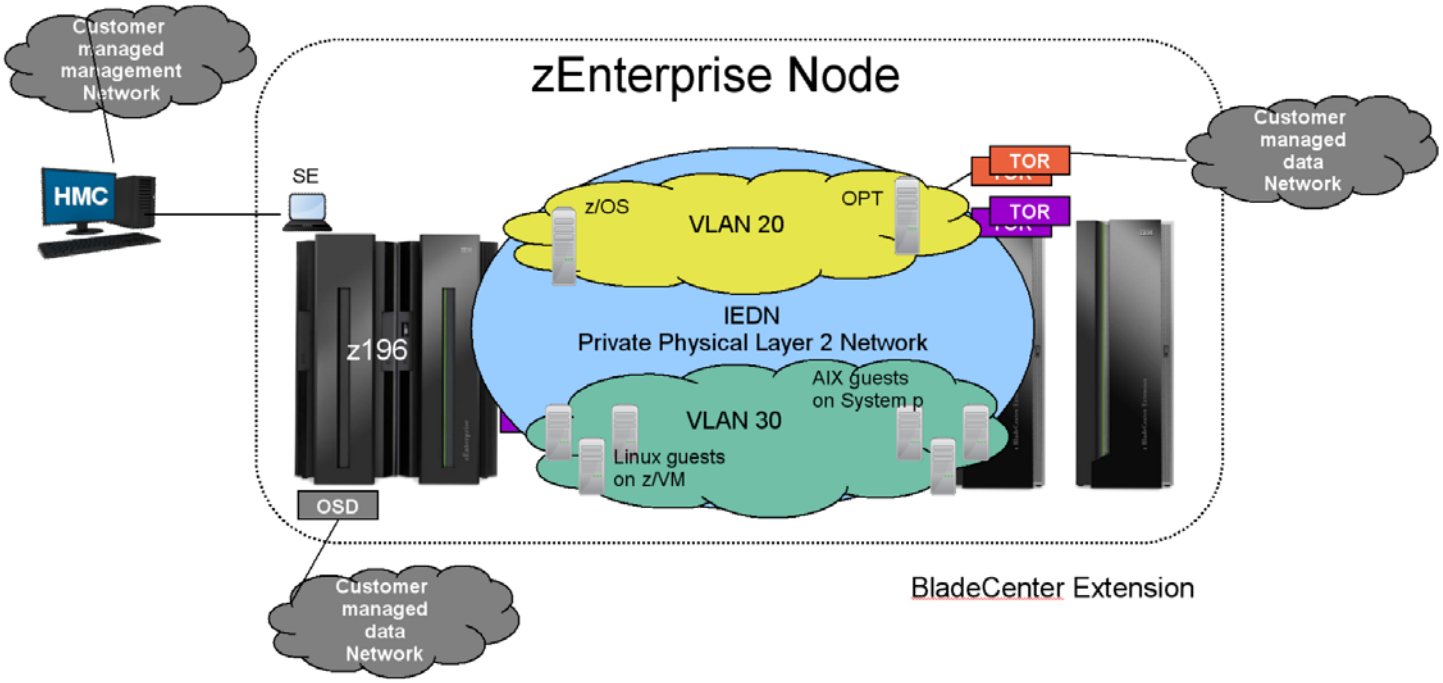


Abb. 14.3.12
zBX VLAN Anbindung

Ein Mainframe-Rechner mit angeschlossener zBX wird als zEnterprise Node bezeichnet. Innerhalb einer zEnterprise Node können mehrere virtuelle Netzwerke (VLAN) konfiguriert werden, welche die Verbindung zwischen den Blades und dem z/OS Rechner herstellen.

Sockets sind ein weit verbreitetes Protokoll in der Schicht 5 des OSI Modells für die Kommunikation zwischen zwei Rechnern. Sockets benutzen TCP/UDP in der Schicht 4 und IP in der Schicht 3.

Die in Abschnitt 12.4.7 erwähnten Hipersockets sind ein mit Sockets kompatibles z/OS Protokoll, welches an Stelle von OSI Schicht 4 und Schicht 3 eine Queue im Hauptspeicher für die Kommunikation zwischen 2 LPARs benutzt. Der Performance Gewinn auf Grund der Vermeidung des Schicht 4 und Schicht 3 Overhead ist signifikant.

HiperSockets Integration mit IEDN bewirkt, dass ein Mainframe über das physische IEDN mit einer zBX unter Vermeidung des Schicht 4 und 3 Overhead kommunizieren kann. Die Verbindung erscheint als ein Single Layer 2 Netzwerk. Über das interne zBX Private Data Network ist eine Hipersocket Kommunikation zwischen Blades und z/OS LPARs möglich. Dies verlängert die Reichweite des HiperSockets Netzwerks außerhalb des Mainframes hin auf die zBX.

OSX ist eine spezielle Version des OSA Ethernet Adapters für eine IEDN Kommunikation zwischen zwei zEnterprise Rechnern.

Die HiperSockets Virtual Switch Bridge kann mit dem Intraensemble Datennetz (IEDN) und OSX Adapter eine Hipersocket Verbindung zu einem anderen Mainframe Rechner durch dessen IEDN OSX-Adapter erstellen, wiederum unter Eliminierung des Schicht 4 und Schicht 3 Overhead.

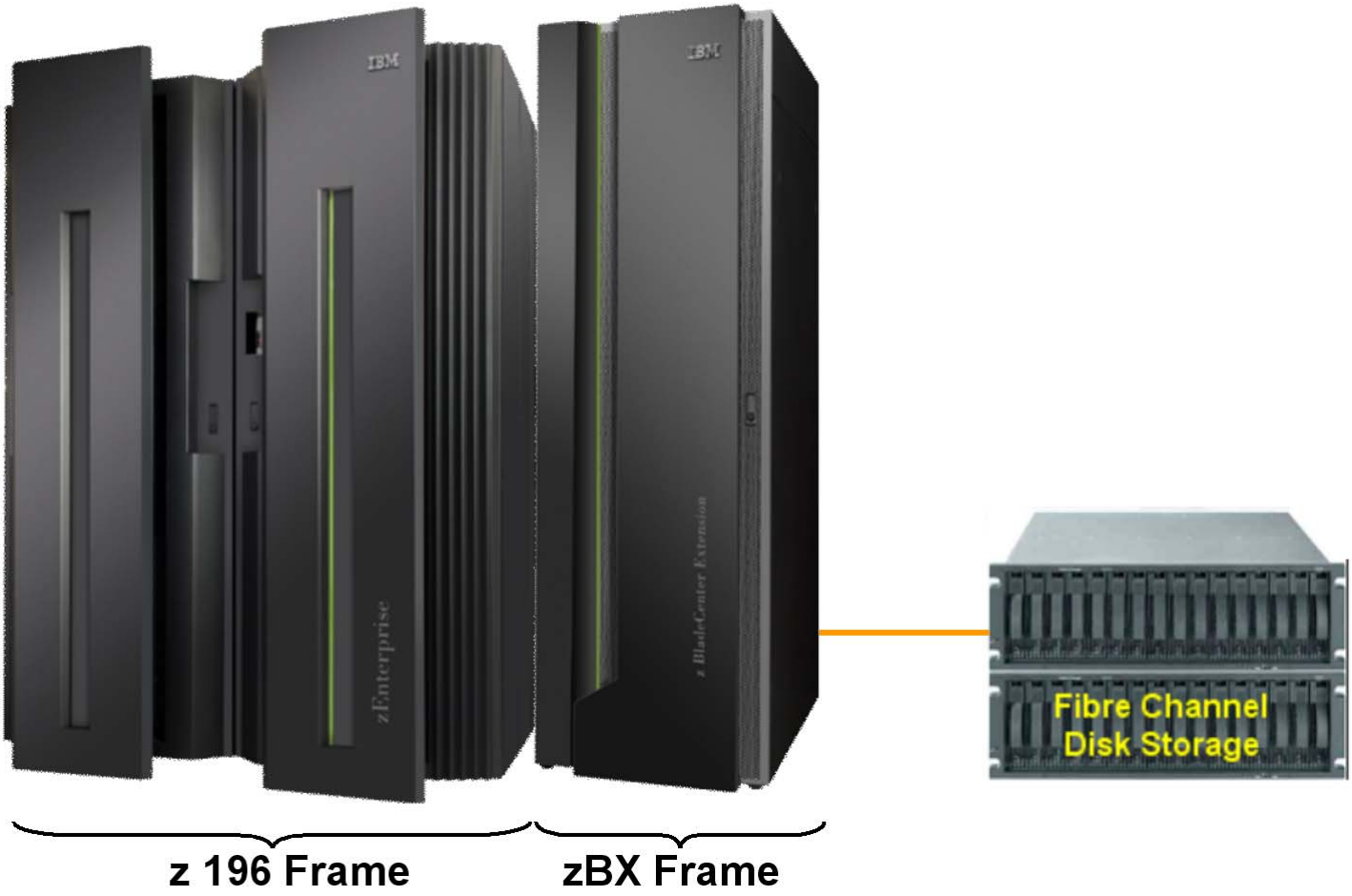


Abb. 14.3.13
Festplattenanschluss an die zBX

Die Blades benötigen eigene Plattenspeicher und entsprechende Anschlüsse. Hierfür ist innerhalb der zBX ein weiteres Fibre Channel Netzwerk und –Switche vorgesehen, über das jede Blade mit externen Fibre Channel SCSI Platten verbunden werden kann (Siehe Band 1 Abschnitt 5.2.3).

14.3.5 Unified Resource Manager

Ein Mainframe Rechner mit angeschlossener zBX wird als zEnterprise Node bezeichnet. Mehrere zEnterprise Nodes bilden ein Enterprise Ensemble. Innerhalb einer zEnterprise Node können mehrere virtuelle Netzwerke konfiguriert werden, welche die Verbindung zwischen den Blades und dem z/OS Rechner herstellen.

Die Hardware Management Konsole (HMC, siehe Abschnitt 14.2.3) spielt eine Schlüsselrolle bei der Administration eines Mainframe Rechners. Bei dem Code in den Flexible Support Prozessoren, den Support Elementen (Think Pad) sowie der HMC handelt es sich um Firmware. Firmware wird immer als Teil der Hardware betrachtet, weil er vom Benutzer oder Administrator nicht einsehbar oder modifizierbar ist.

Beim Anschluss einer zBX wird die Rolle der HMC mittels einer zusätzlichen Funktion drastisch aufgewertet, dem „Unified Resource Manager“. Der Unified Resource Manager übernimmt zusätzlich Management Funktionen der zBX Blades. Dabei werden teilweise bereits vorhandene Mainframe Werkzeuge in angepasster Form verwendet. Die zBX Blades werden auf die gleiche Art administriert und konfiguriert wie die bisher existierenden Komponenten eines Mainframe Servers.

Die Unified Resource Manager Funktionen sind:

- **Operational controls (Operations)**
- **Virtual server lifecycle management (Virtual servers)**
- **Hypervisor management**
- **Energy management (Energy)**
- **Network management (Networks)**
- **Workload Awareness and platform performance management**

Das Management einer heterogenen Umgebung bestehend aus PowerPC und x86 Prozessoren und AIX, Linux und Windows Betriebssystemen soll damit langfristig so vereinfacht werden wie das heute nur in und zwischen Mainframe Komponenten möglich ist. Dies ist allerdings ein langfristiges Ziel, von dem bisher nur Teile verwirklicht worden sind.

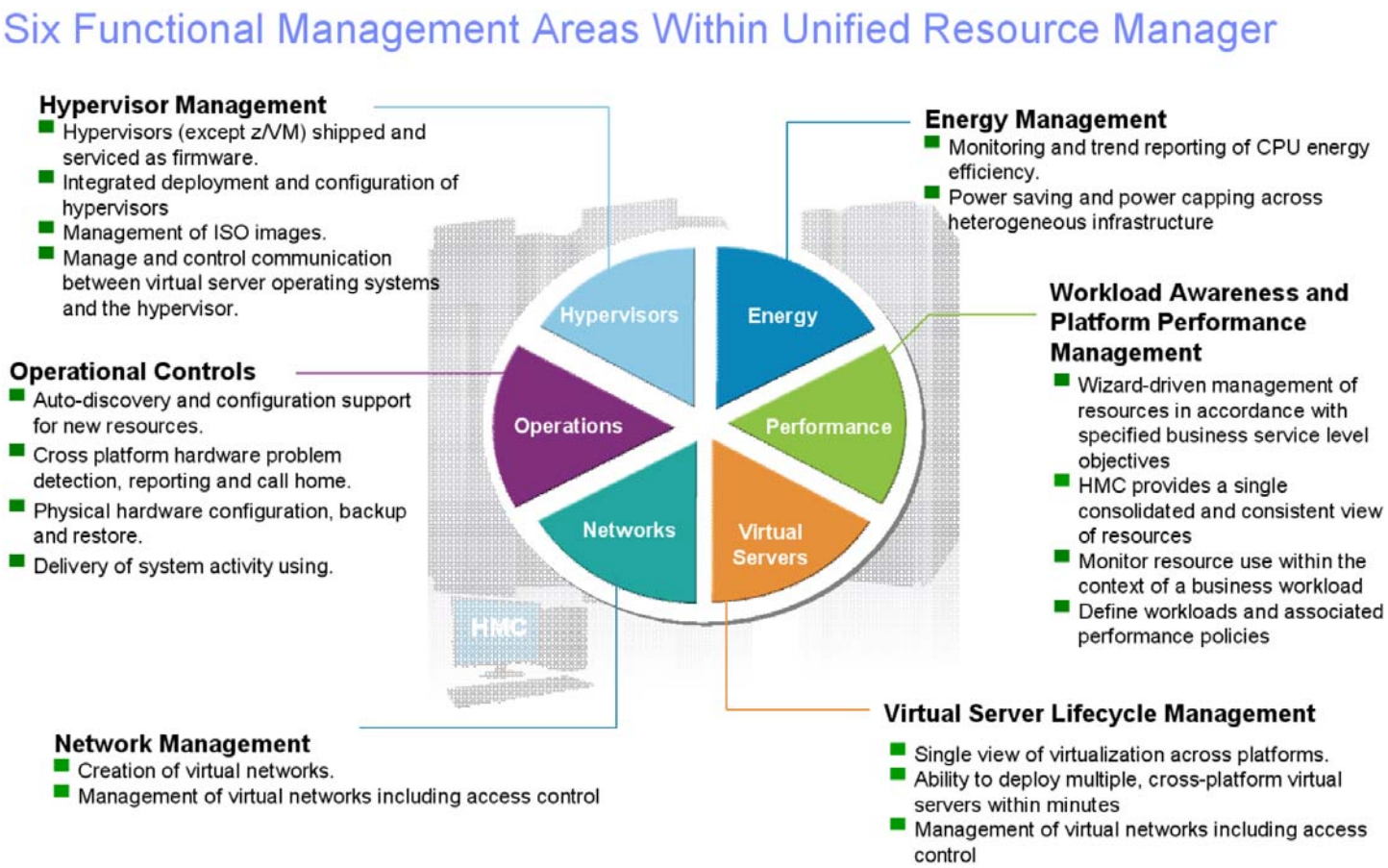


Abb. 14.3.14
Funktionen des Unified Resource Manager

Die zEnterprise Unified Resource Manager läuft auf der Hardware Maintenance Console (HMC). Er kooperiert mit verschiedenen Hypervisor Technologien. Diese sind:

- **PR/SM unterstützt die Definition von logischen Partitionen oder LPARs in den zEnterprise Rechnern, so dass viele logische Partitionen die gleichen physischen Ressourcen gemeinsam nutzen können.**
- **z/VM bietet die Möglichkeit, Betriebssysteme wie Linux, z/OS und andere als Gäste von z/VM auf System z laufen zu lassen, so dass viele virtuelle Maschinen die gleichen physischen Ressourcen gemeinsam nutzen können.**
- **PowerVM Enterprise Edition bietet Virtualisierungsfunktionen für AIX. Es ermöglicht die Erstellung von logischen Partitionen auf den Blades und ermöglicht die gemeinsame Nutzung von Ressourcen zwischen mehreren Betriebssysteme.**
- **Ein in die x86 (System X) Blades integrierter Hypervisor „Kernel based Virtual Machines“ (KVM) bietet eine Virtualisierungslösung für Linux und Windows auf x86-Hardware.**

Die Links in die verschiedenen Hypervisor und indirekte Verbindungen zu z / OS, falls erforderlich, erzeugen eine sehr heterogenen Umgebung. zBX und Unified Resource Manager bieten als Verbesserung ein Integriertes Resource Monitoring, Workload Management, Image Management, Availability Management, Failure Management und Energie Management an. Spezifisch ist es die Aufgabe des Unified Resource Managers, diese unterschiedlichen Virtualisierungslösungen zu konsolidieren. Hierfür existiert eine Unterstützung für „Federated Hypervisors“: PR/SM, z/VM, PowerVM und KVM , mit einem einheitlichem Satz von System Management Policies. Dies ist z.B. ein wichtiger Schritt für den Aufbau einer Cloud-Infrastruktur.

Trotzdem bleibt die Komplexität groß. Wir können an dieser Stelle weitere technologische Entwicklungen erwarten, z.B.:

- **Integrationserweiterung der Blades für neue Architektur Plattformen, z.B. GPU, FPGA, ARM, IPad, Android.**
- **Neue Special-Engines, z.B. mit Algorithmen die in FPGAs oder ASICs ausgeführt werden.**
- **Entwicklungswerkzeuge für neue Architektur Plattformen.**
- **Ausdehnung des z/OS WLM auf zBX Blades.**
- **zBX Coupling Facility.**
- **zBX GDPS.**
- **Shared Memory zwischen Mainframe und zBX, z.B. für Hipersockets.**

14.3.6 Ensemble

Abb. 14.3.15
Ensemble mit drei Nodes

Für Administrationszwecke ist es möglich, mehrere z196 oder zEC12 Server zu einem als „Ensemble“ bezeichneten Verbund zusammenzuschließen. Ein einziger zEnterprise Server mit angeschlossener zBX (1 – 4 Frames) wird hierbei als „Node“ bezeichnet. Ein Ensemble kann aus bis zu 8 Nodes bestehen.

In Abb. 14.3.15 sind drei Nodes (drei z196 Server, teilweise mit angeschlossenen zBXs,) zu einem Ensemble zusammengeschlossen. Eine einzige HMC mit installierter Unified Resource Manager Firmware ist in der Lage, alle angeschlossenen zEnterprise Server einschließlich ihrer lokal angeschlossenen zBXs zu administrieren. Aus Gründen der Zuverlässigkeit ist wie immer eine zweite HMC als Backup vorhanden (Primary und Alternate HMC).

Bitte beachten: Ein Ensemble ist eine administrative Struktur. Unabhängig davon können die zEnterprise Rechner des Ensembles einen oder mehrere Sysplexe oder GDPS implementieren. Dies wird wohl eher die Regel als die Ausnahme sein.

14.3.7 zBX Vorteile

Zuverlässigkeit, Verfügbarkeit und Wartungsfreundlichkeit (RAS) reduziert Ausfallzeiten. zBX ist ein Blade-Center, das für Mainframe-Konnektivität und RAS Leistung ausgelegt ist.

Alle Server laufen virtualisiert unter Kontrolle des Unified Resource Manager.

Die Konsolidierung verteilter Server auf entweder zBX oder zLinux virtuelle Server eliminiert mehr als 90 Prozent der Netzwerk-Infrastruktur-Komponenten (Kabel, Switches und Router). Die Verkabelung zwischen dem zBX und dem Mainframe kann auf wenige Netzwerk Komponenten reduziert werden. zBX konsolidiert System p, System x und Spezial Geräte in einem einzigen integrierten Chassis mit gemeinsamen Zugang zur Stromversorgung und Netzwerken. Dies vereinfacht das Management von Stromversorgung, Kühlung, Netzwerken und Stellfläche, bei deutlicher Senkung der Betriebskosten.

Verbesserte Sicherheit eliminiert Firewall Kosten. Die Konsolidierung verteilter Server in eine zBX verlagert Server-zu-Server-Datenverkehr in VLANs auf dem privaten IEDN und weg von Attacken ausgesetzten externen Netzwerken. Dies kann die Notwendigkeit für Firewalls zwischen zBX Servern sowie zwischen zBX Servern und dem Mainframe eliminieren.

Automation ersetzt viele manuelle Prozesse in fünf kritischen Management-Bereichen: Assets, Deployment, Capacity/Performance, Security sowie Change Management für z/OS, AIX, Linux und Windows Server.

In Zukunft werden wir weitere Verbesserungen in Richtung einer einheitlichen Management Struktur sehen, um all dies zu steuern.

14.4 Netezza

14.4.1 Data Warehouse

Seit vielen Jahren beobachten wir ein ständiges Wachstum der in unseren Computern gespeicherten Daten. Für die Führungskräfte eines Unternehmens ist die Nutzung dieser Daten für ihre Entscheidungsprozesse sehr wichtig.

Business Intelligence Systeme stellen die Informationen für die Entscheidungsprozesse bereit.

Aus technischen Gründen erweist es sich als sinnvoll, ein Business Intelligence System von den datenliefernden operativen Systemen (z.B. einer CICS – DB2 Kombination) zu entkoppeln und auf einer separaten Plattform, als „Data Warehouse“ bezeichnet, zu betreiben. Eine Entkopplung führt einerseits zu einer Entlastung der operativen Systeme und eröffnet andererseits die Option, das analyseorientierte Business Intelligence System auf die Belange von Auswertungen und Berichten hin zu optimieren.

Abb. 14.4.1
Extraktion, Transformation, Laden Prozess

Ein spezieller „Extraktion, Transformation, Laden“ (ETL) Prozess kopiert ständig die operativen Daten in die Data Warehouse Datenbank.

Im Idealfall soll eine derartige Datenbasis unternehmensweit ausgerichtet sein und das Informationsbedürfnis verschiedenster Anwendergruppen abdecken.

14.4.2 Data Mining

Unter Data-Mining – deutsch etwa: „aus einem Datenberg etwas Wertvolles extrahieren“ – versteht man die systematische Anwendung statistischer Methoden auf einen Datenbestand mit dem Ziel, bisher unbekannte Zusammenhänge zu erkennen. Hierbei geht es auch um die Verarbeitung sehr großer Datenbestände (die nicht mehr manuell verarbeitet werden könnten), wofür effiziente Methoden benötigt werden, deren Zeitkomplexität sie für solche Datenmengen geeignet macht.

Data-Mining benutzt Datenanalyse- und Entdeckungsalgorithmen, die unter akzeptablen Effizienzbegrenzungen eine spezielle Auflistung von Mustern (oder Modellen) der Daten liefern. Das Ziel ist die Extraktion von Wissen, das

- **gültig (im statistischen Sinne),**
- **bisher unbekannt und**
- **potentiell nützlich ist,**

„zur Bestimmung bestimmter Regelmäßigkeiten, Gesetzmäßigkeiten und verborgener Zusammenhänge“.

Dies sind einige typische Data-Mining Anwendungen:

Im Finanzsektor: Rechnungsprüfung zur Betrugserkennung

Im Marketing:

- **Marktsegmentierung, beispielsweise Kunden in Bezug auf ähnliches Kaufverhalten bzw. Interessen für gezielte Werbemaßnahmen identifizieren**
- **Warenkorbanalyse zur Preisoptimierung und Produktplatzierung im Supermarkt**
- **Zielgruppen-Auswahl für Werbekampagnen**
- **Kundenprofil-Erstellung zum Management von Kundenbeziehungen in Customer-Relationship-Management Systemen**

Im Internet:

- **Angriffserkennung (Security Attack)**
- **Empfehlungsdienste für Produkte wie beispielsweise Filme und Musik**
- **Netzwerkanalyse in sozialen Netzwerken**
- **Web-Usage-Mining um das Nutzerverhalten zu analysieren**
- **Textmining zur Analyse von großen Textbeständen**

Im Gesundheitswesen:

- **Pharmakovigilanz (Arzneimittelüberwachung nach Marktzulassung im Hinblick auf unbekannte unerwünschte Ereignisse oder Nebenwirkungen)**
- **Das BKK InfoNet des Bundesverbandes der Betriebskrankenkassen (BKK) bietet Kennzahlen und Analysen für Vertragspolitik und Versorgungsmanagement. Die Leistungs-, DMP- und Statistikdaten stehen Krankenkassen und Landesverbänden im Rahmen ihrer Zugriffsrechte zur freien Auswertung zur Verfügung.**

Dies sind zwei Beispiele:

Beispiel Wal-Mart

Wal-Mart (www.wal-mart.com) ist der Marktführer im amerikanischen Einzelhandel, unterhält ein unternehmensweites Data Warehouse, Größe: ca. 300 TB (2003), mit täglich bis zu 20.000 Data Warehouse Anfragen. Jede Eingabe in einer Registrierkasse wird als Transaktion gespeichert. Es erfolgt eine tägliche Auswertung von

- **Artikelumsätzen, Lagerbestand,:**
- **Warenkorbanalyse, Kundenverhalten, ...**
- **Überprüfung des Warensortiments zur Erkennung von Ladenhütern oder Verkaufsschlagern**
- **Standortanalyse zur Einschätzung der Rentabilität von Niederlassungen**
- **Untersuchung der Wirksamkeit von Marketing-Aktionen**
- **Auswertung von Kundenbefragungen, Reklamationen bzgl. bestimmter Produkte etc.**
- **Analyse des Lagerbestandes**
- **Warenkorbanalyse mit Hilfe der Kassenbons**

Beispiel einer Anfrage

Welche Umsatzsteigerungen sind vom 1. – 10. Oktober in den Abteilungen Kosmetik, Elektro und Haushaltswaren in Birmingham, Alabama, angefallen, nachdem wir in Huntsville, Alabama, DVDs um 20 % billiger angeboten haben. ?

Beispiel Otto Versand

Der Versandhändler Otto verbessert mithilfe einer Spezialsoftware seine Bedarfsplanung für das gesamte Sortiment. Das Unternehmen füttert seine Software pro Woche mit 300 Millionen Datensätzen – und erstellt übers Jahr eine Milliarde Prognosen, wie sich der Absatz einzelner Artikel in den folgenden Tagen und Wochen entwickeln wird. Nach Konzernangaben ordert Otto durchschnittlich 30 Prozent weniger Ware als zuvor und hat dadurch deutlich weniger überschüssige Ware auf Lager. Von Einsparungen in zweistelliger Millionenhöhe ist die Rede.

Die wichtigsten Data Mining Prozesse sind ETL, Business Intelligence sowie Advanced Analytics.

ETL

Die Abkürzung ETL steht für „Extraktion, Transformation, Laden“. Dies ist ein Prozess, der Daten aus mehreren Datenquellen in einer Data Warehouse Zieldatenbank vereinigt.

Business Intelligence (BI)

Der englische Ausdruck „Intelligence“ (von lat. intellegere „verstehen“ und legere „lesen, wählen“) bezeichnet die aus dem Sammeln und dem Aufbereiten erworbener Informationen gewonnenen Erkenntnisse.

Der Begriff Business Intelligence (BI) bezeichnet Verfahren und Prozesse zur systematischen Analyse (Sammlung, Auswertung und Darstellung) von Daten in elektronischer Form. Ziel ist die Gewinnung von Erkenntnissen, die in Hinsicht auf die Unternehmensziele bessere operative oder strategische Entscheidungen ermöglichen. Dies geschieht mit Hilfe analytischer Konzepte sowie entsprechender Software bzw. IT-Systeme, die Daten über das eigene Unternehmen, die Mitbewerber oder die Marktentwicklung im Hinblick auf den gewünschten Erkenntnisgewinn auswerten. Mit den gewonnenen Erkenntnissen können Unternehmen ihre Geschäftsabläufe, sowie Kunden- und Lieferantenbeziehungen profitabler machen, Kosten senken, Risiken minimieren und die Wertschöpfung vergrößern.

Advanced Analytics

Der Begriff "Advanced Analytics" ist eine Erweiterung der Business Intelligence. Das Ziel ist jedoch weniger, durch die Analysen der Ist-Situation bzw. Vergangenheit aktuelle Probleme aufzuzeigen, sondern stärker auf Prognosen und Vorhersagen der zukünftigen Entwicklung zu setzen. Unterbereiche sind Predictive Analytics, Simulation und Optimierung

Predictive Analytics

- **Was wird passieren, wenn unsere Kunden genauso einkaufen wie in der Vergangenheit?**
- **Was passiert mit unseren Vertriebsergebnissen, wenn sich die derzeitigen Trends fortsetzen?**

Simulation

- **Wenn wir ein neues Produkt einführen, wie werden unsere Konkurrenten wahrscheinlich reagieren?**
- **Wenn wir unsere Preispolitik ändern, was ist der Einfluss auf unsere Kundenbindung und Marktdurchdringung?**

Optimierung

- **Wie erreichen wir die besten Beladungspläne für unsere LKWs?**
- **In welche Drogen Forschungsprojekte sollen wir investieren, um unsere Gewinne in Anbetracht unserer auslaufenden Drogen Patente zu maximieren?**

14.4.3 Data Sharing Alternativen

Data Mining kann mittels SQL Anfragen an die gleichen relationale Datenbanken erfolgen, die auch für die Transaktionsverarbeitung eingesetzt werden. Bei großen Datenbeständen können aber selbst einfache Anfragen Stunden an Verarbeitungszeit in Anspruch nehmen. In der Praxis dupliziert man die Datenbestände in einer getrennten Data Warehouse Datenbank. Das Data Warehouse verfügt über eigene CPUs, welche Data Warehouse Queries bearbeiten können. Eine Data Warehouse Datenbank ist typischerweise eine relationale Datenbank, und Abfragen erfolgen als SQL Queries. Data Mining Erweiterungen, englisch Data Mining Extensions (DMX), sind proprietäre Erweiterungen des SQL-Standards um die Fähigkeit zu verbessern, mit Data-Mining Modellen zu arbeiten.

Ein Transaktionsserver verarbeitet in der Regel mehrere Transaktionen parallel auf mehren CPUs. Ebenso sind zahlreiche Plattenspeicher vorhanden. Diese sind nach dem „Shared Disk“ Verfahren an die CPUs angeschlossen, weil eine Transaktion in der Regel auf den ganzen Datenbestand zugreifen möchte. Bei einem Data Warehouse ist es möglich, die Datenbank auf einzelnen Festplatten zu partitionieren, jeder Festplatte eine getrennte CPU zuzuordnen, und eine Abfrage auf zahlreichen CPUs parallel gegen die partitionierte Datenbank durchzuführen. Diese Architektur wird als „Shared Nothing“ bezeichnet.

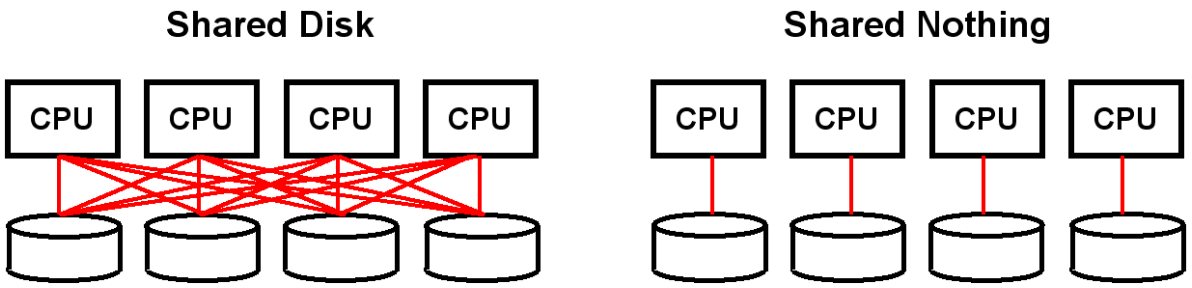


Abb. 14.4.2
Shared Nothing erlaubt nur Zugriffe auf zugeordnete Festplatten

Eine Shared-Nothing System gilt als besser als ein Shared Disk System für Anwendungen, die eine Reihe von Datenbanken parallel durchsuchen sollen. Dazu gehören Data Warehouse Anwendungen.

Die Shared-Nothing Architektur beschreibt eine Distributed Computing Architektur, bei der jeder Knoten unabhängig und eigenständig seine Aufgaben mit seinem eigenen Prozessor und den zugeordneten Speicherkomponenten wie Festplatte und Hauptspeicher erfüllen kann und kein bestimmter, einzelner Knoten für die Verbindung zu einer Datenbank notwendig ist. Die Knoten sind über ein LAN- oder WAN-Netzwerk miteinander verbunden. Jeder Knoten verfügt darüber hinaus über eine Kopie des Datenbank-Management-Systems. Ein Knoten kann Aufgaben an einen anderen nicht ausgelasteten Knoten weitergeben.

Shared Nothing ist auf Grund seiner Skalierbarkeit beliebt für Webanwendungen oder parallele Datenbanksysteme. Wie bei Google gezeigt werden konnte (Abschnitt 17.3.2), ist ein Shared Nothing System nahezu unbegrenzt durch Ergänzung zusätzlicher Knoten in Form preiswerter Computer ausbaufähig, weil kein einzelnes Netzwerkelement existiert, dessen begrenzte Leistung die Geschwindigkeit des gesamten Systems vermindert.

Die Firma Teradata ist ein bedeutender Data Warehouse Hersteller, der seit 1979 ein Shared nothing Relationales Datenbankmanagementsystem vertreibt. Die wichtigsten Mitbewerber im Data Warehouse Markt sind

- **IBM mit dem „PureData System for Analytics“ (Netezza) Produkt,**
- **Oracle mit dem „EXAdata“ Produkt und**
- **SAP mit dem „HANA“ In-Memory Datenbank Produkt.**

Apache Hadoop ist ein freies, in Java geschriebenes Framework für skalierbare, verteilt arbeitende Software. Es ermöglicht es, intensive Rechenprozesse mit großen Datenmengen (Petabyte Bereich) auf Computerclustern durchzuführen. Das IBM-Produkt InfoSphere BigInsights basiert auf Hadoop.

Problem: Während der Shared Nothing Ansatz eine deutliche Leistungssteigerung bringt, wachsen die Anforderungen so schnell, dass die Verarbeitungsdauer von Data Warehouse Anfragen ein echtes Problem darstellt.

14.4.4 PureData System for Analytics (Netezza)

Das IBM PureData System for Analytics (alte Bezeichnung: Netezza, englische Aussprache: „netiesa“) ist eine Data Warehouse Appliance, d.h. ein kombiniertes System aus Hardware und speziell darauf abgestimmter Software, welches sich von alternativen Produkten wie z. B. Teradata oder Oracle Exadata durch zwei Merkmale unterscheidet:

- **Ein PureData System for Analytics ist zwar unabhängig von zBX. Der „IBM DB2 Analytics Accelerator for z/OS“ (IDAA) ermöglicht aber eine ähnlich enge Integration in z/OS (spezifisch DB2) wie bei den zBX Appliances.**
- **Mittels massiver Parallelisierung von Berechnungen und dem Einsatz von FPGAs (Field Programmable Gate Arrays) ist eine deutlich höhere Verarbeitungsgeschwindigkeit möglich.**

Ein Field Programmable Gate Array (FPGA) ist ein Integrierter Schaltkreis (IC) der Digitaltechnik, in den eine logische Schaltung programmiert werden kann. Anders als bei der Programmierung von Computern oder Steuerungen bezieht sich hier der Begriff Programm nur in zweiter Linie auf die Vorgabe zeitlicher Abläufe im Baustein, sondern vor allem auf die Definition von dessen Funktionsstruktur. Durch die Programmierung von Strukturvorschriften wird zunächst die grundlegende Funktionsweise einzelner universeller Blöcke im FPGA und deren Verschaltung untereinander festgelegt. Durch die spezifische Konfiguration interner Strukturen können in einem FPGA verschiedene Schaltungen realisiert werden. Diese reichen von Schaltungen geringer Komplexität, wie z. B. einfacher Synchronzähler, bis zu hochkomplexen Schaltungen, wie Mikroprozessoren. FPGAs werden in allen Bereichen der Digitaltechnik eingesetzt, vor allem aber dort, wo es auf schnelle Signalverarbeitung und flexible Änderung der Schaltung ankommt, um beispielsweise nachträgliche Verbesserungen an den implementierten Funktionen vornehmen zu können, ohne dabei direkt die physische Hardware ändern zu müssen.

Mit der Einführung der FPGAs wurden kompakte, anwenderspezifische Schaltungen in geringen Stückzahlen ermöglicht. Heutzutage gestatten sie z. B. die preiswerte und flexible Fertigung komplexer Systeme wie Mobilfunk-Basisstationen als Alternative zur teureren Auftragsfertigung durch Halbleiterhersteller.

Die modernste (2013) PureData System for Analytics (PDSA) Implementierung wird als Modell N2001 bezeichnet.

http://de.wikipedia.org/wiki/Field_Programmable_Gate_Array enthält eine FPGA-Übersicht.

14.4.5 S-Blades und FPGA Konfiguration

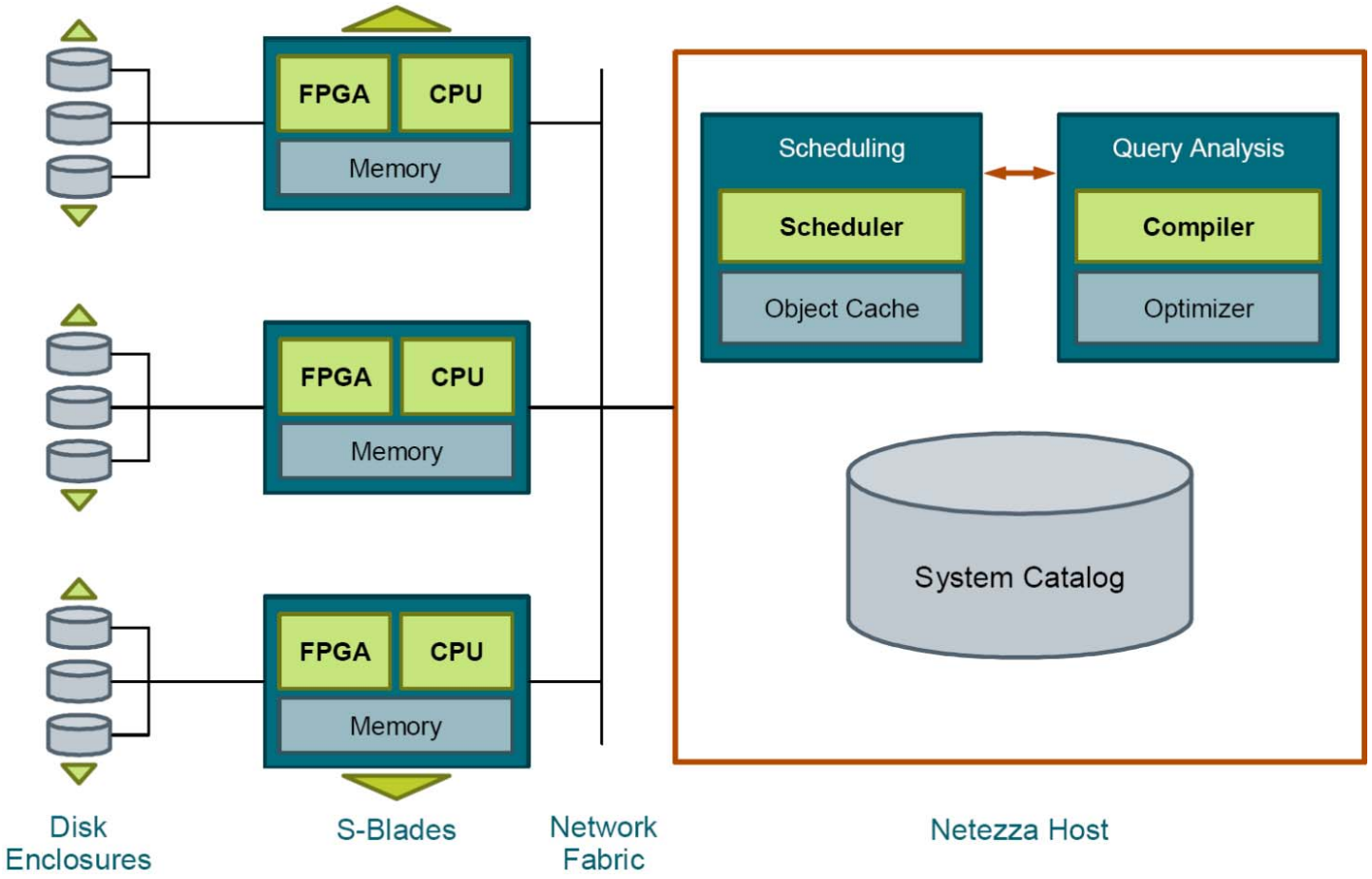


Abb. 14.4.3
S-Blades und der Netezza Host

Die Basiskonfiguration eines PureData System for Analytics (PDSA) besteht aus 288 SAS2 600-GB Festplatten (bei vierfacher Komprimierung stehen dem Benutzer 173 TByte zur Verfügung), 4 sogenannten S-Blades mit zusammen 64 CPU-Cores, 64 FPGA-Kernen, einem internen Netzwerk sowie einem internen Steuerrechner, der als Host bezeichnet wird.

Der Host ist ein selbständiger Hochleistungs-Linux-Server, der aus Verfügbarkeitsgründen doppelt vorhanden ist. Er kompiliert SQL-Abfragen von Advanced Analytics, Business Intelligence oder ETL-Anwendungen in ausführbare Code-Segmente und verteilt diese zur Ausführung auf die S-Blades.

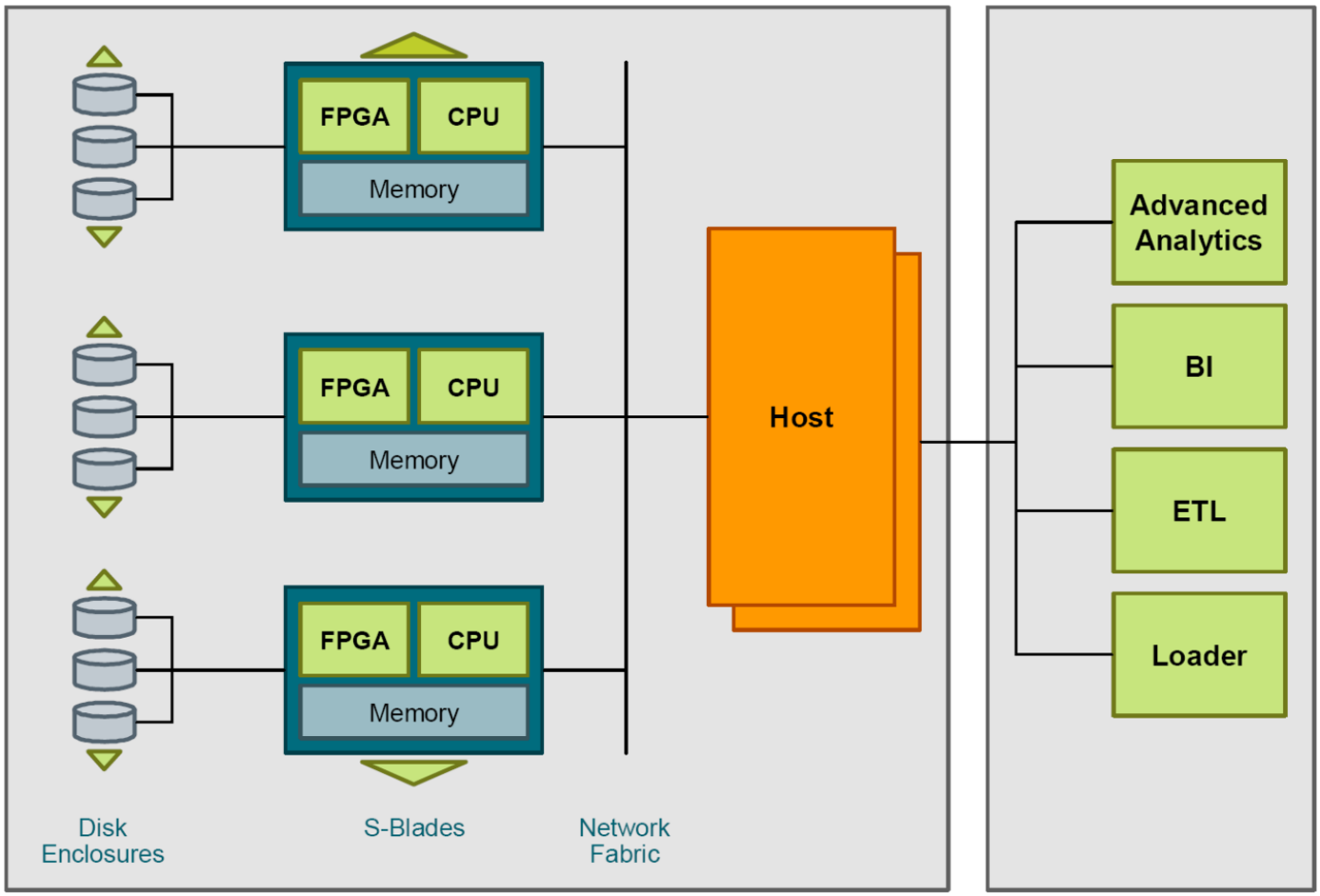


Abb. 14.4.4
Die Advanced Analytics, BI und ETL Anwendungen laufen auf einem getrennten Rechner

Einmal verteilt, führen die S-Blades die Verarbeitung durch. Wenn die Verarbeitung abgeschlossen ist, aggregiert der Host die Ergebnisse. Eine S-Blade ist ein kompletter Computer mit einem Prozessor, Hauptspeicher, Festplattenspeicher und einem Kernstück, dem „FPGA (Field Programmable Gate Array)“. Ein S-Blade enthält 16 CPU-Kerne und 16 FPGA-Kerne, wobei jedem CPU-Kern ein FPGA-Kern zugeordnet ist. Jedem FPGA-Kern sind darüber hinaus drei Festplatten zugeordnet.

Der „Loader“ bewirkt das Kopieren der Unternehmensdaten auf die Festplatten.

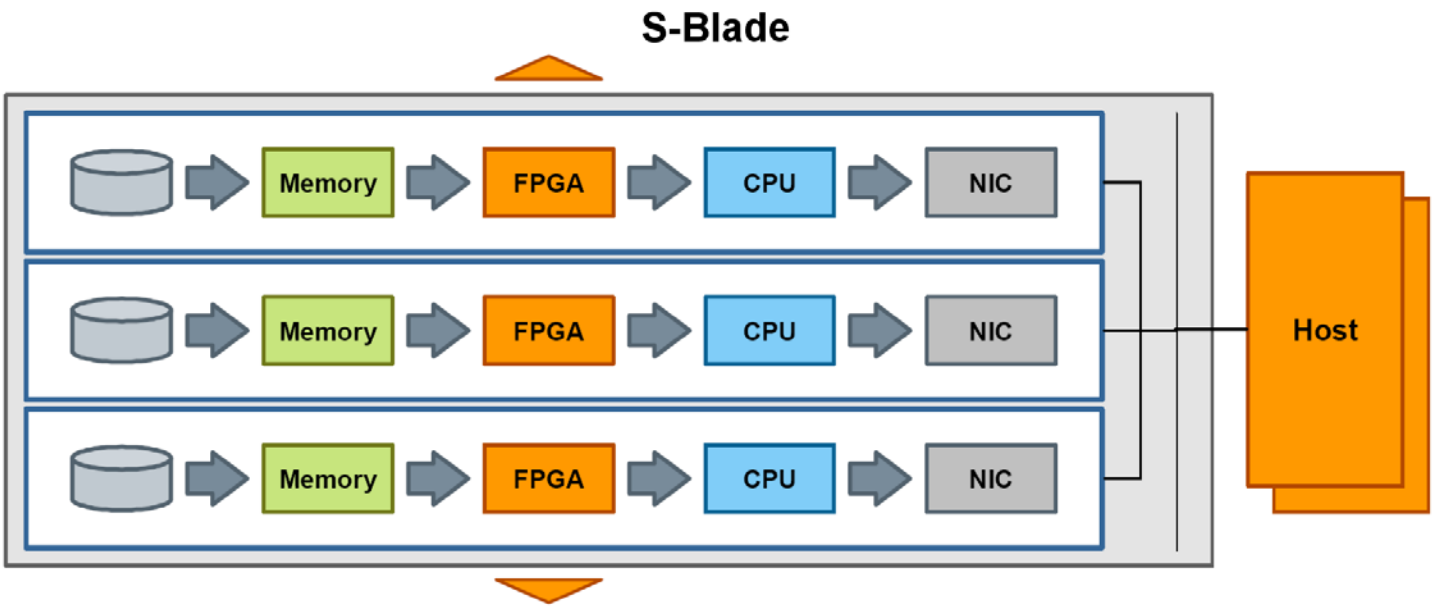


Abb. 14.4.5
Die Summe der S-Blades implementiert einen Shared Nothing Ansatz

Eine dediziertes High-Speed Interconnect Netzwerk (nicht gezeigt) liefert komprimierte Daten von den Festplatten in den Hauptspeicher der S-Blades mit maximaler Streaming-Geschwindigkeit. Die FPGAs entpacken und filtern 95-98% der Daten der gespeicherten Tabellen „on the fly“ heraus. Übrig bleiben nur die Daten, die benötigt werden, um die Anfrage zu beantworten. Diese werden von den CPUs der S-Blades weiterverarbeitet.

Anfragen werden im SQL Format gestellt (mit einigen Erweiterungen).

Der Network Interface Controller (NIC) stellt die Verbindung zum aktiven Host her.

http://www.cedix.de/VorlesMirror/Band2/PDSA01.pdf

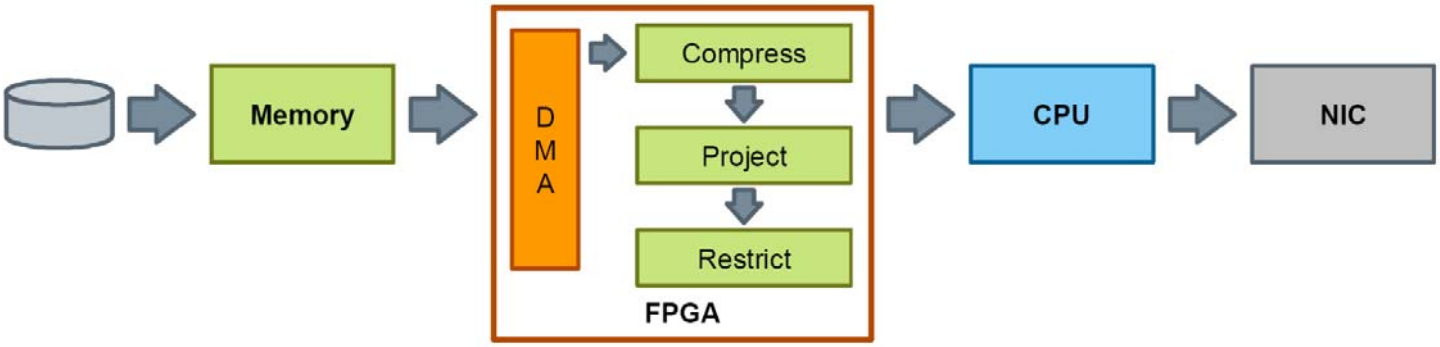


Abb. 14.4.6
Funktionen der FPGAs

Die FPGAs enthalten drei Engines:

- **Die Compress Engine dekomprimiert Daten "on the fly". Das Ergebnis ist eine signifikante Beschleunigung der langsamsten Komponente in jedem Data Warehouse, der Festplatte.**
- **Die Project and Restrict Engines bewirken eine weitere Leistungssteigerung durch Herausfiltern von Spalten und Zeilen, bezogen auf Parameter in den SELECT- und WHERE-Klauseln einer SQL-Abfrage.**
- **Die Visibility Engine spielt eine entscheidende Rolle bei der Aufrechterhaltung der ACID-Bedingungen. Sie filtert Zeilen aus, die bei einer Abfrage unsichtbar sein sollten, z. B. Zeilen, die zu einer Transaktion gehören, welche noch nicht abgeschlossen (COMMIT) wurde.**

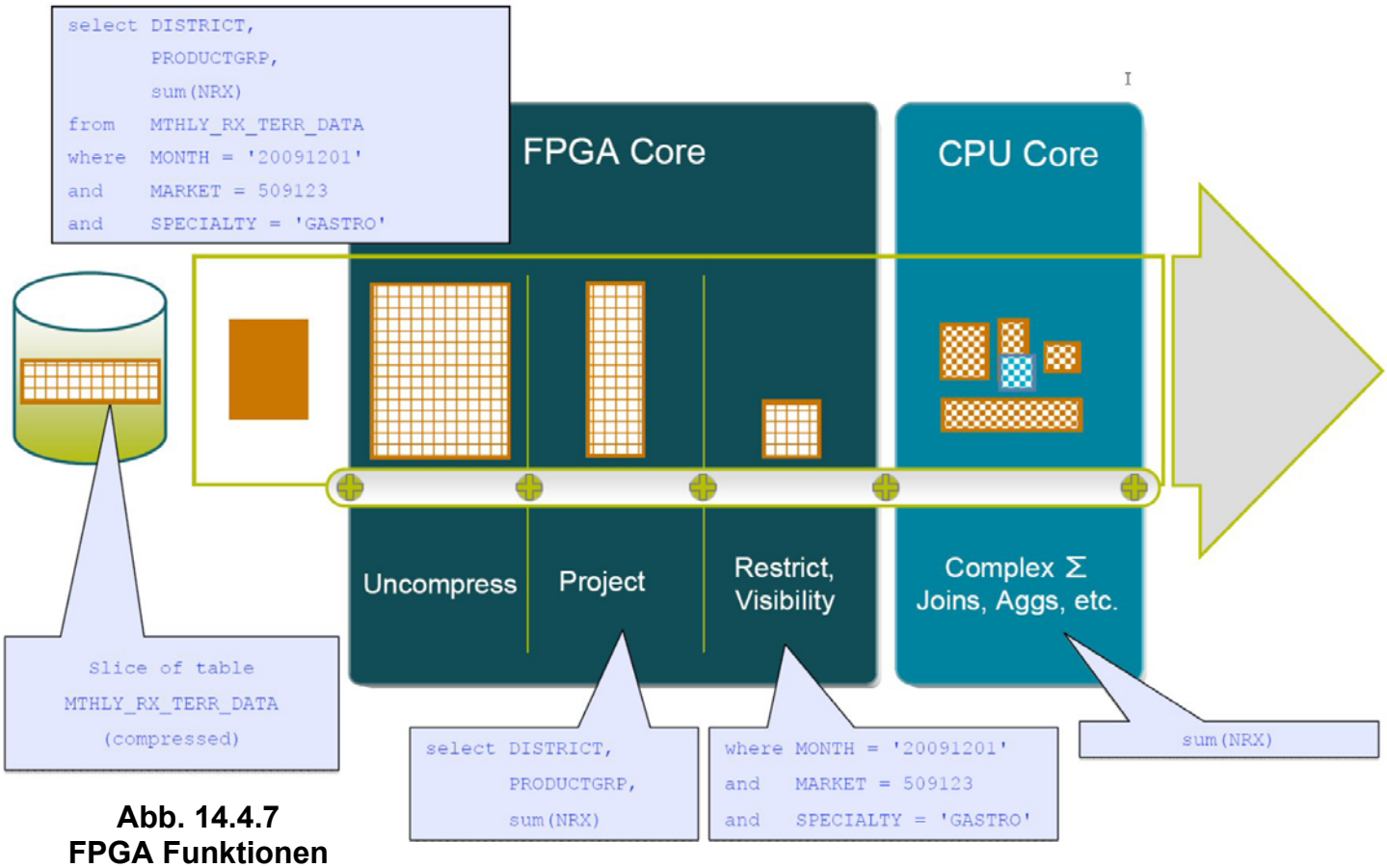
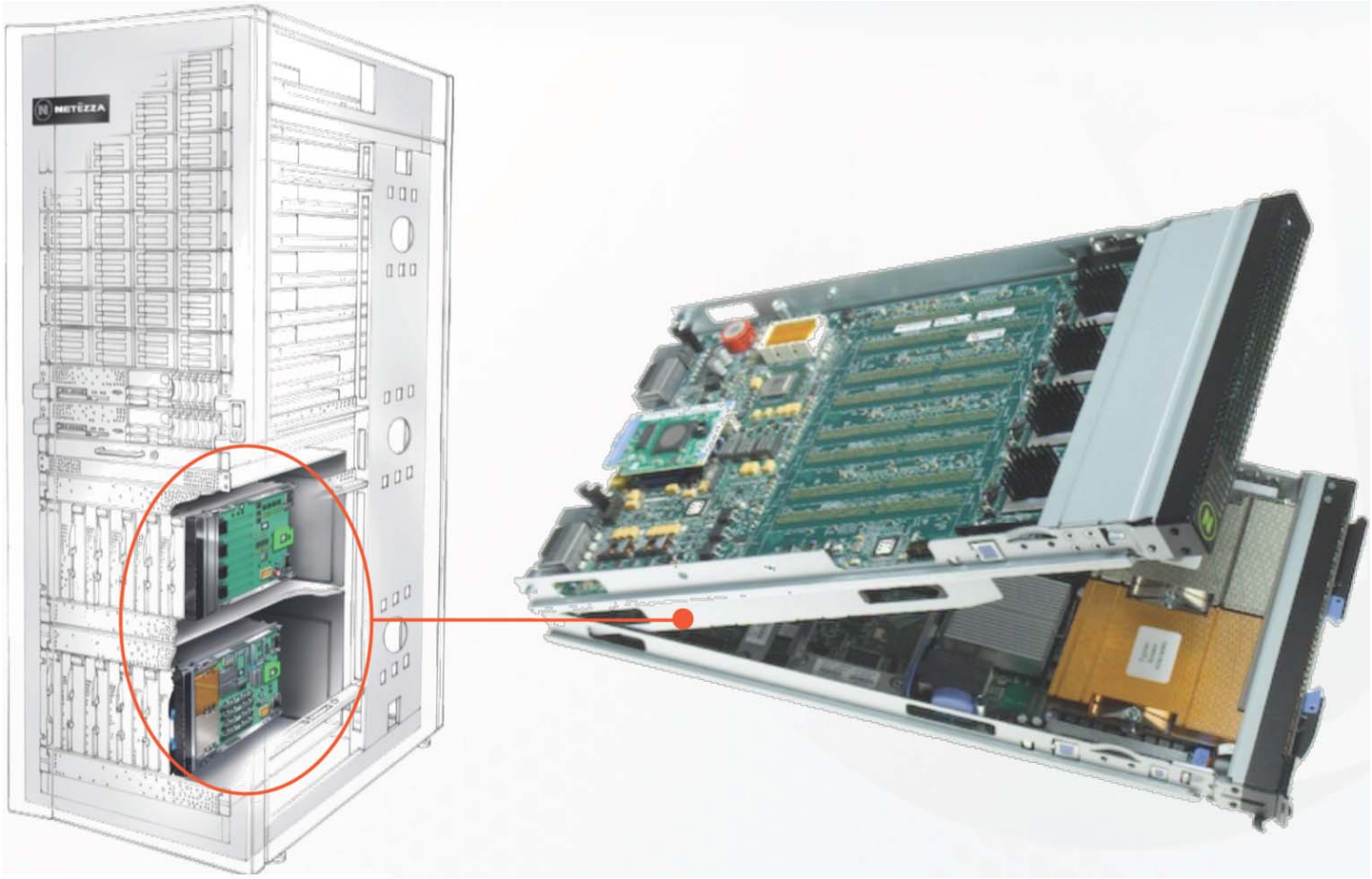


Abb. 14.4.7
FPGA Funktionen

14.4.6 PureData System for Analytics N2001-010 Hardware

Abb. 4.4.8
PureData System for Analytics N2001 Hardware



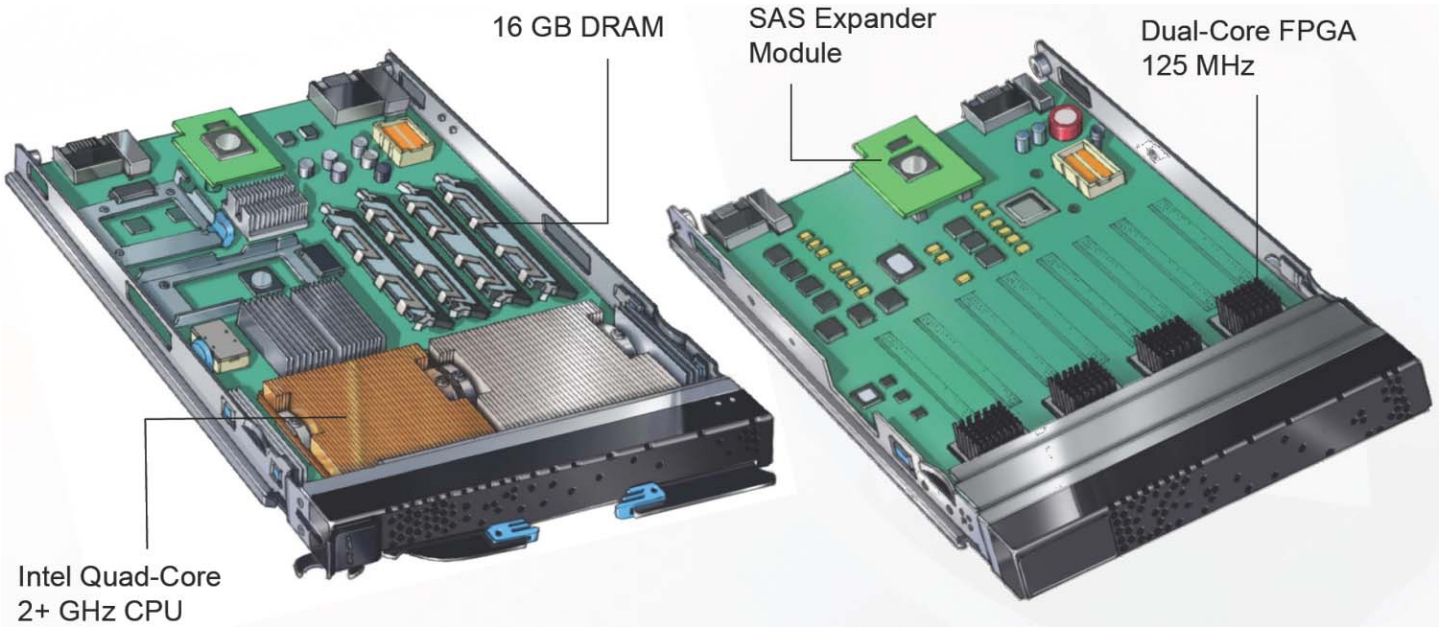


Abb. 14.4.9
Aufbau der S-Blades

Jedes PureData for Analytics S-Blade besteht aus 2 Platinen. Die linke Platine ist ein Standard-x86-Blade, wie es von IBM auch in anderen Blade-Center-Produkten eingesetzt wird. Die rechte Platine ist proprietär. Sie enthält die FPGA-Bausteine, die für den großen Performance-Gewinn verantwortlich sind.

14.4.7 Benutzer definierte Funktionen

FPGAs sind Halbleiterchips, die programmiert werden können. Sie werden seit vielen Jahren für eine Vielzahl von Anwendungen eingesetzt.. Netezza ist das erste Produkt, welches FPGAs benutzt, um Streaming-Daten in einem Data Warehouse Appliance verarbeiten. In allen anderen Data Warehouse Systemen werden alle Daten für eine Abfrage eingelesen. Danach wird die SQL WHERE Clause verarbeitet. Mit Netezza, anstatt eine riesige Menge von Daten zu bewegen, verarbeitet ein FPGA die WHERE Clause als Teil des Streamings der Daten von der Festplatte. Nur die Daten, die für die Verarbeitung des nächsten Schrittes erforderlich sind, werden weiter gereicht.

PureData System for Analytics (Netezza) Benutzer können UDFs (User Defined Functions) und UDAs (User Defined Aggregates) schreiben (beide als UDxs bezeichnet). UDAs geben einen Wert für mehrere Zeilen zurück, während UDFs einen Wert pro Zeile zurückgeben. Auf UDAs- und UDFs kann mit einer einfache SELECT-Anweisung zugegriffen werden.

Mit UDxs erlaubt Netezza das Schreiben von eigener Verarbeitungslogik, wie Data Mining-Algorithmen, für die S-Blades. Durch die Umsetzung dieser und anderer Technologien erreicht Netezza Geschwindigkeiten, die 10-100 Mal höher als bei andere Lösungen sind.

Es ist möglich, eigene UDxs erstellen. Unabhängige Softwareanbieter wie Fuzzy Logix (http://www.fuzzyl.com/) bieten Software-Pakete an, die Hunderte von vorgefertigten Mathematik, Statistic, Data Mining Algorithmen, sowie finanziellen Modellen enthalten.

14.4.8 IBM DB2 Analytics Accelerator for z/OS

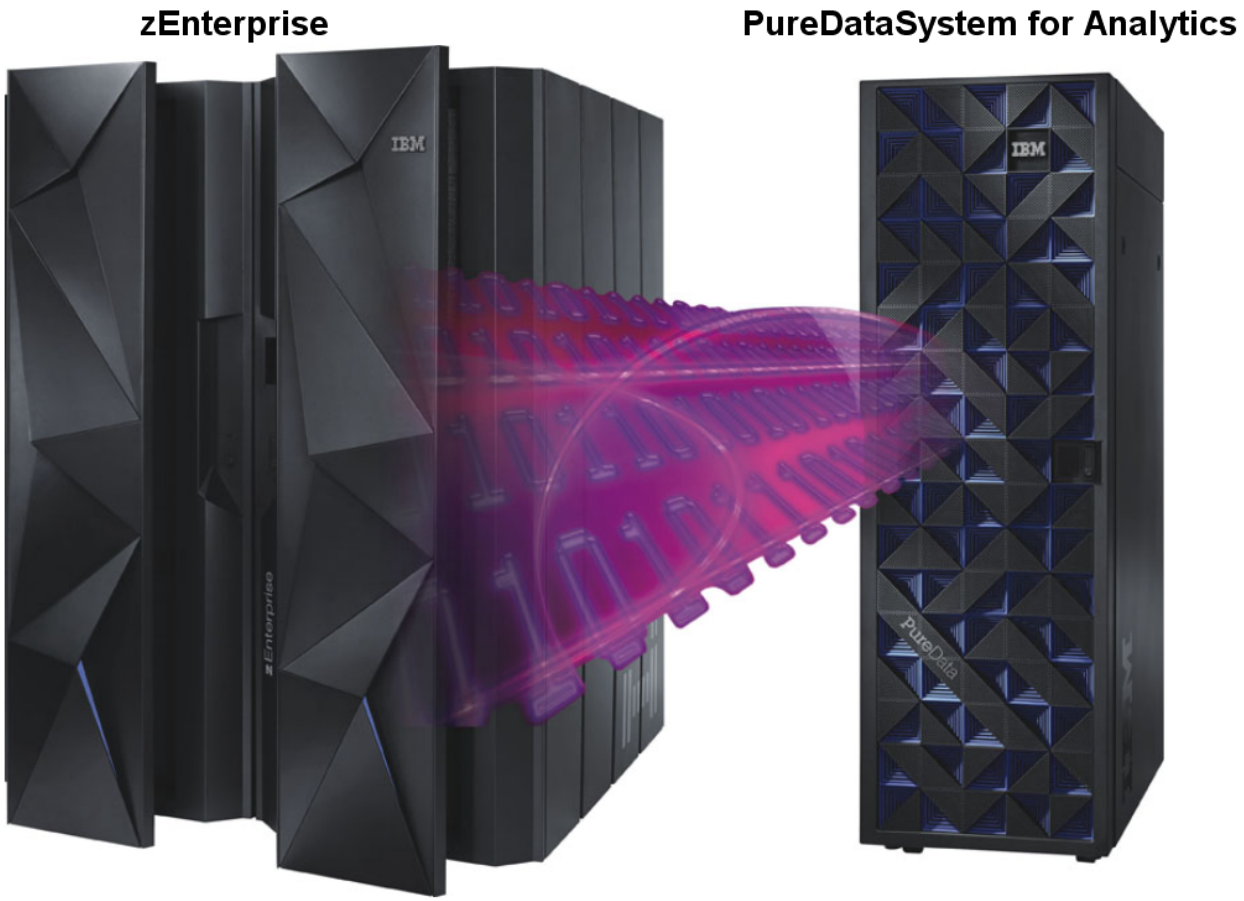


Abb. 14.4.10
14)Netezza Anschluss an einen zEC12 Rechner

In einem mittelgroßen Unternehmen kann ein Unix-, Linux- oder Windows-Rechner als Primärsystem dienen, auf dem die unternehmenskritischen Anwendungen laufen. Ein PureData System for Analytics kann an beliebige Primärsysteme angeschlossen werden. Von besonderem Interesse ist jedoch die Verbindung mit einem z/OS-Rechner, weil hier zahlreiche zusätzliche Funktionen möglich sind.

Der "IBM DB2 Analytics Accelerator" for z/OS (IDAA, englische Aussprache „Eye-Da") ist eine Data Warehouse Appliance für IBM System z und basiert auf dem PureData System for Analytics (Netezza). IDAA kombiniert Eigenschaften von DB2 für z/OS mit den Stärken eines PureData System for Analytics.

IDAA ist in DB2 für z/OS integriert. Es wird eine vollständige Transparenz der Anwendungen erreicht, die Abfragen an DB2 für z/OS übergeben. Benutzer und Anwendungen sehen nur eine DB2-für-z/OS Schnittstelle.

IDAA nutzt Kopien der DB2 -Tabellen, die im PureData-System gespeichert werden. Dies bietet das Beste aus beiden Welten: die gute Leistung von DB2 für z/OS für transaktionale Abfragen und branchenführende Leistung für analytische Abfragen. Abfragen werden von IDAA an das PureData-System weitergeleitet und von diesem verarbeitet.

14.4.9 IDAA – z/OS Integration

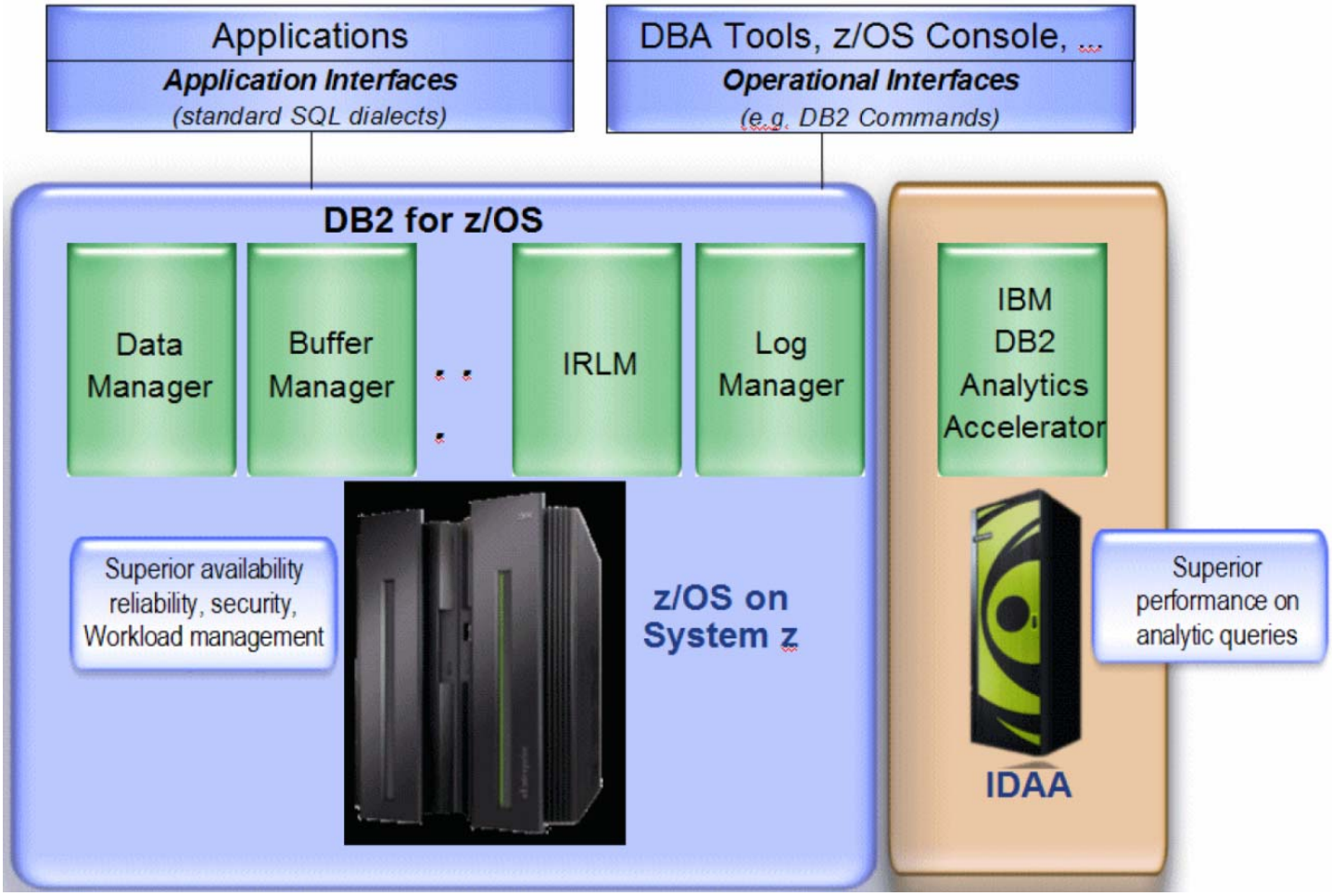


Abb. 14.4.11
Der DB2 Analytics Accelerator ist eine zusätzliche DB2 Komponente

DB2 besteht aus mehreren internen Komponenten, den Ressourcenmanagern: Buffer Manager für Daten-Cache-Management, Interne Ressourcen-Manager (IRLM) für Locking, usw. (Band 1, Abschnitt 7.1.2). Diese Komponenten sind für die Außenwelt von Anwendungen und Datenbank-Administration nicht sichtbar, wenn diese sich mit DB2 durch Anwendungs-Schnittstellen wie SQL oder operative Schnittstellen wie Commands und Utilities verbinden. Die interne Verarbeitungssteuerung unter den Komponenten ist vollständig transparent für die Außenwelt. Wir können damit einen anderen Ressource Manager hinzufügen unter Beibehaltung der bestehenden Schnittstellen. Keine Änderung einer Anwendung oder eines Datenbank Administrations-Verfahrens ist erforderlich.

Die tiefe Integration des PureData-Systems in DB2 für z/OS unterscheidet IDAA von anderen Data-Warehouse-Lösungen.

14.4.10 DB2 Optimizer

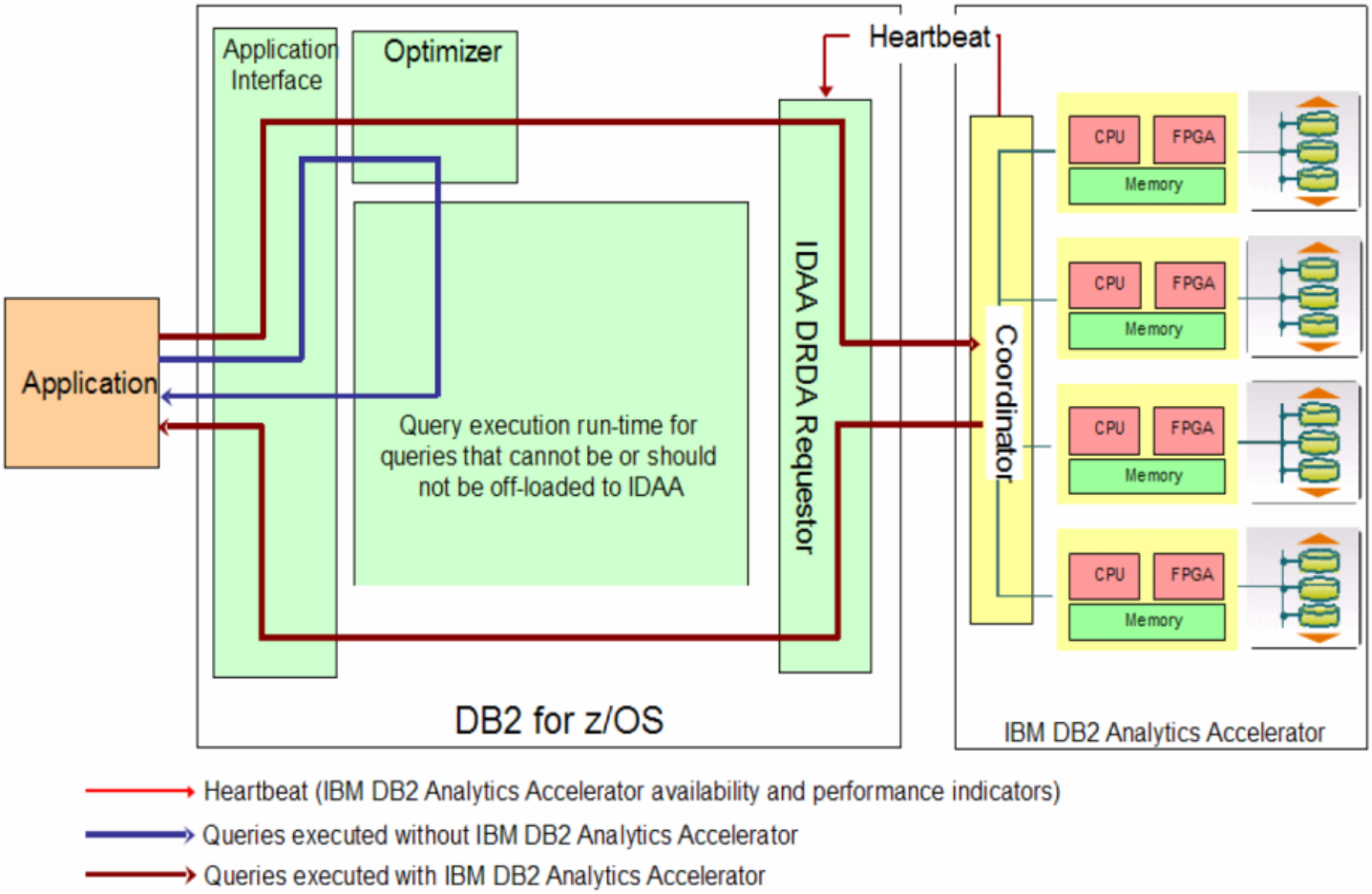


Abb. 14.4.12
Anschluss des DB2 Analytics Accelerators

Die Optimizer-Komponente von DB2 für z/OS entscheidet, ob eine SQL-Anfrage unter z/OS verarbeitet wird, oder an IDAA und letztendlich an das PureData-System weitergeleitet wird.

z/OS und der IBM DB2 Analytics Accelerator for z/OS repräsentieren derzeitig die technologische Speerspitze auf dem Gebiet Data Mining.

z/OS Appliances sind Acceleratoren für z/OS. Neben den Power und x86 Blades in einer zBX spielen „WebSphere DataPower Appliances“ (Abschnitt 14.3.3) eine wichtige Rolle. Sie sind ebenfalls eng in die zBX Infrastruktur integriert.

Ein PureData System for Analytics kann zunächst in beliebigen Umgebungen eingesetzt werden. Es kann aber auch in Verbindung mit IDAA als eine z/OS Appliance eingesetzt werden. In dieser Rolle benutzt es heute nicht die zBX und Unified Resource Manager Infrastruktur, und dafür eine eigene DB2 Infrastruktur Integration. Wir spekulieren, dass die zBX und Unified Resource Manager Infrastruktur Integration zu einem späteren Zeitpunkt ebenfalls erfolgen wird.

In der Zwischenzeit sind das PureData System for Analytics und IDAA weitere Beispiele, wo sich das z/Ost-Betriebssystem von anderen Platformen durch seine zahlreichen technologisch führenden Eigenschaften abhebt.

4.5 Weiterführende Information

Ein technisches Hewlett Packard Video
http://www.youtube.com/watch?v=PpbEeW7ItHs&feature=endscreen&NR=1

Eine detaillierte zBX Darstellung ist enthalten in
http://www.cedix.de/VorlesMirror/Band2/EMA02.pdf

Ein vertriebsorientiertes Video über zBX und den Unified Resource Manager existiert unter

http://www.youtube.com/embed/CVPS1vkCaBs

und

http://www.engadget.com/2010/07/23/ibms-zenterprise-architecture-makes-mainframes-cool-again-also/

Ein Netezza Video ist zu finden unter

http://www.youtube.com/watch?v=iO-Mg5AhmXw

http://www.cedix.de/VorlesMirror/Band2/zBXdetail.pdf
enthält eine detaillierte Präsentation zum Thema zBX

http://www.cedix.de/VorlesMirror/Band2/PDSA01.pdf

beschreibt die PureData System for Analytics N2001 Ankündigung Februar 2013

FPGA Tutorials sind zu finden unter
http://www.cs.ucr.edu/~vahid/courses/122b_w02/dt96_fpgatutorial.pdf
http://www.altera.com/literature/tt/tt_my_first_fpga.pdf
http://www.eetimes.com/design/programmable-logic/4015129/How-to-design-an-FPGA-from-scratch
http://www.mu21.de/Development//users/.michael/studium/vhdl/2-praktikum/dokumentation/1-calculator.pdf

Ein FPGA Video Tutorial ist zu finden unter
http://www.youtube.com/watch?v=wwkGmDaa4oM

Beispiele für Spezialelektronik unter Benutzung von FPGAs sind zu finden unter
http://www.gbm.de/fpga-systeme/?gclid=CPLD1pergbQCFUy5zAodCkgAYQ

Ein Home Computer (nicht x86 kompatibel) mit FPGAs implementiert ist zu sehen in
http://www.youtube.com/watch?v=dPuSGyrbd6c

Ein weiterer Home Computer
http://www.youtube.com/watch?v=GK6QzsSut_E

Ein FPGA PC in Aktion ist zu sehen unter
http://www.youtube.com/watch?v=V0jXQXZHToE

Interessante Details in:
IBM Redbook: Optimizing DB2 Queries with IBM DB2 Analytics Accelerator for z/OS
http://www.redbooks.ibm.com/abstracts/sg248005.html

Andersartige Data Mining Ansätze von Oracle und SAP, siehe
http://www.cedix.de/VorlesMirror/Band2/SAP01.pdf

Trend: Zentralisierung auf dem Mainframe

Siehe:

http://www.cedix.de/VorlesMirror/Band2/Rubin01.pdf

15. Java Enterprise Edition

15.1 Java Virtual Machine

15.1.1 Eigenschaften der Java Virtual Machine

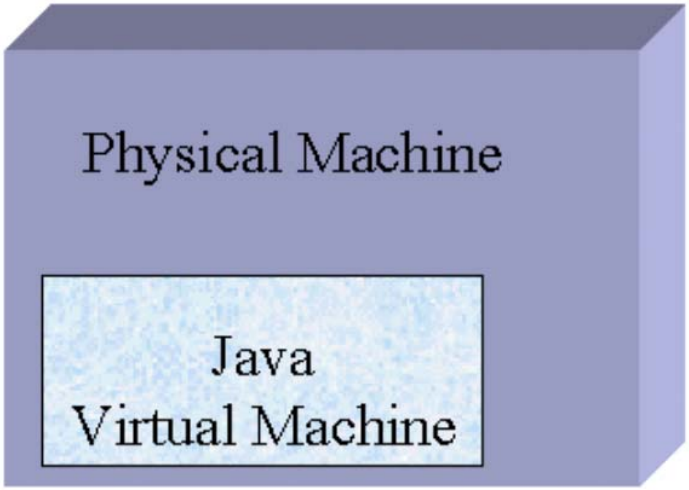


Abb. 15.1.1
Emulation der Java Hardware Architektur

Im Zusammenhang mit Java ist eine neue Hardware Prozessor Architektur entwickelt worden, vergleichbar zur System z, PowerPC oder x86 Architektur. Diese Architektur wird als Java Virtual Machine Architektur (JVMA) bezeichnet.

Wenn Java Quellcode kompiliert wird, entsteht Object Code. Dieser Objekt Code wird als „Byte Code“ bezeichnet.

Das Ausführungen eines Java Programms erfolgt im Prinzip auf speziellen Java-Prozessoren. Dies sind Mikroprozessoren, die Java-Bytecode als Maschinensprache verwenden. Sie wurden jedoch nie in größeren Stückzahlen gebaut. Statt dessen wird die Java Virtual Machine Architektur fast immer auf anderen Rechner Plattformen mit Hilfe einer Java Virtuellen Maschine (JVM) emuliert. Dies ist vergleichbar zur System z Emulation auf einem x86 Rechner mittels Hercules oder zPDT .

Jazelle DBX (Direct Bytecode eXecution) ermöglicht einigen ARM Processor Versionen die Ausführung von Java Bytecode in Hardware als Alternative zur normalen JVM Programm Ausführung. Die Dalvik virtuelle Maschine ist eine inkompatible Alternative zur JVM, die von Android für die Ausführung von Java Code benutzt wird.

Die Java Virtual Machine bietet neben der Platformunabhängigkeit auch einen Gewinn an Sicherheit. Eine JVM überwacht zur Laufzeit die Ausführung des Programms, verhindert also z. B., dass ein Programm über Arraygrenzen hinweg liest oder schreibt. Im speziellen Fall von Java fällt diese Überwachung sehr einfach aus, da Java keine Zeiger unterstützt. Somit werden Pufferüberläufe verhindert, die vor allem bei den in C oder C++ geschriebenen Programmen vorkommen können.

Um die Geschwindigkeit ("performance") der Programmausführung zu erhöhen, setzen die meisten JVM Implementierungen sogenannte JIT-Compiler ein, die unmittelbar während des Programmablaufs den Bytecode „Just In Time“ („Gerade zur rechten Zeit") in Maschinencode übersetzen. Eine Weiterentwicklung dieses Ansatzes, der Hotspot-Optimizer von Sun, behebt teilweise den Geschwindigkeitsnachteil der JVM, der hohe Speicherbedarf bleibt jedoch bestehen. Außerdem hat Java entwurfsbedingt einige Performance Nachteile, vor allem durch die automatische Speicherbereinigung (garbage collection). Einige JVM Implementierungen, zum Beispiel Insignia Jeode oder IBM J9, können diese Nachteile teilweise kompensieren.

BEA Jrockit, Sun Hotspot und IBM J9 sind die derzeitig führenden JVM Implementierungen.

Eine Java Anwendung, welche keine Probleme mit der begrenzten Hauptspeichergröße hat, läuft im 64 Bit Adressiermodus langsamer als im 31 Bit Adressiermodus !

Innerhalb einer JVM können mehrere Prozesse (Threads) ablaufen. Servlets machen beispielsweise hiervon Gebrauch.

Die JVM schottet die Threads vom Betriebssystem ab. Dies hat zur Folge, dass es nicht mehr möglich ist, mit systemeigenen Mitteln Java Prozesse zu kontrollieren. Die JVM stellt aber auch keine eigenen Funktionen zur Prozesskontrolle und -steuerung bereit. Somit können diese auch nicht von außen beendet werden, wenn sie aufgrund eines Fehlers das Gesamtsystem stören. Im Fehlerfall muss die gesamte JVM beendet werden. Viele JVM Implementierungen erlauben deshalb das direkte Abbilden (Mappen) von dedizierten Java-Threads auf Betriebssystem-Prozesse (native Threads).

Heute ist das Mappen von Java-Threads in der Regel die Default-Einstellung, d. h. nur in Ausnahmefällen wie dem Verwenden einer älteren JVM erfolgt das Thread-Management nur durch die JVM und nicht durch das Mapping auf Threads des Betriebssystems.

Die Isolation der Threads innerhalb der JVM untereinander ist unvollständig. Dies ist besonders kritisch, wenn mehrere Transaktionen als threads in der gleichen JVM verarbeitet werden. Hierauf wird später detailliert eingegangen.

15.1.2 Heap und Stack

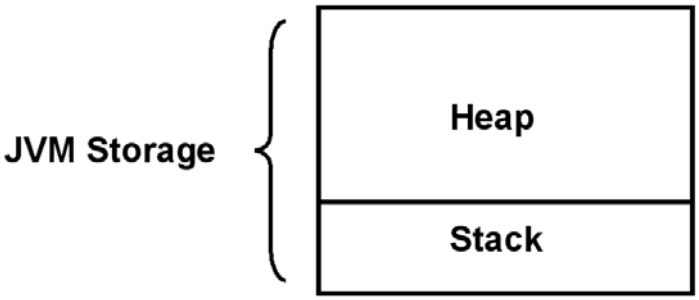


Abb. 15.1.2
Aufteilung in Heap und Stack

Der JVM Interpreter verwaltet einen Speicherbereich, der dem Anwendungsprogrammierer zugänglich ist, und der aus den beiden Teilen Heap und Stack besteht. Der Heap enthält unterschiedliche Arten von Information während der Programmausführung:

- **Lokale Variablen und Stacks für die aktiven Methoden, und**
- **Arrays und Objekte, die während der Programm Ausführung erzeugt werden.**

Der Heap existiert nur einmal pro Instanz der VM.

Der Heap dient zur Speicherung von dynamischen Informationen über konkrete Objekte. Für jedes Objekt sind dies sämtliche Instanzvariablen - sowohl die, die in seiner eigenen Klasse deklariert wurden, als auch die aus sämtlichen Vorfahren dieser Klasse.

Arrays und Objekte können nicht auf dem Stack gespeichert werden. Sie werden in dem Heap gespeichert.

Der `new`-Operator, der vom Anwendungsprogramm aufgerufen wird, um eine Variable auf dem Heap anzulegen, gibt dem Anwendungsprogramm eine Referenz auf die im Heap erzeugte dynamische Variable zurück. An welcher Stelle des Heaps die dynamische Variable angelegt wird, entscheidet nicht der Programmierer, sondern die virtuelle Maschine. Über diese Referenz kann man dann auf die dynamische Variable im Heap zugreifen.

Die Größe des Heaps ist beschränkt. Daher kann es zu einem Überlauf des Heaps kommen, wenn ständig nur Speicher angefordert und nichts zurückgegeben wird. Weiterhin wird mit zunehmendem Gebrauch der Heap zerstückelt, so dass der Fall eintreten kann, dass keine größeren Objekte mehr auf dem Heap angelegt werden können, obwohl in der Summe genügend freier Speicher vorhanden ist, aber eben nicht an einem Stück.

In Java werden Objekte im Heap nicht explizit freigegeben. Es wird vielmehr in unregelmäßigen Abständen durch die virtuelle Maschine der „Garbage Kollektor“ aufgerufen. Der Garbage Collector gibt den Speicherplatz, der nicht mehr referenziert wird, frei. Er ordnet ferner den Speicher neu (Defragmentierung), so dass auf dem Heap wieder größere homogene unbenutzte Speicherbereiche entstehen.

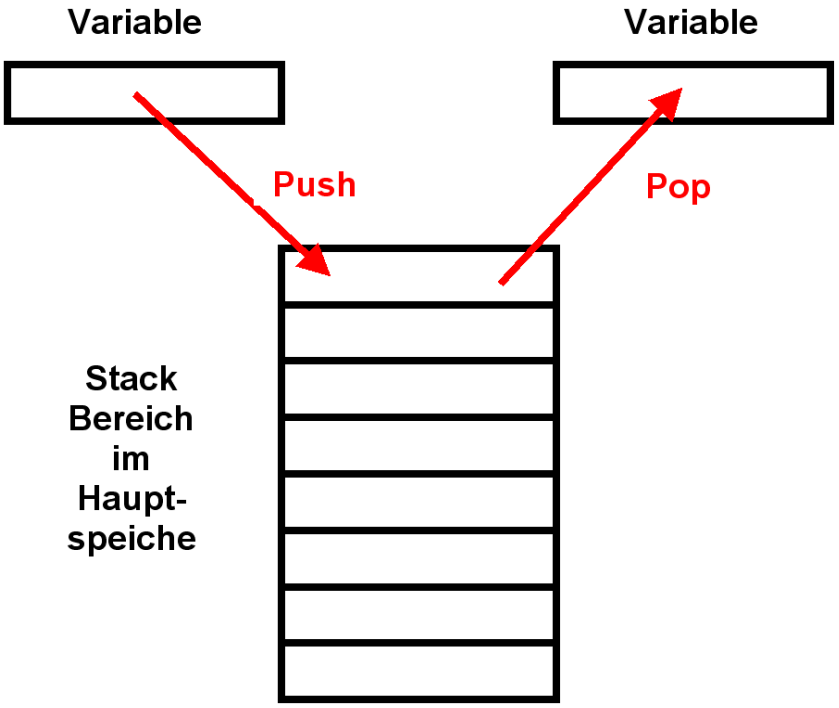


Abb. 15.1.3
Stack Operation

Ein Stack ist ein Container, in dem man nur an der Spitze Elemente ablegen oder von oben wieder wegnehmen kann. Es besteht keine Möglichkeit, ein Element aus der Mitte zu entnehmen oder ein Element in die Mitte einzufügen. Diese Abarbeitungsreihenfolge wird auch als LIF0-Prinzip (last in–first out) bezeichnet. Mit der Methode `push()` kann ein Element auf einen Stack abgelegt werden. Mit der Methode `pop()` kann das oberste Element vom Stack wieder entnommen werden.

Ein Stack kann aus Variablen einfacher Datentypen aufgebaut werden - oder im Falle von Klassen auch aus Referenzen auf Objekte. Sind die Referenzen, die auf einem Stack gespeichert werden, vom Typ Object, so kann jede beliebige Referenz auf dem Stack abgelegt werden.

Der Java-Stack speichert lokale Variablen innerhalb von Methodenaufrufen, sowie Operanden für arithmetische Operationen. Bei jedem Methodenaufruf wird ein Stack-Frame auf den Stack gepuscht, das Platz für die lokalen Variablen dieser Methode und einige Zusatzdaten bietet. Wird die Methode (durch `Return` oder durch eine Exception) verlassen, wird der zugehörige Frame wieder vom Stack genommen.

Die JVM unterhält eine getrennten Stack für jeden Thread.

15.1.3 Class Loader

In Java wird kein ausführbares Programm durch einen Linker erzeugt. Statt dessen werden benötigte Klassen zur Laufzeit durch die JVM nachgeladen. Diese verwendet hierfür eine Klasse `ClassLoader` :

```
public abstract class ClassLoader extends Object
```

Ein `ClassLoader` erhält durch den Aufruf seiner Methode `loadClass()` die Aufforderung, Klassen aus `.class`-Dateien oder `.jar`-Archiven vom lokalen Dateisystem laden. Um diese Dateien zu lokalisieren, wertet der System-ClassLoader die Umgebungsvariable `CLASSPATH` aus.

Ein Java Programm besteht aus einem losen Verbund von einzelnen `.class` Dateien, die bei Bedarf in die virtuelle Maschine geladen werden. Klassen, die eine `main()` Methode haben, können zum Starten einer Anwendung benutzt werden.

Jedes Klassen-Objekt enthält eine Reference zu dem ClassLoader der es definierte. Es existiert ein spezieller Bootstrap-ClassLoader für die Bootstrap-Klassen, und ein Erweiterungs-ClassLoader für die Klassen im lib/ext-Verzeichnis.

Der Java Development Kit (JDK) ist ein Subset des Software Development Kit (SDK). Der JDK ist zuständig für das Schreiben und Ausführen von Java Programmen. Der SDK enthält weitere Software, z.B. Debugger oder Dokumentation.

JDKs unterschiedlicher Hersteller befolgen grundsätzlich alle Java Specifikationen, können aber Funktionen enthalten, die in dem Java Standard nicht festgelegt sind. Beispiele sind Garbage Collection, Compilation Strategien und Optimierungsverfahren. Als Folge können bei der Portierung von Java Programmen Schwierigkeiten auftreten.

15.1.4 Java Runtime Environment

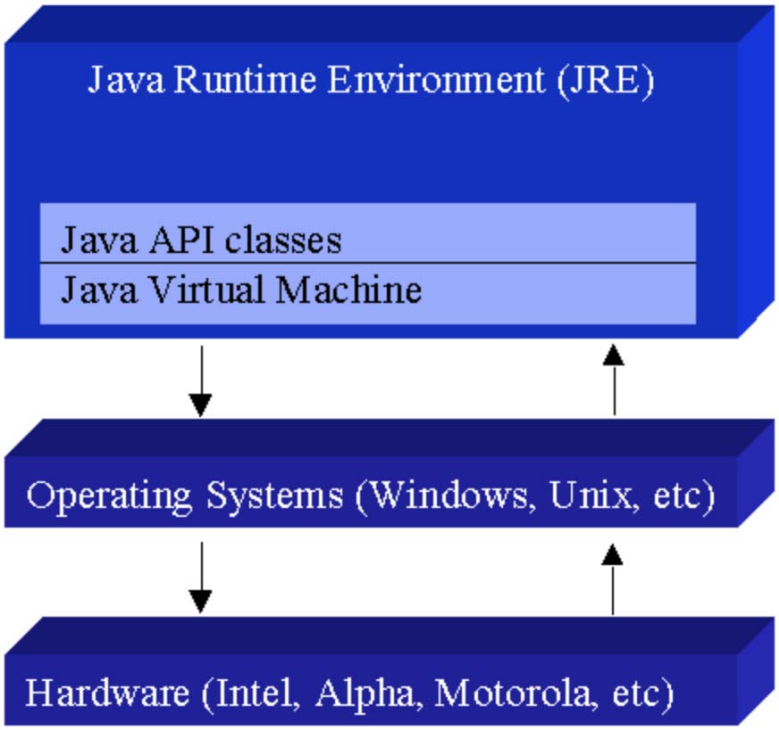


Abb. 15.1.4
Einbettung der JVM in die Java Runtime Umgebung

Die Java Virtual Machine ist Teil eines größeren Systems, dem Java Runtime Environment (JRE). Jedes Betriebssystem und jede Hardware Architektur arbeitet mit einem unterschiedlichen JRE. Das JRE besteht aus einem Satz von Base Klassen, welche eine JVM beinhalten wie auch eine Implementierung der Base Java API. Die JRE Implementierungen auf unterschiedlichen Plattformen ermöglichen die Portabilität von Java Programmen. Java Programme können auf einer bestimmten Plattform nur laufen, wenn dort ein JRE vorhanden ist.

Java arbeitet intern ausschließlich mit Unicode UTF-16 Encoding.

Bei der Default-Codierung werden von der JVM Unicode-Zeichen bis \u00FF so gut wie möglich in die Code-Tabelle des Betriebssystem abgebildet.

Dies muss z.B. beachtet werden, wenn ein Java Programm auf eine DB2 oder IMS Datenbank zugreift.

Die “z/OS Support for Unicode Conversion Services” konvertieren alphanumerische Daten zwischen UTF-8, UTF-16, ASCII und EBCDIC.

Dies kann kompliziert werden, weil die JVM selbst häufig in C/C++ implementiert ist, und auf ASCII Daten zugreift.

15.1.5 JVM unter z/OS und zLinux

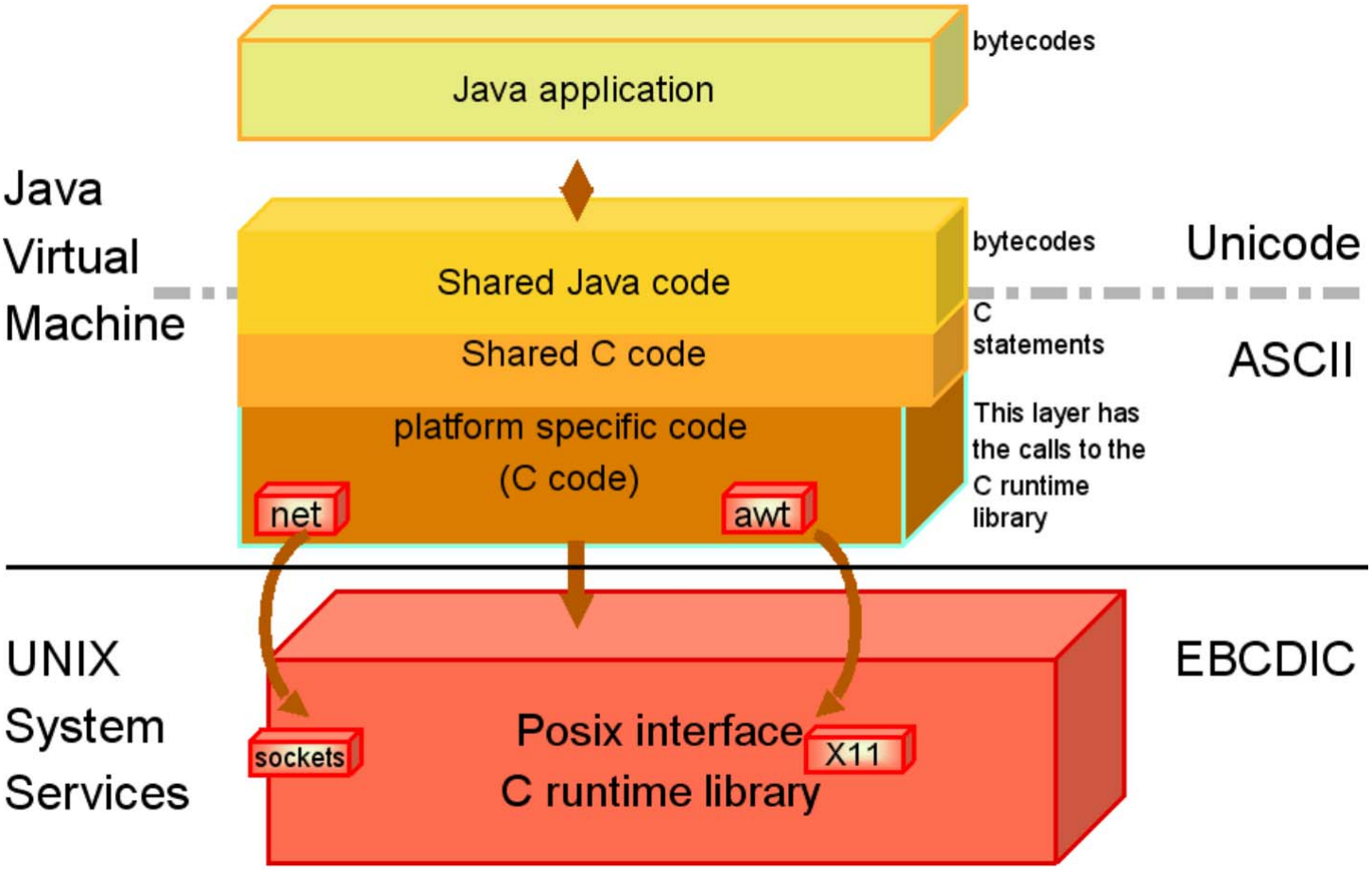


Abb. 15.1.5
Mainframe Implementierung der JVM

Die JVM ist auf den meisten Plattformen teilweise in C/C++ implementiert, so auch unter zLinux. Die zLinux JVM geht davon aus, dass alle Daten in ASCII gespeichert sind; der Zugriff auf EBCDIC Daten erfordert zusätzliche Maßnahmen.

Unter z/OS ist es komplizierter. Die JVM ist als Teil der Unix System Services (USS) (Abschnitt 17.1.2) implementiert und enthält aus Kompatibilitätsgründen ebenfalls ASCII Support. Die allermeisten z/OS Daten sind jedoch in EBCDIC gespeichert. Die JVM bemüht sich, bei einer Kommunikation mit JVMs auf anderen Plattformen diesen Unterschied weitgehend unsichtbar zu machen.

Wird die Environment Variable IBM_JAVA_ENABLE_ASCII_FILETAG definiert, und File Encoding für eine von z/OS unterstützte ASCII Code Page spezifiziert, dann werden alle neu angelegten Files mit einem "Coded Character Set Identifier" (CCSID) Tag versehen, der dieser Code Page entspricht. CCSID ist eine 16-bit Ziffer, die das Encoding spezifiziert.

15.1.6 Java Programmausführung

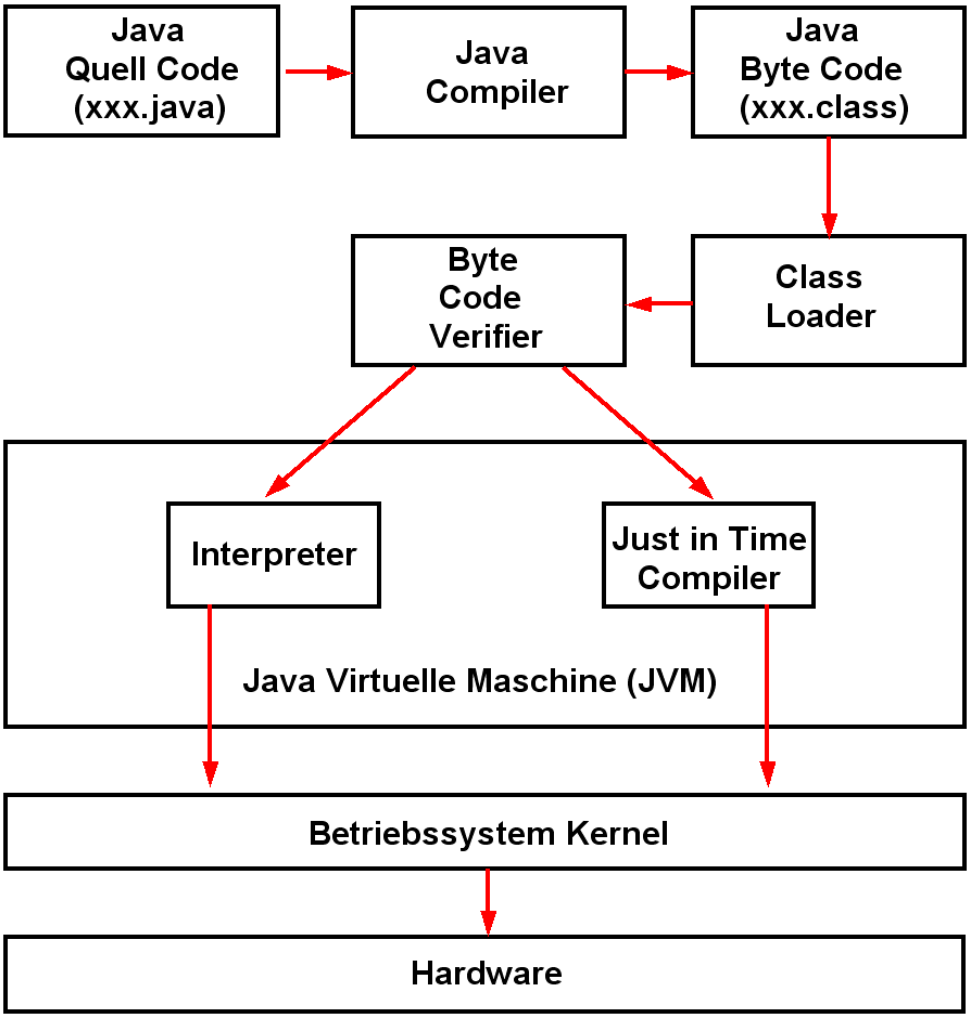


Abb. 15.1.6
Grundsätzliche Java Programm Ausführung

Der Java-Compiler übersetzt Java Quellcode in Java-Objekt-Code (universell als Java Byte Code bezeichnet).

Der kompilierte Code wird mit Hilfe eines Class Loaders und eines Byte-Code Verifiers in die JVM zur Ausführung geladen.

Der Code wird mit Hilfe eines Interpreters oder eines Just-in-Time-Compiler ausgeführt.

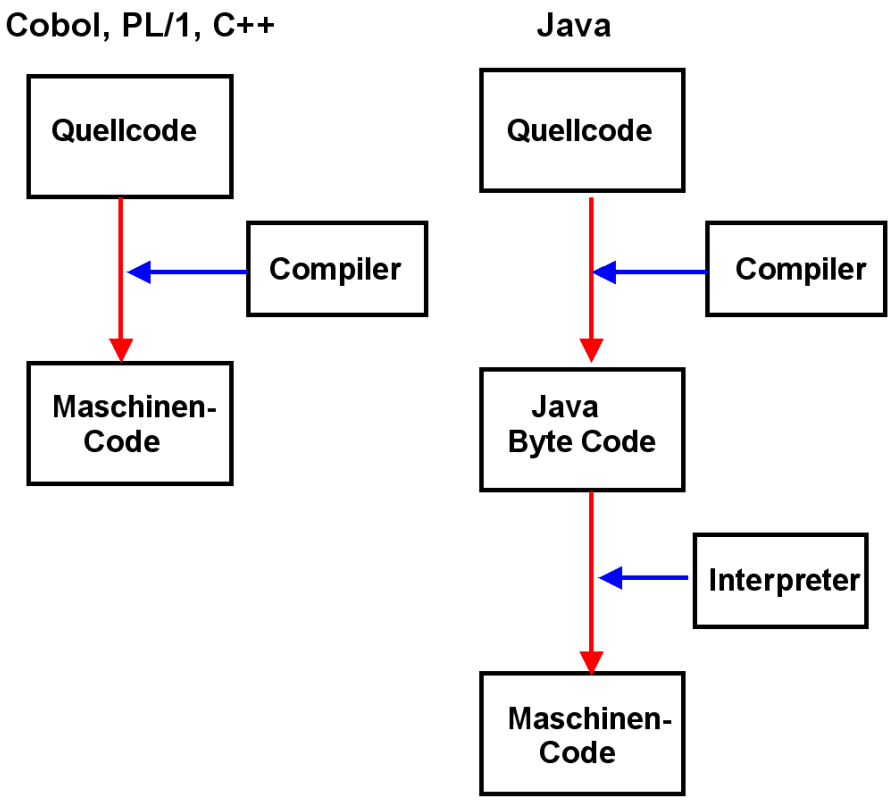


Abb. 15.1.7
Optimising und Just in Time Compiler

Cobol, PL/1 oder C/C++ Quell Code wird durch einen optimierenden Compiler (optimising compiler) in Object Code übersetzt, aus dem nach dem Linking und Loading ausführbarer Maschinencode entsteht.

Bei Java erzeugt der Compiler statt dessen Plattform-unabhängigen Byte Code, der anschließend in einer JVM durch einen Interpreter oder Just-in-Time Compiler ausgeführt wird.

Die JVM selbst ist plattformabhängig. Sie ist typischerweise in C++ implementiert.

Optimizing Compiler werden für Sprachen wie Cobol, PL/1, C++ eingesetzt; sie erzeugen für die Ausführung besonders schnellen Maschinencode. Interpreter und Just in Time Compiler (JIT) sind in der Regel langsamer.

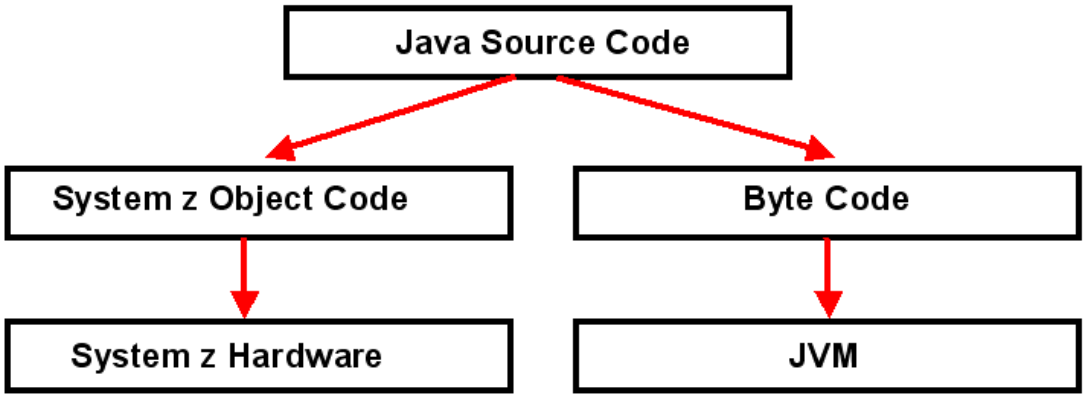


Abb. 15.1.8
z/OS Java Ausführungsalternativen

Unter System z ist die Ausführung sowohl als System z Object Code als auch als Byte Code möglich.

The z/OS High Performance Java Compiler hat die Option, System z Object Code (binaries) an Stelle von Java Byte Code zu generieren. Der System z Object Code kann dann direkt in den Hauptspeicher ohne Benutzung einer JVM gelinked und geladen werden.

Der übersetzte Objekt Code ist nicht mehr portierbar, hat aber eine kürzere Ausführungszeit. Es ist möglich, im Compiler Durchlauf gleichzeitig Byte Code zu erzeugen, der dann auch in anderen Umgebungen ausgeführt werden kann.

15.1.7 Java Anwendungen unter z/OS

Eines der wichtigsten z/OS Funktionen für Java ist die Verfügbarkeit von Record oriented Files (Data Sets) zusätzlich zu unstrukturierten Dateien.

JRIO ist eine Klassenbibliothek, ähnlich zu java.io. Während java.io Byte-orientierte oder feldorientierte Zugriffe auf Dateien ermöglicht, bietet JRIO einen Record-orientierten Zugang. Sie können mit JRIO auf VSAM-Datensätze und non-VSAM-Datensätze (PDS oder sequentiell) und HFS-Dateien zuzugreifen:

- **VSAM-Datensätze (nur KSDS)**
- **Non-VSAM Record orientierte Datensätze**
- **System Katalog**
- **Partitionierted Data Set (PDS) Directory**
- **DDNAME Unterstützung**

In VSAM KSDS können Sie auf Datensätze in Entry Sequence Reihenfolge oder durch primäre eindeutigen Schlüssel zugreifen. JRIO bietet indizierten I/O-Zugriff auf Datensätze innerhalb eines VSAM KSDS mit z/OS nativer Unterstützung.

Die Java Record I/O Funktionalität ist als Teil des IBM JZOS Batch Toolkits verfügbar.

Die meisten Java Anwendungen laufen unter z/OS innerhalb eines Application Servers wie dem CICS Transaction Server oder dem WebSphere Application Server (WAS). Eine Alternative sind Java Stand-alone Anwendungen.

Wir definieren eine Java Stand-alone Anwendung als eine Anwendung, die nicht innerhalb eines Applikationsservers läuft. Ein Java Stand-alone Anwendung verfügt über ein Main-Methode, die ähnlich wie der Main entry Point eines traditionellen COBOL oder PL/1 Lademoduls arbeitet. Die Main Method ist die erste Methode, die ausgeführt wird wenn die Anwendung gestartet wird. Sie kann Parameter enthalten, die der Anwendung übergeben werden.

Ein Java Stand-alone Anwendung läuft in einer eigenen Java Virtual Machine (JVM). Diese muss gestartet werden, ehe die ersten Befehle der Anwendung ausgeführt werden.

Ein Java Stand-alone Anwendung ist autonom und benötigt einen Trigger um gestartet zu werden. Dieser Trigger kann sein:

- **Jemand gibt manuell einen Befehl auf der Unix System Services Command Line ein, um eine JVM mit der Anwendung zu starten.**
- **Ein Job Scheduler startet ein JCL Script, das eine JVM mit der Anwendung startet.**

15.1.8 Trennung zwischen Entwicklung und Produktion

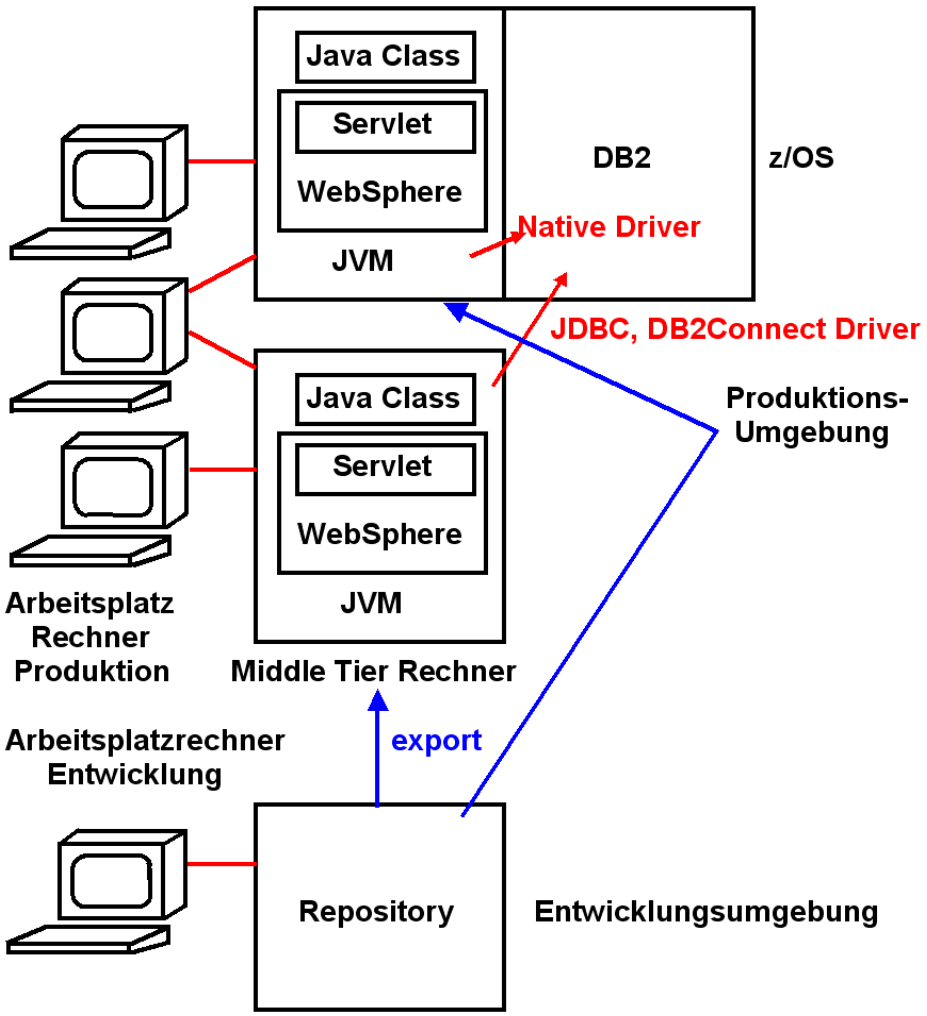


Abb. 15.1.9
Entwicklungsumgebung für neue Anwendungen

Die Entwicklung neuer Java Anwendungen kann mit Hilfe des JDK erfolgen. Sie erfolgt jedoch typischerweise in einer Entwicklungsumgebung (Integrated Development Environment, IDE).

Die Ausführung einer neu entwickelten Java Anwendung erfolgt häufig auf einem Web Application Server. Dieser befindet sich

- **entweder auf einem Middle Tier Server, der mit dem Mainframe verbunden ist, oder**
- **unmittelbar auf dem Mainframe Rechner**

Auf dem Mainframe läuft der Web Application Server entweder unter zLinux oder unter z/OS Unix System Services.

15.1.9 Integrated Development Environment (IDE)

Entwicklungsumgebungen sind Software Produkte, die von viele Herstellern erhältlich sind.

Weit verbreitet sind Eclipse , NetBeans, und IntelliJ IDEA

Eclipse und NetBeans sind Open Source. Eclipse kann heruntergeladen werden von www.eclipse.org . Ca. 350 MByte.

Eclipse (wie auch NetBeans und IntelliJ IDEA) ist sehr leistungsfähig und sehr beliebt, erfordert aber einen nicht unerheblichen Lernaufwand.

Eclipse hat eine sehr offene Architektur, und damit leichte Erweiterungsmöglichkeiten, die als Plug-ins bezeichnet werden.

Das Eclipse WebSphere Rational Application Developer für System z (RDz) Plug-in ist für z/OS Entwicklungen die wichtigste moderne Entwicklungsumgebung. RDz wurde wiederholt von IBM umbenannt: WSED → WDz → RDz, blieb aber im Wesentlichen das gleiche Software-Produkt.

Eclipse wurde ursprünglich für Java Programmierer entwickelt. Spezifisch das RDz Plug-in unterstützt aber auch Entwicklungen in COBOL, PL/I, C/C++ und Assembler.

15.2 Servlets

15.2.1 Statische HTML Seiten

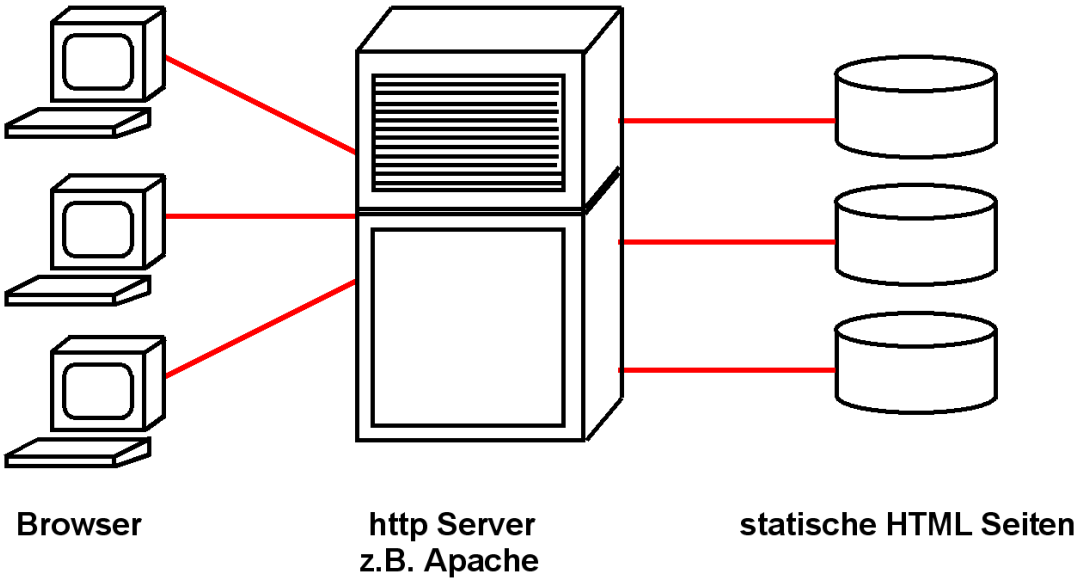


Abb. 15.2.1
Apache ist der am weitesten verbreitete http Server

Statische HTML Seiten sind der einfachste Fall einer Client/Server Anwendung. Der Klient (Browser) ruft eine (statische) HTML Seite auf, die unmodifiziert an den Klienten zurückgespielt wird. Die aus den statischen HTML Seiten bestehende Datensammlung wird lediglich offline geändert (z.B. durch das Hochladen mittels FTP). Der sehr große Vorteil: Es gibt keinerlei Datenintegritätsprobleme.

Es wird angenommen, dass 90% aller Web Zugriffe auf statische HTML Seiten erfolgen.

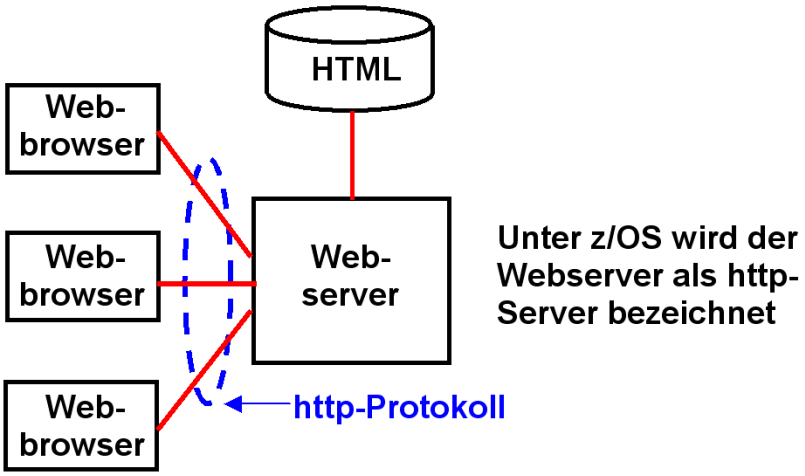


Abb. 15.2.2
http ist eines der am weitesten verbreiteten Internet Protokolle

HTML Seiten werden zum/vom Web Browser mit Hilfe des HTTP Protokolls übertragen. HTTP wird auch als „Web RPC“ betrachtet. Wie der eigentliche RPC ist HTTP zustandslos: Request/Response. Keine Session. HTTP erlaubt die Übertragung selbstbeschreibender Daten. Bei jeder Verbindungsaufnahme müssen Datenformate neu ausgehandelt werden.

HTTP ist eines von vielen Schicht 5 Übertragungsprotokollen. Beispiele für Schicht 5 Protokolle sind: Telnet, FTP, 3270, HTTP, SOAP, IIOP, RMI/JRMP, ...

Ein Web-Server unter z/OS, wie der HTTP Server unter Unix System Services, arbeitet ähnlich wie ein Web-Server auf anderen Plattformen. Der Benutzer sendet eine HTTP-Anfrage an den z/OS HTTP Server, um eine bestimmte Datei zu erhalten. Der HTTP Server ruft die Datei von seinem Datei-Repository ab, und schickt sie an den Benutzer, zusammen mit Informationen über die Datei (z. B. MIME-Typ und Größe) in dem HTTP-Header.

Der z/OS HTTP Server unterscheidet sich von anderen Web-Servern. Da z/OS Dateien im EBCDIC Format codiert sind, müssen z/OS Dokumente zunächst in das ASCII-Format konvertiert werden, ehe sie über das Internet an einen Endbenutzer geschickt werden. Binäre Dokumente und Bilder müssen nicht konvertiert werden.

Der z/OS HTTP Server führt diese Konvertierungen durch und erspart damit dem Programmierer diesen Schritt.

Für das Hochladen von Dokumenten verwendet der Programmierer normalerweise FTP. Dabei gibt der Programmierer ASCII als FTP-Transport-Format an. Der z/OS http Server konvertiert das Format dann automatisch in EBCDIC. Wenn „binary“ als FTP-Transport-Format angegeben wird, wird die Datei nicht konvertiert.

Der zLinux HTTP-Server hat diese Probleme nicht, da zLinux alle Daten im ASCII-Format verarbeitet und in der Regel kein EBCDIC benutzt.

Beim Transfer von Daten zwischen zwei Rechnern kann der Java Programmierer das Encoding der Daten spezifizieren.

Es wird behauptet, dass mehr als 50% aller betriebswirtschaftlich relevanter Daten, auf die über das WWW zugegriffen wird, im EBCDIC Format auf Mainframes gespeichert sind.

15.2.2 HTML Forms

Beim Aufruf von statischen HTML Seiten sendet der Browser lediglich eine URL an den Web Server. Für die Nutzung des Browsers und WWW als Client Server System ist es erforderlich, weitere Daten vom Klienten an den Server zu senden. Dies geschieht mittels „Form Data“. Weiterhin muss der Server eine Anfrage beantworten können. Bei der Nutzung von Java geschieht dies in der Regel mit Hilfe von Servlets und Java Server Pages (JSP).

HTML Forms sind ein einfache Werkzeuge, mit dem ein Benutzer Daten mit Hilfe des HTTP-Protokolls an einen Server schicken kann. HTML Forms erlauben es dem Browser, „Form Data“ an den Server zu senden,

Ein HTML-Form besteht aus einem Code-Block, der mit dem <FORM> Tag anfängt und dem </FORM> Tag aufhört. Eine HTML-Seite kann mehrere Forms enthalten.

Der FORM Tag spezifiziert:

- **Die zu benutzende HTTP-Methode. In den meisten Fällen ist dies POST; die Daten werden innerhalb des Bodys der Nachricht übertragen.**
- **Die Action. Dies ist meistens die URL, es kann aber auch die Action mit ihrem Namen angegeben werden.**
- **Der Typ der MIME-Enkodierung der Daten in der FORM. Der Default ist "application/x-www-form-encoded".**

Wenn Sie jemals eine Web-Suchmaschine verwendet haben, einen on-line Buchladen besuchten, Aktienkurse on-line verfolgten oder ein Quote für den Preis eines Flugtickets abfragten, haben Sie wahrscheinlich eine merkwürdig aussehenden URLs gesehen, wie

Dies ist ein Beispiel für eine GET Anfrage
http://host/path?user=Marty&origin=bwi&dest=lax

Dieser Teil wird als „Form Data“ bezeichnet und ist ein häufig benutztes Verfahren, um Daten von einer gesendeten Webseite an ein Server-seitiges Programm zu übergeben.

Daten zusätzlich zu einer URL sind Form Data. Sie werden in der Regel von einem Browser an einen Webserver in einem von zwei Formaten übertragen.

- **Für GET-Anfragen werden die Form Daten an das Ende der URL nach einem Fragezeichen angebracht, siehe obiges Beispiel.**
- **Für POST-Anfragen werden die Form Daten an den Server in die http Nachricht eingebettet.**

Für alle außer sehr einfachen Anfragen wird die POST-Methode verwendet.

Abb. 15.2.3
Logon Screen, generiert durch eine HTML Form

Die hier dargestellte Wiedergabe auf einer HTML Seite erwartet eine Eingabe seitens des Benutzers in den hiefür vorgesehenen Feldern.

Diese Darstellung wird mit Hilfe von Form Tags innerhalb des HTML Codes programmiert.

Folgend ist eine HTML Seite dargestellt, die vom Server an den Browser gesendet wurde. Nachdem der Benutzer die beiden Felder ausgefüllt hat und den „Submit“ Button aktiviert, wird der Inhalt der beiden Felder, zusammen mit der Action, als Form Data an den Server gesendet.

```
<HTML>
<HEAD><TITLE> Login </TITLE> </HEAD>
<BODY>
<H2>Login to Secure Site</H2>

< FORM METHOD=POST
ACTION="http://abc.de/servlet/HelloWorld.servlet" >

Username: <INPUT TYPE="TEXT" NAME="username"
SIZE="25"><BR>
Password: <INPUT TYPE="PASSWORD"
NAME="password" SIZE="25"><P>

<INPUT TYPE="SUBMIT" VALUE="Submit">
<INPUT TYPE="RESET" VALUE="Clear">
</FORM>

</BODY> </HTML>
```

Abb. 15.2.4
Form Tag zur Erzeugung des Logon Screens

Ein HTML-Form besteht aus einem Code-Block, der mit dem <FORM> Tag anfängt und dem </FORM> Tag aufhört. Eine HTML-Seite kann mehrere Forms enthalten.

Der FORM Tag spezifiziert:

- **Die zu benutzende HTTP-Methode. Hier ist dies POST; die Daten werden innerhalb des Bodys der http Nachricht übertragen.**
- **Die Action. Dies ist meistens die URL, es kann aber auch die Action mit ihrem Namen angegeben werden.**
- **Der Typ der MIME-Enkodierung der Daten in der FORM. Der Default ist "application/x-www-form-encoded".**

Wenn der Server die Nachricht mit den Form Date empfängt, wird das Programm xyz.servlet im Verzeichnis abc.de/servlet aufgerufen, wobei die beiden Variablen username und password übergeben werden.

Multipurpose Internet Mail Extensions (MIME) ist ein Standard, der die Struktur und den Aufbau von E-Mails und anderen Internetnachrichten festlegt.

15.2.3 Dynamischer WEB Seiten Inhalt

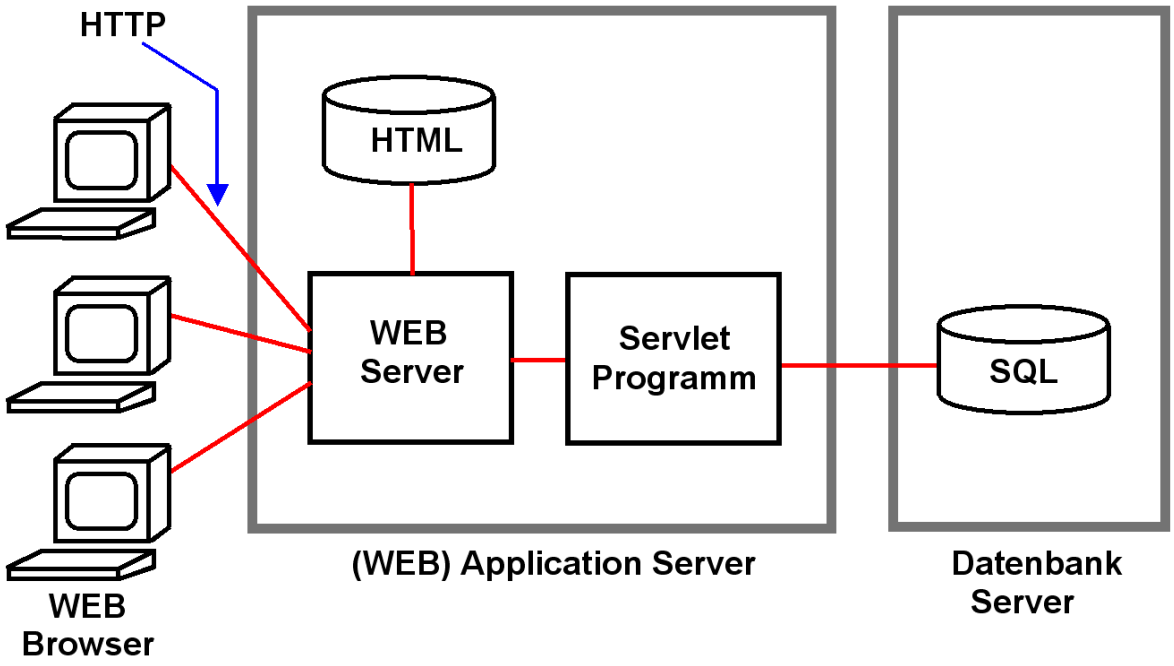


Abb. 15.2.5
Auswertung des Form Tags durch ein Servlet

Damit der Server mit den übertragenen Form Data etwas anfangen kann, enthält die übertragene Nachricht die Angabe eines Programmnamens, der mit dem Parameter ACTION spezifiziert wird. Das Programm wird aufgerufen und die übertragenen Daten werden verarbeitet. Zur Erstellung einer Antwort kann das Programm z.B. Daten aus einer z/OS DB2 Datenbank auslesen, um eine dynamische HTML Seite zu erstellen.

Der HTML Code `<FORM METHOD=POST ACTION="/servlet/HelloWorld">`
ruft ein Java Servlet mit dem Namen HelloWorld auf.

Ein Servlet Tag , z.B. `<SERVLET CODE="HelloServlet"></SERVLET>`
in einer HTML-Seite bewirkt das Ausführen eines Servlets auf dem Server.

Java Servlets sind normale Java-Klassen, die auf einem Server innerhalb einer standardisierten Laufzeit-Umgebung, der "Servlet Engine" oder des "Servlet Containers", ablaufen. Der Servlet Container beinhaltet eine normale Java Virtuelle Maschine.

- **Servlets sind vollwertige Java-Programme; sie verfügen über alle Java APIs, einschließlich JDBC (Java Data Base Connectivity) und SQLJ. Ein Applet kann im Gegensatz dazu auf keine Server-seitigen Daten zugreifen.**
- **Im Gegensatz zu CGI erfordert das Java Servlet nur "Light Weight Context Switches" implementiert über Java Threads. Daraus resultiert ein deutlich besseres Leistungsverhalten.**
- **Da das Servlet im Hauptspeicher verbleibt, können Verbindungen (Connections) zur Datenbank offen gehalten werden. Ein Servlet kann einen gemeinsamen Vorrat an Datenbankverbindungen verwalten, und diese je nach Bedarf einzelnen gleichzeitig ausgeführten Anwendungsprogrammen zuordnen.**
- **Leistungsfähiges Fehler- und Type-Checking.**

15.2.4 Servlet Container

Servlets laufen in einer Servlet-spezifischen Laufzeitumgebung, die auch als Container oder Servlet Engine bezeichnet wird. Der Servlet Container besteht im Wesentlichen aus einer Reihe von spezifischen Java Klassen, die innerhalb der gleichen JVM wie die Servlets untergebracht sind. Eine Servlet Klasse erbt mit Hilfe von „extends HttpServlet“ die Eigenschaften dieser Container Klassen.

Der Servlet Container verbessert u.a. die Servlet-Ausführungszeit und stellt dem Programmierer vorgefertigte Strukturen zur Verfügung. Servlet Container haben keine Transaktions-, Persistence- und Sicherheitseigenschaften. Ein Servlet Container ist ein Programm, das Requests für Servlets und Java Server Pages (JSP) behandelt. Der Servlet Container ist verantwortlich für:

- **Erstellung von Servlet-Instanzen,**
- **Initialisierung von Servlets,**
- **Dispatching von Requests,**
- **Verwaltung des Servlet-Kontextes für die Nutzung durch die Web-Anwendungen.**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public
class HalloWeltServlet extends HttpServlet
{

   public final static String message = "<html>\n" +
                                "<head><title>Hallo Welt</title></head>\n" +
                                "<body>\n" +
                                "<h1>Hallo Welt</h1>\n" +
                                "</body></html>\n";

   public void init()
   {
      System.out.println("In HalloWeltServlet init");
   }

   public void destroy()
   {
      System.out.println("In HalloWeltServlet destroy");
   }

   public void service(ServletRequest req, ServletResponse res)
   throws ServletException, IOException
   {
      PrintWriter  out = res.getWriter();
      out.println(message);
   }
}
```

Vererbung der Servlet Klassen

Dies ist eine Java Variable mit dem Namen „message“ .

message hat die Form eines html Programms (why not ?) .

Jedes Servlet verfügt über die Methoden init, destroy und service.

Die Methode service ist für die Bearbeitung des Servlet Aufrufs zuständig.

Der in message enthaltene HTML Code wird an den Browser gesendet

Abb. 15.2.6
Beispiel: HalloWeltServletJava

15.2.5 Java Server Pages

Java Server Pages (JSP) sind in der Java Programmiersprache geschrieben. Eine JSP ist in Wirklichkeit eine andere Darstellungsform eines Servlets.

JSPs benutzen XML-artige Tags und Scriplets um die Logik zu kapseln, die den Inhalt der Seite generiert.

Alternativ kann die Anwendungslogik woanders liegen, und die Java Server Page greift hierauf mit den Tags und Scriplets zu.

Dies ermöglicht eine Trennung der HTML Seiten-Logik vom Seitenentwurf und der Seitenwiedergabe.

Sie können den folgenden Text in einer Datei mit der .jsp extension im JSP directory speichern und mit einem Browser ansehen:

```
<html>
<head>
<title>JSP Example </title>
</head>
<body>
Hello! The time is now <%= new java.util.Date() %>
</body>
</html>
```

Die Zeichenfolgen <%= und %> schließen Java Epressions ein. Diese werden zur Run Time ausgewertet. Die Klasse new java.util.Date() ist Bestandteil des JDK. (Normalerweise würde hier ein komplexeres Präsentationslogik Programm stehen).

Bei jedem Reload der HTML Seite in den Browser wird die gültige Zeit wiedergegeben.

15.2.6 Interaktion Servlet - JSP

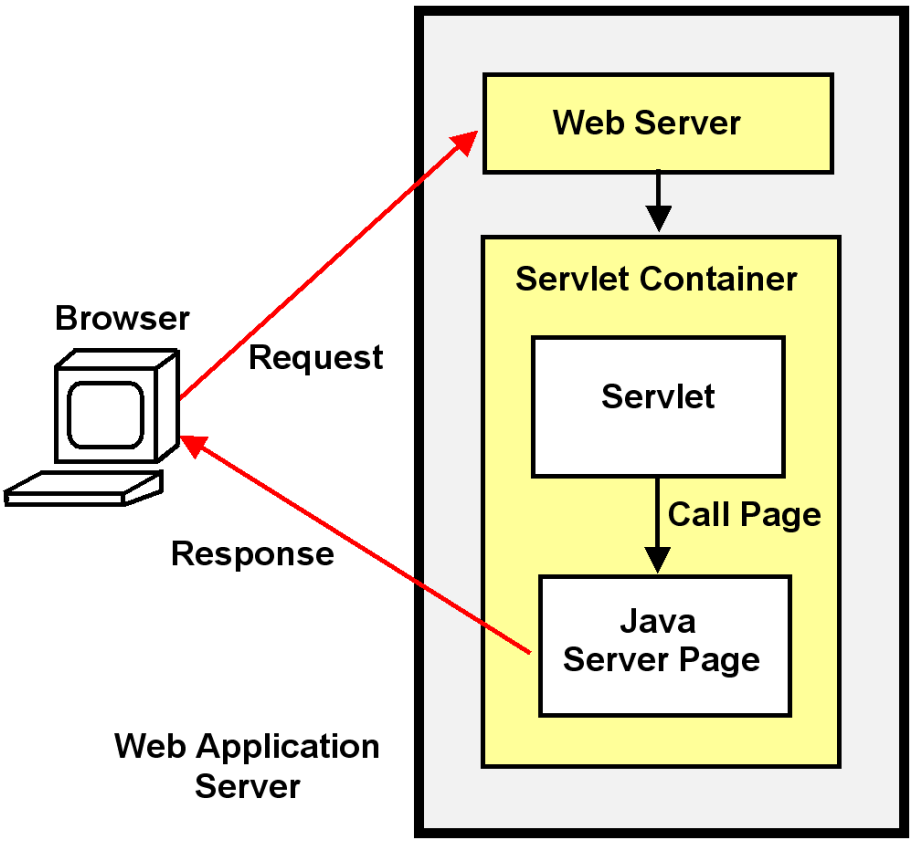


Abb. 15.2.7
Die Kombination Servlet – Java Server Page ist sehr gebräuchlich

In der Praxis ist es eher selten, dass eine JSP direkt aufgerufen wird. Statt dessen wird in der Regel ein Servlet aufgerufen, welches wiederum eine JSP aufruft.

15.2.7 Business- und Präsentationslogik

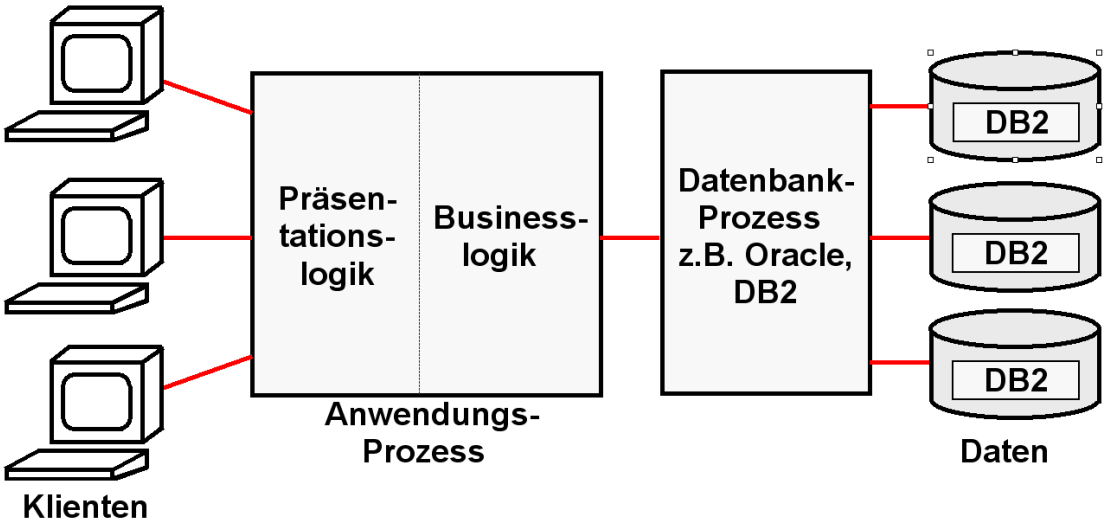


Abb. 15.2.8
Aufteilung einer Anwendung in Business Logik und Präsentationslogik

In einfachen Fällen (z.B. einige hundert Zeilen Code) kann ein Servlet die Business Logik und eine JSP die Präsentationslogik implementieren. In komplexeren Fällen ist es sinnvoll, beides in eine größere Anzahl von Java Klassen zu strukturieren. Hier bietet es sich an, den Großteil der Programmierlogik als getrennte Java Klassen zu implementieren, wobei die Servlets und JSPs lediglich Steuerfunktionen übernehmen.

Hierfür bietet sich ein Java Komponenten Modell an, die Java Beans.

Komponenten sind unabhängige, in sich abgeschlossene, wohl definierte Software Einheiten, die eine spezifische Leistung über standardisierte Schnittstellen bieten. Komponenten lassen sich mit anderen Komponenten zu größeren Einheiten zusammenfügen, die wiederum Komponenten oder eigene Anwendungen sind.

15.2.8 Java Beans

Unter Java Beans versteht man kleine Java-Programme (Klassen) mit festgelegten Konventionen für die Schnittstellen, die eine Wiederverwendung in mehreren Anwendungen (Applikationen, Servlets und Applets) ermöglichen, ähnlich wie bei Unterprogramm-Bibliotheken in anderen Programmiersprachen.

Dies ist vor allem im Hinblick auf das Software-Engineering von komplexen Programmsystemen interessant.

Dafür existiert ein eigenes Beans Development Kit BDK, das man zusätzlich zum JDK installieren kann, und eine Package java.beans, die ab Version 1.1 im JDK enthalten ist,

JavaBeans sind ein Objektorientiertes Java Komponenten Modell. JavaBeans sind Java binary parts. Sie werden häufig für visuelle Komponenten eingesetzt (etwa Buttons und Scrollbalken)

Hauptmerkmale der Java Beans sind:

- **Properties (Eigenschaften, z.B. get und set)**
- **Methoden**
- **Events (Ereignisse)**
- **Namens Konventionen**
- **Introspection (BeanInfo Klasse)**

Für unternehmensweite Anwendungen (Enterprise Applications) fehlen Schlüsseleigenschaften, z.B. Transaktionsdienste, Namensdienste und Sicherheitsdienste. Werden Java Beans hiermit angereichert, spricht man von Enterprise Java Beans.

15.2.9 Nutzung von Java Beans

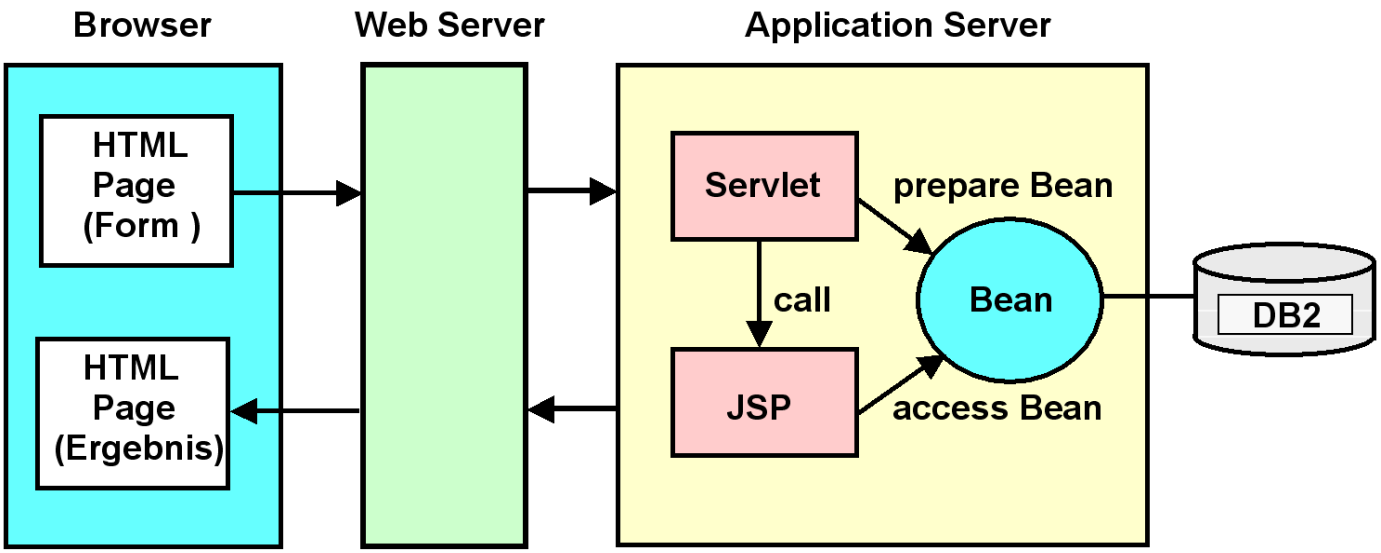


Abb. 15.2.9
Beans werden für die Business Logik eingesetzt

Der Web Server parsed eine HTML Seite und ruft ein Servlet auf.

Ein Servlet ist ein Java Programm, das Bildschirm Output in der Form einer HTML Datei produziert.

Eine JavaServerPage ist eine HTML Seite mit zusätzlichen JSP Tags.

In der Praxis: Servlets und JSP werden von verschiedenen Programmierern erstellt (Model-View-Controller Ansatz). Eine JSP ist zwar eine vollwertige Java Komponente, aber der Java Code Anteil innerhalb der JSP wird in der Regel auf ein Minimum reduziert.

Es existieren (wie für HTML Seiten) spezielle Werkzeuge für das Erstellen von JSPs, die das Hand-coding von HTML Statements automatisieren.

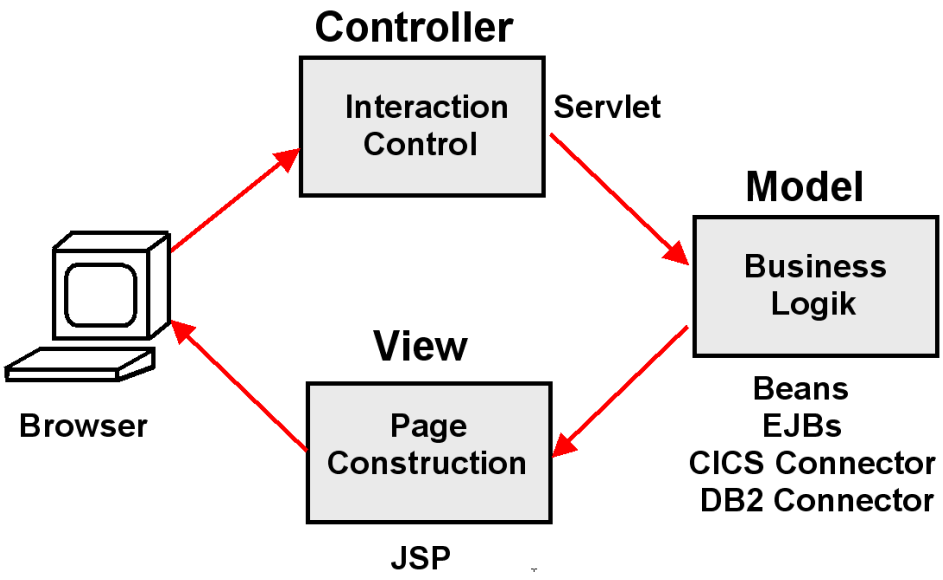


Abb. 15.2.10
MVC ist ein weit verbreitetes Programmiermodell

Als Model/View/Controller Triade (MVC) wird ein zentrisches Programmier Modell bezeichnet. Die gesamte Anwendungslogik (EJB, Servlet, JSP) läuft auf dem Server. Der Klient (thin client) braucht nur einen Browser.

Die Model/View/Controller Triade wird durch die oben dargestellte Kombination von Servlet, Java Beans und JSP implementiert.

Das "Modell" (Business Logic) ist ein Anwendungsobjekt und kapselt die Business Logik. Der "View" ist die Bildschirm Darstellung (Präsentation Logic) dieses Objektes. Der "Controller" definiert, wie die Benutzerschnittstelle auf Benutzereingaben reagiert.

Command- und Data Beans oder Enterprise Java Beans (plus häufig CICS, IMS Programme, oder Stored Procedures) sind das "Modell".

JSP´s und View Beans sind der "View"

Das Servlet ist der "Controller"

MVC entkoppelt Modell und View zur Verbesserung von Flexibilität und Re-Use. Der Entwickler der Browser Darstellung arbeitet nur mit der Java Server Page.

15.2.10 Architektur einer JSP Web Anwendung

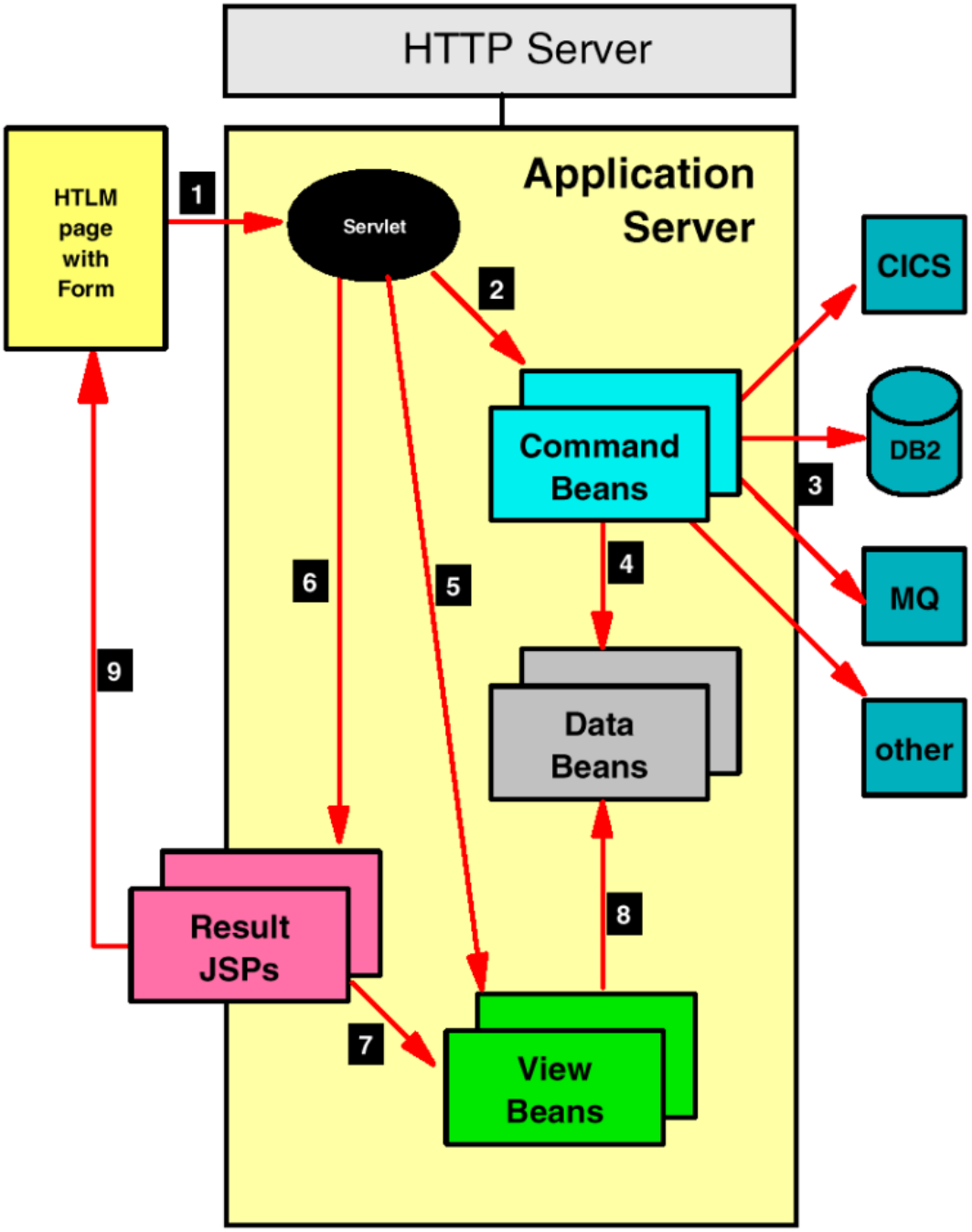


Abb. 15.2.11
Details des Model - View - Controller Ansatzes

1. HTML-Seite, die in einem vorherigen Schritt erstellt wurde, enthält eine oder mehrere HTML Forms, die ein Servlet für die Verarbeitung des nächsten Interaktion aufrufen.

2. Das Servlet erhält die Kontrolle aus der Application Server zur Validierung und Kontrolle des Ablaufes. Es ruft die Command Beans auf, welche die Geschäftslogik ausführen.

3. Command Beans steuern die Verarbeitung der Geschäftslogik. Die Logik kann in Command Beans eingebettet sein. Alternativ kann die Verarbeitung an Back-End oder Enterprise-Systeme delegiert werden, wie relationale Datenbanken, MQSeries, Transaktionen Server (CICS, IMS) usw. Command Beans rufen Datenbank-und Transaktions-Systemen über „Konnektoren" auf..

4. Ergebnisse der Command Beans (oder Back-End-Systeme) werden in Data Beans gespeichert. Data Beans können ein SQL Ergebnis oder eine CICS COMMAREA enthalten. Sie sind das Äquivalent von Unit Records.

5. View Beans definieren, wie Data Beans auf dem Bildschirm dargestellt werden sollen. Ein Servlet initialisiert die View Beans und registriert sie, so dass eine JSPs sie finden kann. View Beans enthalten die gleichen Informationen wie Data Beans, sind aber für die Verwendung durch JSP-Code optimiert.

6. Das Servlet ruft eine JSP zur Ausgabe Verarbeitung und Formatierung in Abhängigkeit von den Ergebnissen der Command Beans auf. JSPs generieren die Ausgabeinformationen für den Browser.

7. JSPs benutzen Tags zur Deklaration der View Beans, sowie um den Zugriff auf alle dynamischen Daten zu erhalten, die in der Ausgabe wiedergegeben werden müssen.

8. View Beans enthalten eine oder mehrere Data Beans und enthalten zugeschnittene Methoden, so dass die JSP Zugriff auf die Daten hat, die in den Data Beans gespeichert sind. Data Beans enthalten nicht notwendigerweise die erforderlichen Methoden, damit eine JSP auf die Daten zugreifen kann.

9. JSP assembliert die Ausgabe und sendet sie als HTML-Seite mit dynamischen Daten zurück an den Browser. In vielen Fällen enthält die Ausgabe wieder Form Tags, damit der Benutzer den Dialog mit der Anwendung fortzusetzen kann.

15.3 Enterprise Java Beans

15.3.1 Java Enterprise Edition

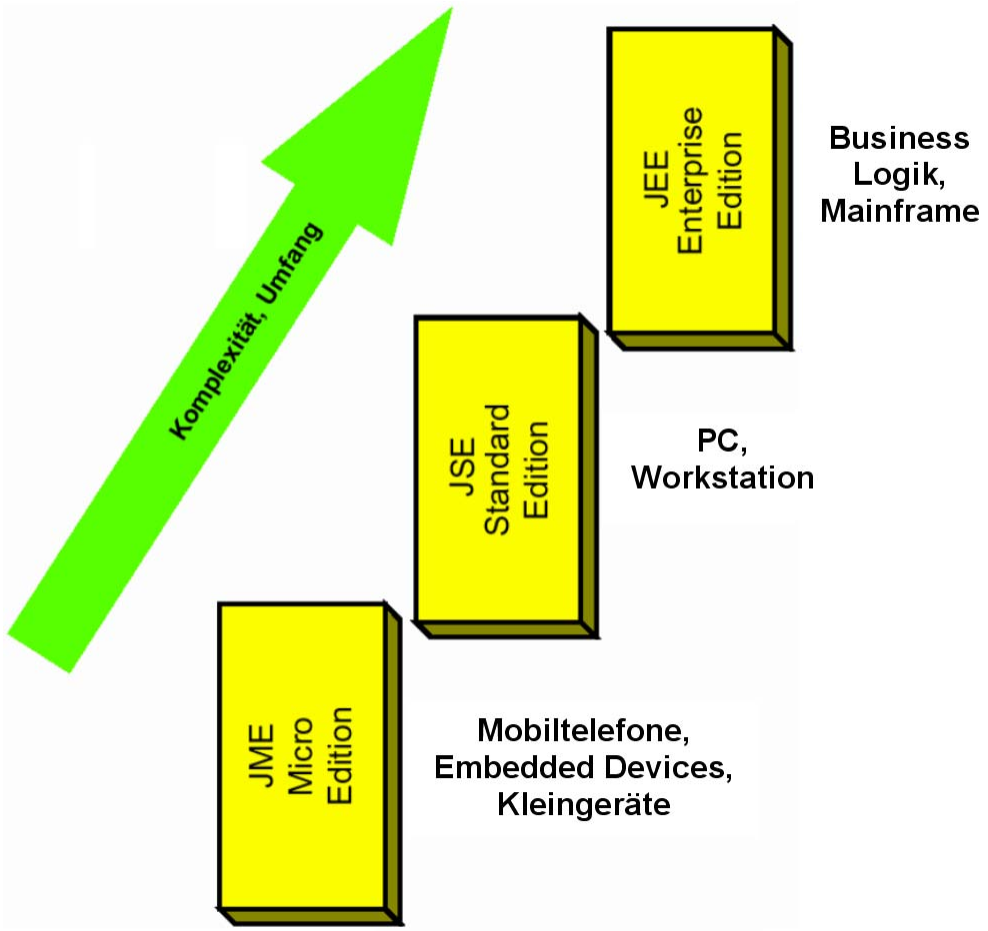


Abb. 15.3.1
Hierarchie der unterschiedlichen Java Editionen

Java existiert in drei unterschiedlichen Editionen mit einem unterschiedlichen Funktionsumfang:

JME – Java Micro Edition
JSE – Java Standard Edition

} **Frühere Bezeichnungen: J2ME, J2SE, J2EE**

Die Java Platform, Enterprise Edition, abgekürzt Java EE (JEE), oder früher J2EE, ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten verteilten Geschäftsanwendungen und insbesondere Web-Anwendungen. JEE spezifiert:

- **Enterprise JavaBeans (EJB), die Komponenten der Geschäftsanwendungen,**
- **Infrastruktur zur Ausführung von EJBs.**

JEE unterscheidet sich von JSE vor allem durch die Verfügbarkeit von Enterprise Java Beans (EJB). Enterprise Java Beans sind ein Java basiertes Server Komponentenmodell mit sehr wesentlichen Erweiterungen gegenüber einfachen Java Beans.

Die Java Enterprise Edition (JEE) beinhaltet eine Spezifikation für Verteilte Geschäftsapplikationen. Darin ist neben den Komponenten der Geschäftsapplikation, den Enterprise JavaBeans (EJB), auch die Infrastruktur zu deren Ausführung spezifiziert. Die JEE – Spezifikation zielt auf die Implementierung von mehrschichtigen Client/Server – Anwendungen ab. Mit EJBs werden Geschäftsprozesse und Geschäftskomponenten modelliert (z.B. Kunde, Auftrag, Rechnung).

JEE erschien im März 98. Das Final Release der Version 1.1 der EJB Specifikation erfolgte im Dezember 1999. Aktuell ist die Version 6.0 der JEE Spezifikation, verfügbar seit 2009. JEE Version 7 ist für 2013 vorgesehen.

Die JEE Infrastruktur beinhaltet Applikationserver mit sogenannten Containern. Spezifisch laufen Enterprise Java Beans (EJBs) in einer als EJB Container bezeichneten Laufzeitumgebung, in denen die EJBs ausgeführt werden. Der EJB Container ist anders als, und unabhängig von, dem Servlet Container. Ein Application Server, der zusätzlich zu einem Servlet Container auch einen EJB Container enthält, wird generell als JEE Server oder als Web Application Server bezeichnet.

Der JEE Server, beziehungsweise der EJB Container, interagiert mit Systemressourcen des Unternehmens (z.B. Datenbanken) und übernimmt auch die Interaktion mit geografisch verteilten Beans in anderen Servern. Weiterhin kontrolliert er die Ausführung von selbst definierten Transaktionen und handhabt Sicherheitseinstellungen.

JEE Server bieten also typische Middleware–Techniken an, und ermöglichen eine vereinfachte Komponentenprogrammierung. So sind als Teil der Infrastruktur auch die Kommunikationsformen der EJB mit den Containern und die Kommunikation zwischen den Containern vorgeschrieben. Ebenso ist die Schnittstelle des Servers zu Klienten und zu Ressourcen weit gehend reglementiert.

EJB Programmierer sollen sich größtenteils mit der Lösung der Geschäftsabläufe befassen und wiederverwendbare Komponenten produzieren. Das Idealbild ist, dass diese Komponenten dann in jeglichen nach JEE spezifizierten Servern laufen sollen. Die Wiederverwendbarkeit von Komponenten hat sich in der Praxis bisher aber nur in wenigen Teilbereichen durchsetzen können.

15.3.2 Web Application Server

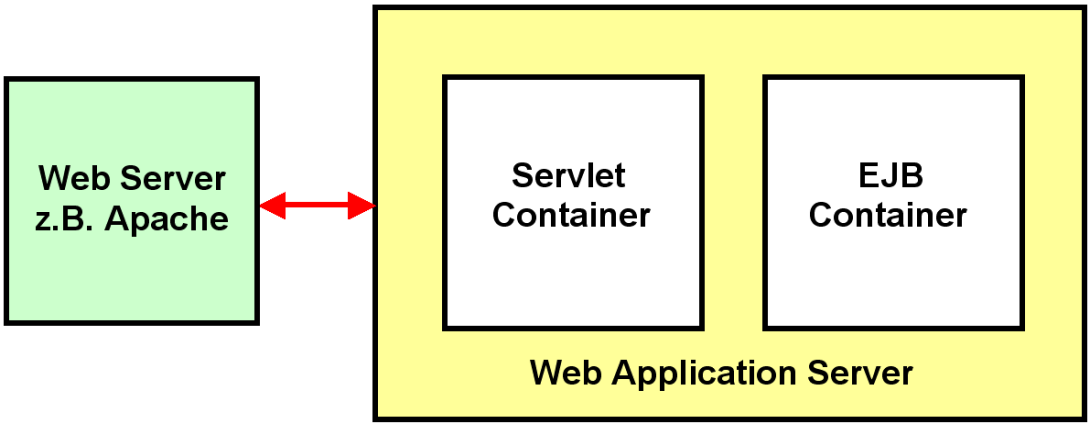


Abb. 15.3.2
Der Web Application Server implementiert den JEE Standard

Servlets benötigen für ihre Ausführung eine Servlet Laufzeitumgebung, den Servlet Container. EJBs benötigen für ihre Ausführung eine EJB Laufzeitumgebung, den EJB Container.

Ein Web Application Server ist ein Software Produkt, welches sowohl einen Servlet Container als auch einen EJB Container enthält. Häufig dient ein Servlet als EJB Client.

Der Web Server (z.B. Apache oder z/OS http Server) und der Web Application Server laufen typischerweise als getrennte Prozesse auf dem gleichen physischen Server. Weil der Begriff „Web Server“ mehrfach belegt ist, sollte statt dessen der Begriff http Server benutzt werden.

WebSphere Web Application Server (IBM), Web Logic (Oracle/Bea) und Netweaver (SAP) sind die am weitesten verbreiteten Web Application Server. Tomcat, JBOSS, GlassFish und Geronimo (Apache Foundation) sind Open Source Software Produkte mit reduzierter Funktionalität.

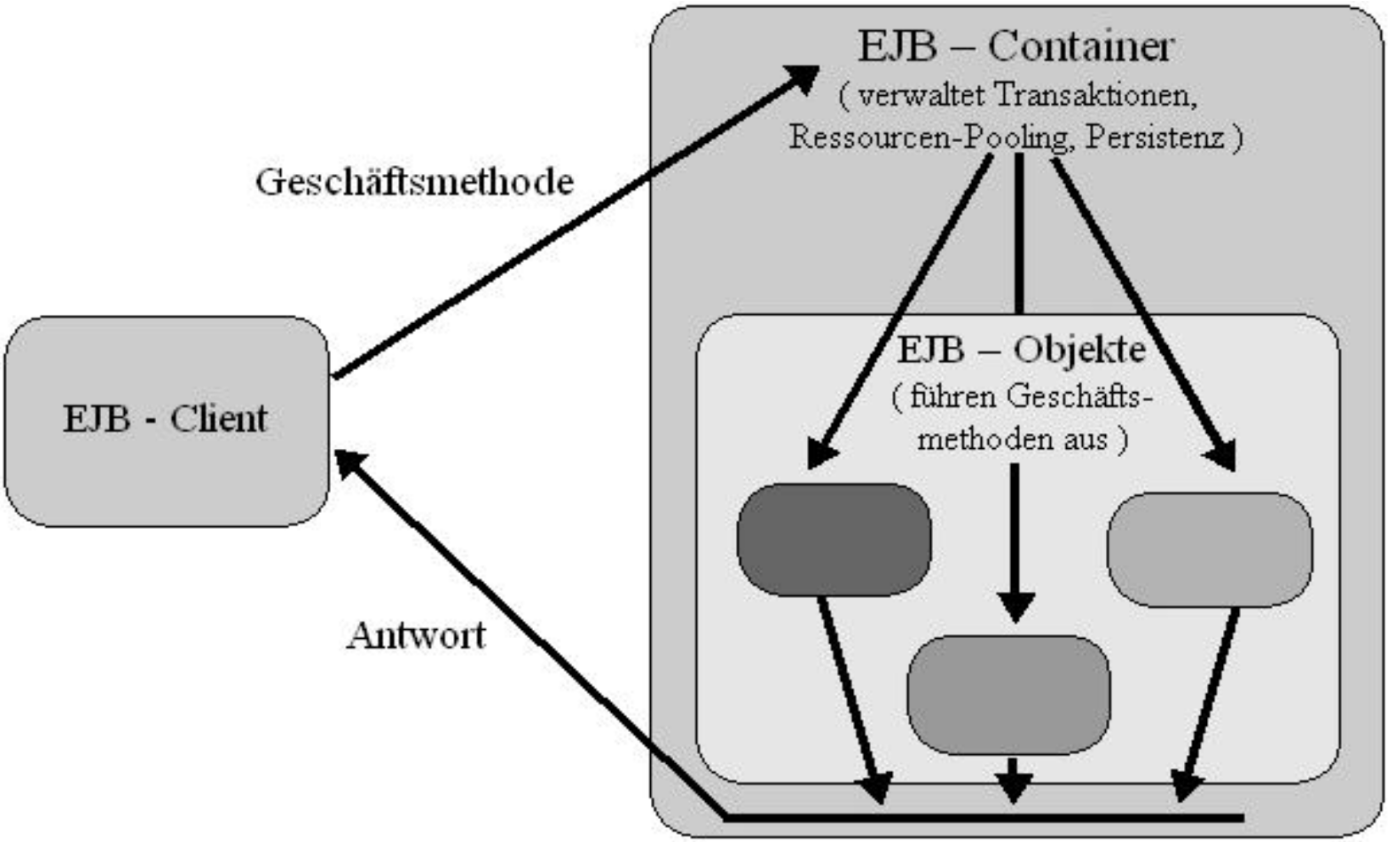


Abb. 15.3.3
Der EJB Container dient als Server in einer Client/Server Konfiguration

Der EJB Klient ist in der großen Mehrzahl der Fälle (aber nicht immer) ein anderes Java Programm außerhalb des EJB Containers. Der EJB Container enthält typischerweise zahlreiche als Enterprise Java Beans implementierten Komponenten, die gemeinsam die Business Logik darstellen.

Präsentationslogik wird in der Regel nicht mit EJBs implementiert.

15.3.3 Distributed Objects

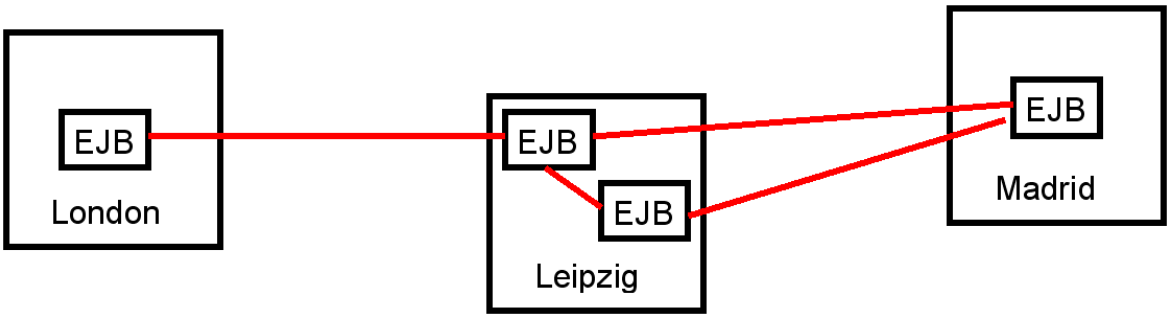


Abb. 15.3.4
EJBs sind grundsätzlich für eine geografisch verteilte Transaktionsverarbeitung ausgelegt.

Distributed Objects sind Enterprise JavaBeans, die sich auf geografisch getrennten Servern untergebracht sind.

Enterprise JavaBeans können sinnvollerweise überall dort eingesetzt werden, wo verteilte Objekte benötigt werden. Ein Beispiel ist eine Online Banking Anwendung: Ein Benutzer sitzt zu Hause und möchte sich mit allen seinen Bankkonten verbinden, gleichgültig wo und bei wem sich diese befinden, und sie alle im Zusammenhang und in einer angenehmen Benutzeroberfläche vorfinden. Die EJB – Komponentenarchitektur ermöglicht es verschiedenen Finanzinstituten, Benutzerkonten als unterschiedliche Implementierungen eines gemeinsamen Interfaces Account zu exportieren.

Da die Account Objekte zugleich EJBs sind, kann man die Transaktionsfähigkeiten des EJB Servers dazu nutzen, die Account Objekte als transaktionale Komponenten zu implementieren. Der Client kann mehrere Kontenoperationen innerhalb einer einzigen Transaktion durchführen und sie dann alle entweder mit einem Commit oder einem Rollback abschließen.

15.3.4 Persistenz

Die permanente Speicherung eines Objektes auf einem Plattenspeicher wird als Persistenz bezeichnet. Konzeptuell können Objekte in einer Objektdatenbank (z.B. POET oder Jasmine) gespeichert werden.

In der Praxis werden Objekte als SQL (oder IMS, ADABAS oder VSAM) Daten gekapselt; der Zugriff erfolgt z.B. über eine JDBC (Java Data Base Connectivity) oder SQLJ Schnittstelle.

Persistente Objekte existieren permanent außerhalb des Gültigkeitsbereichs des Programms, das sie erzeugt hat.

Persistenz wird implementiert, indem der Status (die Attribute) eines Objekts zwischen den einzelnen Programmausführungen gespeichert wird. Wenn das Objekt erneut benötigt wird, wird dieses aus seiner gespeicherten Form wieder hergestellt. Der Herstellungsprozess erzeugt ein neues Objekt, das mit dem ursprünglichen identisch ist.

Bei der Persistenz werden den gespeicherten Daten alle Objektattribute (etwa Klassenname, Feldname und Zugriffs-Modifier) zugeordnet. Ein Benutzer kann sich darauf verlassen, dass persistente Objekte Katastrophen und andere Störfälle überdauern. RAID 5, 6 oder 10 Plattenspeicherstrukturen und weitgehende Sicherheits- und Recovery-Maßnahmen in z/OS und den Enterprise Storage Servern ermöglichen dies.

15.3.5 Dienstleistungen des EJB Containers

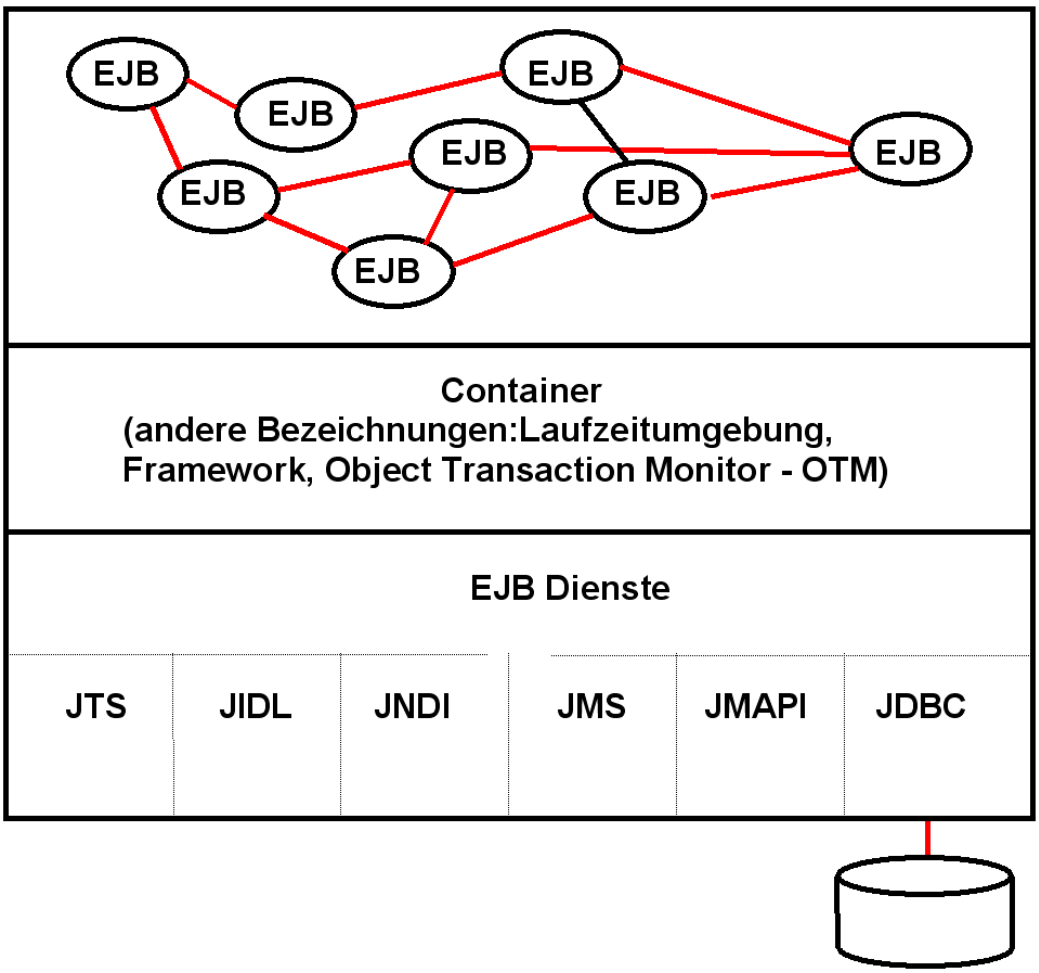


Abb. 15.3.5
Der EJB Container stellt den EJBs eine Reihe von Diensten zur Verfügung

Es existieren drei unterschiedliche Arten von EJBS, die sich gegenseitig aufrufen können :

- **Entity Beans**
- **Session Beans**
- **Message Beans**

Enterprise Java Beans sind Java Beans mit erweiterter Funktionalität. Diese Funktionalität wird von dem EJB Container als EJB-Dienste zur Verfügung gestellt und enthält unter anderem:

- **JTS** **Java Transaction Service**
- **JNDI** **Java Naming directory Interface**
- **JMS** **Java Messaging Service**
- **JDBC** **Java Data Base Connectivity**
- **JMAPI** **Java Management API**
- **JIDL** **Java interface definition language**

JTS, der Java transaction Service stellt die Funktion eines Transaction Servers zur Verfügung. Die Kommunikation zwischen den in Java implementierten Anwendungen und dem JTS geschieht über ein Protokoll namens Java Transaction API (JTA). Siehe Abschnitt 19.1.1.

JNDI, die Java Naming and Directory Interface, ist eine API für den Zugriff auf Namens- und Verzeichnisdienste. Zum Beispiel wird sie verwendet, um Datenbanken oder EJBs zu lokalisieren, die von einem Servlet benötigt werden.

JMS, der Java Message Service, ist eine API zum Aufrufen einer asynchrone Übermittlung von Nachrichten.

JDBC, die Java Database Connectivity API, greift auf Daten in bestehenden Datenbanken über eine gemeinsame Schnittstelle zu.

JMAPI, die Java Management API, definiert den Zugriff auf einen Satz von Diensten zum Verwalten von Java-Ressourcen.

JIDL, die Java Interface Definition Language, ist eine Schnittstelle zu einer Reihe von CORBA Dienstleistungen für verteiltes Rechnen.

EJBs sind Komponenten, die eine objektorientierte Sicht auf persistent gespeicherte Entitäten repräsentieren, also eindeutig identifizierbare und über Attribute beschreibbare Informationseinheiten,. Diese Entitäten sind beispielsweise in einer Datenbank gespeichert. Handelt es sich dabei um eine relationale Datenbank, so korrespondieren Entity Beans z.B. mit jeweils einer Zeile der entsprechenden Datenbanktabelle.

15.3.6 Session und Entity Beans

Session Beans implementieren die eigentliche Business Logik. Man unterscheidet zustandslose (stateless) und zustandsbehaftete (stateful) Session Beans.

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

Im Gegensatz dazu müssen einer zustandslosen Session Bean bei jedem Aufruf alle Informationen als Parameter übergeben werden, die für die Abarbeitung dieses Aufrufs benötigt werden.

Zustandslose Session Beans können von mehreren Klienten gleichzeitig benutzt werden. Bei den zustandsbehafteten Session Beans muss für jeden Klienten eine eigene Kopie angelegt werden.

Entity Beans repräsentieren für den Klienten eine objektorientierte Sicht auf einen Datensatz. Sie erlauben im Gegensatz zu Session Beans auch Mehrbenutzerbetrieb: Auf eine Instanz einer Entity Beans können gleichzeitig mehrere Benutzer zugreifen. Der Container, in den Entity Beans während der gesamten Lebensdauer eingebettet sind, stellt dazu Mechanismen bereit, um z.B. Sicherheit, Transaktionskonsistenz und Parallelität sicherzustellen.

Da Entity Beans nicht an einen einzelnen Client gebunden sind, endet ihre Lebensdauer nicht nach dem Beenden einer Client - Verbindung.

Im Gegensatz zu Session Beans können bzw. müssen sie sogar nach einem Systemausfall automatisch wiederhergestellt werden. Ihre Existenz ist an das Vorhandensein der mit ihnen verbundenen Daten gebunden.

Eine Entity Bean speichert Daten (ihre Variablen) persistent. Die Erstellung einer neuen Entity Beans erzeugt z.B. automatisch eine neue Zeile in einer Datenbank und fügt die bei der Erstellung mit übergebene Daten der Datenbank hinzu. Wird die EJB gelöscht, wird automatisch der mit dieser EJB verbundene Datensatz aus der Datenbank gelöscht.

15.3.7 Session Fassade Architektur

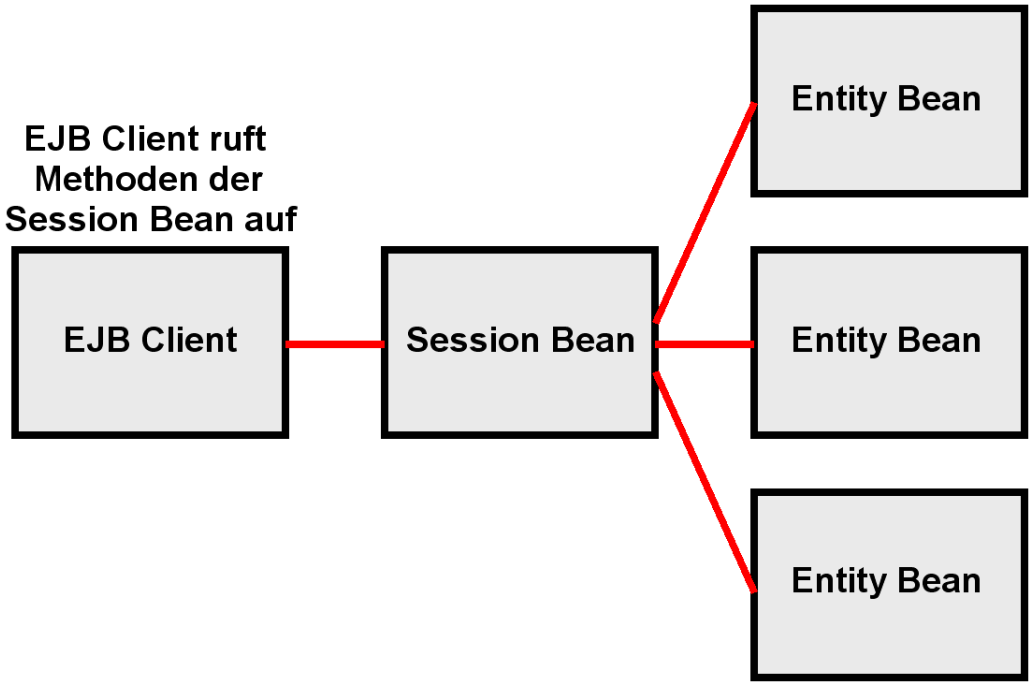


Abb. 15.3.6
Die Session Fassade benutzt Session- und Entity Beans

In der ursprünglichen Konzeption der J2E Architektur werden Session Beans und Entity Beans gemeinsam benutzt, um die Business Logik in Java zu implementieren. Dies geschieht mit Hilfe der „Session Fassade Architektur. Hierbei führen Session Beans Geschäftsprozesse (Business Logik) aus, und manipulieren persistente Objekte (Daten), die als Entity Beans modelliert werden. Dabei ging man davon aus, dass Entity Beans in objektorientierten Datenbanken (z.B. Poet, Jasmine) gespeichert würden. Dies hat sich in der Praxis aber nicht durchgesetzt.

Weitere Probleme mit dem Session Fassade EJB Architektur Modell sind:
- **Entity Beans sind verteilte Objekte, die von beliebigen Klienten aufgerufen werden können**
- **Entity Beans sind transaktionsfähig**

Beides ist bei einer Session Fassade nicht erforderlich und damit unnötiger Overhead.

Manche Anwendungen haben deshalb ausschließlich Session Beans eingesetzt und die Persistenz mit eigenem Code implementiert. Um dies zu erleichtern, stellten einige Hersteller (miteinander inkompatible) Persistence-Frameworks zur Verfügung. Weit verbreitet sind die „Java Data Objects“ (JDO) der Apache Foundation als Ersatz für die Entity Beans. JDOs sind reguläre Java Klassen, die über einen Persistence Encapsulation Mechanismus verfügen.

Aus den JDOs entstand die Java Persistence API (JPA) als Teil des EJB 3.0 Standards. Mit der Einführung des EJB 3.0 Standards wurden Entity Beans deprecated (sind obsolet geworden). Stattdessen sollen Persistent Entities verwendet werden. Eine Persistence Entity ist ein Plain Old Java Object (POJO), das üblicherweise auf eine einzelne Tabelle in der relationalen Datenbank abgebildet wird. Instanzen dieser Klasse entsprechen hierbei den Zeilen der Tabelle. Persistence Entities können je nach Designvorgabe als einfache Datenhaltungs-Klassen (vergleichbar mit einem Struct in C) realisiert werden oder als Business-Objekte inklusive Business-Logik.

15.3.8 Annotation

Als Annotation wird ein Sprachelement bezeichnet, das die Einbindung von Metadaten in den Java Quelltext erlaubt. Dieses Element wurde mit der Version Java5.0 eingeführt (verfügbar seit 2004 als Nachfolger der Java Version 1.4).

Annotationen beginnen mit einem @-Zeichen. Daran schließt sich ihr Name an. Optional kann eine kommagetrennte Parameterliste folgen, die in runden Klammern eingefasst wird. Beispielsweise markiert die Annotation im folgenden Quelltextausschnitt die Klasse A als überholt (deprecated):

```
/**
 * @deprecated  Die Klasse A wurde mit Version 10.3 durch die Klasse ANeu
ersetzt.
 */
@Deprecated
public class A {}
```

Eingesetzt werden Annotationen unter anderem im Java-EE Umfeld, um Klassen um Informationen zu erweitern, die vor der Einführung von Java5.0 in separaten Dateien hinterlegt werden mussten. Prominente Beispiele sind Home- und Local Interfaces sowie Deployment-Deskriptoren.

15.3.9 Java Naming and Directory Interface

Eine Methode einer Java Klasse wird aufgerufen, indem man ihren Namen spezifiziert. Die Klasse und ihre Methode sind irgendwo mit irgendeiner Adresse gespeichert. Wenn wir einen Server im Internet aufrufen, verwenden wir den Internet Domain Name Service (DNS). Wir rufen einen Server mit einem Namen auf, z.B. leia.informatik.uni-leipzig.de. DNS übersetzt den Namen in eine Internet Adresse, z.B. 139.18.4.30.Für komplexere Vorgänge benutzen wir einen Verzeichnisdienst (directory service) an Stelle eines Namensdienstes (name service) wie DNS. LDAP (Light Weight Directory Access Protocol) ist ein weit verbreiteter Verzeichnisdienst.

Wenn eine Methode einer Java Klasse eine Methode einer zweiten Java Klasse aufruft, muss sie dessen Lokation (Adresse) kennen. Das kann schwierig sein, wenn die zweite Klasse sich irgendwo im Netz befindet.

Lookup (englisch für „Nachschlagen“) ist der Vorgang, mit dem die Adressen von benannten Objekten ermittelt werden.

Java Klassen greifen über eine standardisierte Schnittstelle auf einen Verzeichnisdienst zu, um die Lokation einer entfernten Klasse und deren Methoden zu finden. Diese Schnittstelle ist die Java Naming and Directory Interface (JNDI).

JNDI ist kein Verzeichnisdienst, sondern eine Schnittstelle (API) zu einem Verzeichnisdienst. Die Schnittstelle ist dabei unabhängig von der tatsächlichen Implementierung. Es ist die Aufgabe des Herstellers eines JEE Servers (z.B. IBM WebSphere oder Oracle Weblogic), diesen mit einem Verzeichnisdienst auszurüsten, auf dem mittels JNDI zugegriffen werden kann. In vielen Fällen wird hierfür LDAP benutzt. JNDI ist eine sogenannte Service Provider Interface (SPI), das Herstellern eines Web Application Servers erlaubt, eigene Lösungen in ihren JEE Server einzubinden.

Die API enthält:

- **einen Mechanismus zur Bindung (binding) eines Objekts an einen Namen**
- **Methoden für den Abruf von Informationen anhand eines Namens**
- **ein Ereigniskonzept, über das Klienten über Änderungen informiert werden**
- **spezielle Erweiterungen für LDAP-Funktionalitäten**

In JNDI werden die Namen hierarchisch angeordnet. Namen sind üblicherweise Strings wie „com.mydomain.MyBean“, können aber auch beliebige Objekte sein, die die Schnittstelle javax.naming.Name implementieren. Im Namens- bzw. Verzeichnisdienst ist für jeden Namen entweder das ihm zugeordnete Objekt selbst gespeichert oder eine JNDI-Referenz auf das zugeordnete Objekt. Die Programmierschnittstelle von JNDI („JNDI API“) definiert, wo nach dem Objekt zu suchen ist.

JNDI erlaubt die Nutzung praktisch aller Arten von Namens- und Verzeichnisdiensten, insbesondere von:

- **LDAP**
- **DNS**
- **NIS**
- **CORBA Namensdienst**
- **Dateisystem**

In der Praxis wird JNDI vor allem dazu benutzt, im Rahmen von Java-Enterprise-Anwendungen verteilte Objekte in einem Netzwerk zu registrieren und sie für Remote-Aufrufe (RMI) weiteren Java-Programmteilen zur Verfügung zu stellen.

15.4 Schnittstellen

15.4.1 Struktur des Web Application Servers

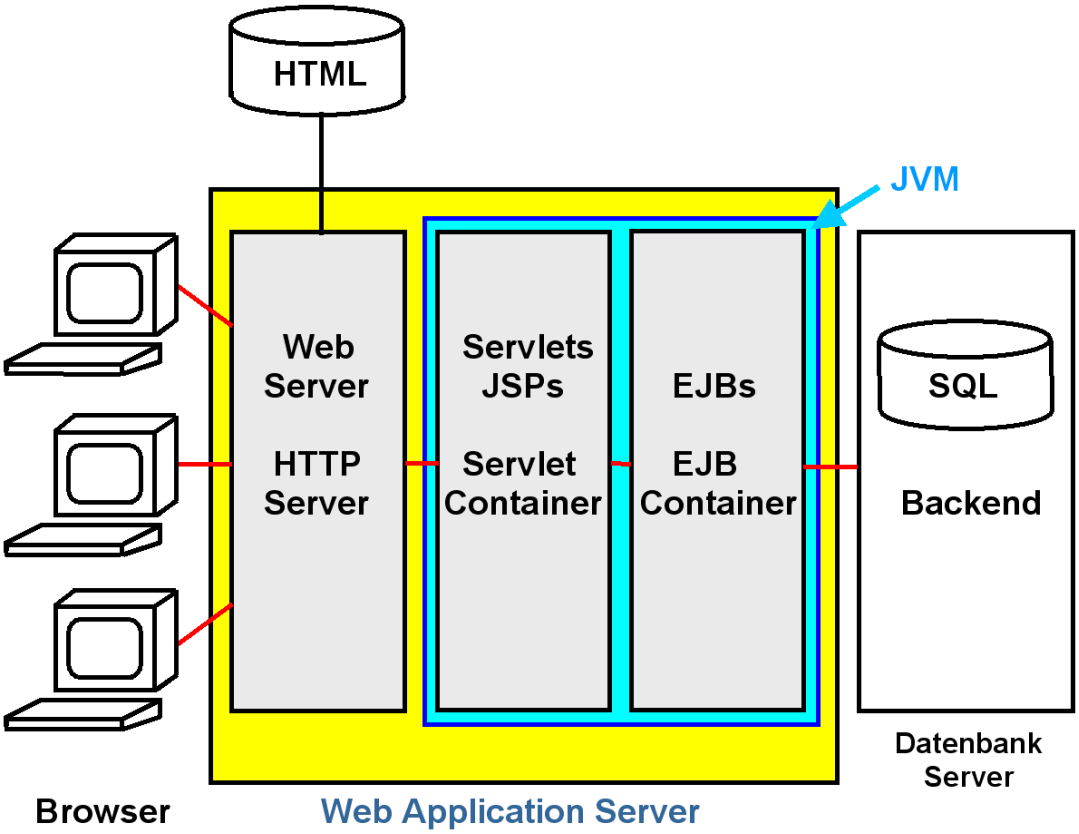


Abb. 15.4.1
Struktur des Web Application Servers

IBM bezeichnet seinen JEE Server als WebSphere Web Application Server (WAS). Der HTTP Server kann ein getrennter Prozess sein.

Servlet Container und EJB Container laufen in einer gemeinsamen Java virtuellen Maschine (JVM). Dies reduziert den Kommunikationsaufwand. Servlet Klassen können EJB Klassen direkt aufrufen. Bei getrennten JVMs erfolgt die entsprechende Kommunikation mit Hilfe des RMI oder RMI/IIOP Protokolls (Kapitel 16).

Der Web Application Server enthält weitere Elemente für Administration und Datenbankzugriff.

Der Web Application Server ist ein Prozess, der normalerweise in seinem eigenen virtuellen Adressenraum läuft. Er besteht aus mehreren Komponenten:

1. **Der Web Server (HTTP Server) ist vielfach Apache.**
2. **Der Java Application Server unterhält u.a. eine Java Virtuelle Maschine**
3. **Der Servlet Container (Servlet Engine) ist eine Java Laufzeit Umgebung (runtime component) für die Ausführung von Servlets und Java Server Pages.**
4. **Der EJB Container ist eine Laufzeit Umgebung für die Ausführung von deployed Enterprise Java Beans.**

Apache ist der am weitesten verbreitete http Server. Apache läuft unter AIX, HP-UX, Linux, Solaris, Windows, und z/OS. Apache wird von einer Open Community von Entwicklern unter der Leitung der Apache Software Foundation kontinuierlich verbessert.

Der IBM HTTP Server basiert auf dem Apache http Server. Das Download ist kostenlos. Der IBM HTTP Server ist ebenfalls ein Teil der IBM WebSphere Application Server Distribution Package.

Neben Apache existieren zahlreiche weitere http Server. Am bekanntesten ist der Microsoft Internet Information Server (IIS), vorgesehen für eine Benutzung mit Microsoft Windows. Dies ist nach Apache der zweitwichtigste http Server.

Der IBM Lotus Domino Server ist ein spezialisiertes Software Product mit einer eigenen Datenbank (Notes Storage Facility). Der Lotus Domino HTTP Server wird gelegentlich auf Mainframes eingesetzt.

Daneben existieren zahlreiche weitere HTTP Server Produkte.

In der Vergangenheit war ein Web Server immer ein Server, der eine URL auswerten und statische HTML Seiten an einen Klienten (Browser) zurückliefern konnte. Bei der Einführung von CGI und Servlets erfolgte eine Ergänzung (Plug-in), um beim Parsen von CGI oder Servlet Tags das entsprechende CGI oder Servlet Programm aufrufen zu können.

Später erfolgte eine Begriffsänderung (Begriffsverwirrung). Man begann komplexe WWW Anwendung unter Nutzung von Servlets, JSPs und EJBs zu erzeugen, sowie viele unterschiedliche derartige Anwendungen auf dem JEE Server zur Verfügung zu stellen. Zu einer vollständigen Anwendung (z.B. Banküberweisung, Hotelreservierung, Bestellung beim Otto Versand) besteht diese neben Servlets, JSPs und EJBs zusätzlich auch aus einer Reihe von statischen HTML Seiten. Man fasst diese anwendungsspezifischen statischen HTML Seiten mit den Servlets und JSPs zu einer Einheit zusammen, und nennt diese Einheit Web Server, Web Engine oder Web Container. Den bisherigen Web Server bezeichnet man zur Unterscheidung als http Server. Der http Server verwaltet alle die HTML Seiten, die nicht einer Anwendung spezifisch zugeordnet werden können.

Leider ist diese Zweideutigkeit bis heute geblieben. Es ist deshalb sinnvoll einen Server für statische HTML Seiten (wie z.B. Apache) grundsätzlich als http Server zu bezeichnen.

15.4.2 Web Server Plug-in

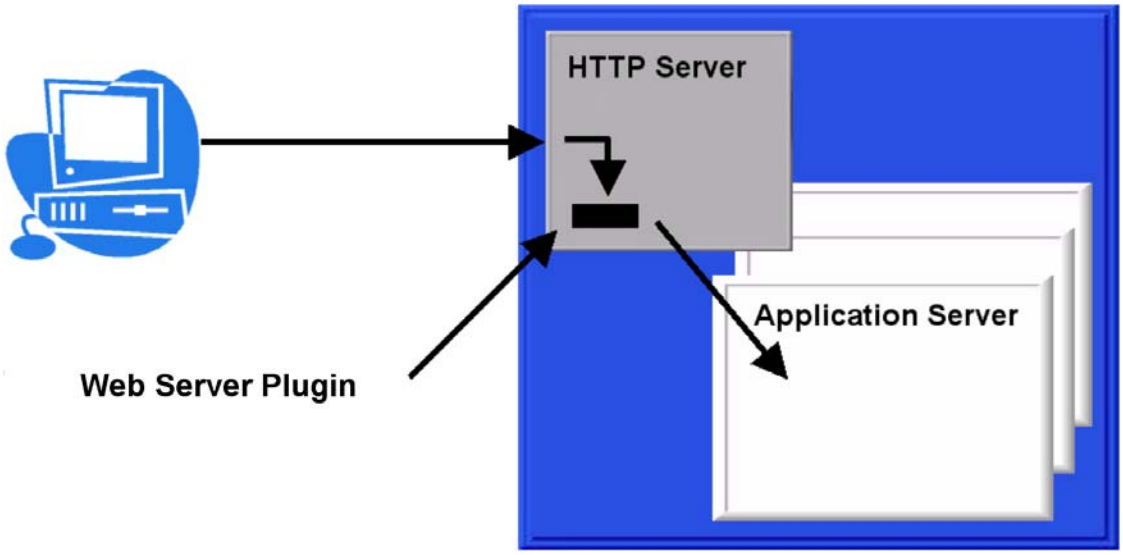


Abb. 15.4.2
Rolle des http Server Plug-ins

Es besteht der Wunsch der Benutzer von JEE Servern wie Weblogic, WebSphere Web Application Server (WAS) und Netweaver, mit unterschiedlichen http Server Produkten (z.B. Apache, IIS) zusammen arbeiten zu können. Deswegen enthält eine JEE Server Distribution mehrere Interface Komponenten, die eine Verbindung mit unterschiedlichen http Servern ermöglichen. Diese Interface Komponenten werden als „Plug-in“ bezeichnet.

Die WebSphere Web Application Server Distribution enthält z.B. Plug-in’s für Apache, den IBM http Server, für IIS und für den Lotus Domino http Server. Bei der Installation wird das Plug-in in den http Server integriert.

Die ursprüngliche WebSphere Application Server Standard Edition für OS/390 (verfügbar in 1999) lief in dem gleichen Adressenraum wie der http-Server.

Der Web Application Server war als Erweiterung zu einem http-Server gedacht, bestehend aus 2 Hauptkomponenten:

- **Ein Plug-in für den Web-Server (http-Server), welcher die Anforderung an den tatsächlichen Application Server weiterreicht, und**
- **Der Application Server selbst.**

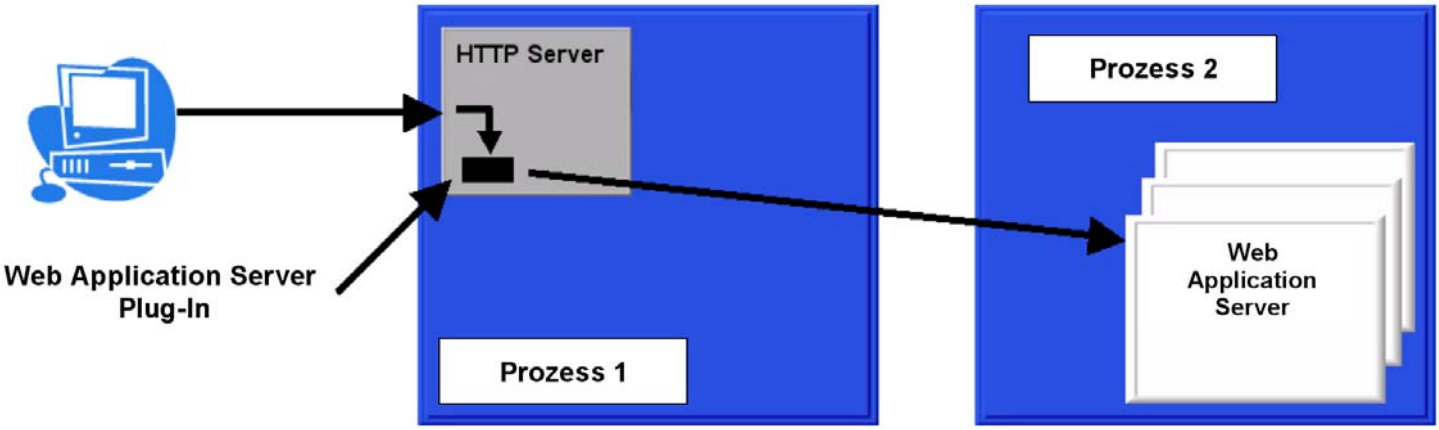


Abb. 15.4.3
Plug-in integriert in den http Server

Heute läuft ein Web Application Server als eigener Prozess, in der Regel auf der gleichen Maschine wie der http-Server (Web-Server). Es könnte auch auf einem anderen Rechner laufen. Es können mehrere Application Server Kopien als unabhängige Prozesse konfiguriert werden.

Das Webserver Plug-in ist ein Teil der Web Application Server Software-Package, wird aber in dem http-Server-Adressraum installiert. Es kann mit dem Application Server über mehrere Protokolle kommunizieren, abhängig von dem Application Server Hersteller. IBM WebSphere WAS, einschließlich der z/OS-Version verwendet HTTP und HTTPS.

Dies ist der Grund dafür:

Ein WebSphere WAS Software Package enthält den IBM HTTP Server-Code, der als Standard-Option installiert ist. Sie kann jedoch konfiguriert werden, einen anderen Web Server zu benutzen, z. B. Microsoft IIS. Damit IIS auf den WebSphere WAS zuzugreifen kann, muss der mit WebSphere mitgelieferte Plug-in in dem IIS-Adressen Raum installiert werden.

Wenn Anforderungen in dem HTTP Server eintreffen, entscheidet die http-Server-Konfigurationsdatei (httpd.conf), ob die Anfrage an den Plug-in code weitergereicht werden soll.

Das Plug-in enthält eine eigene Konfigurationsdatei. Diese entscheidet, an welchen von potentiell mehreren Application Servern die Anforderung weiter gereicht wird.

Abb. 15.4.4 und 15.4.5 zeigen zwei mögliche Web Application Server Konfigurationen.

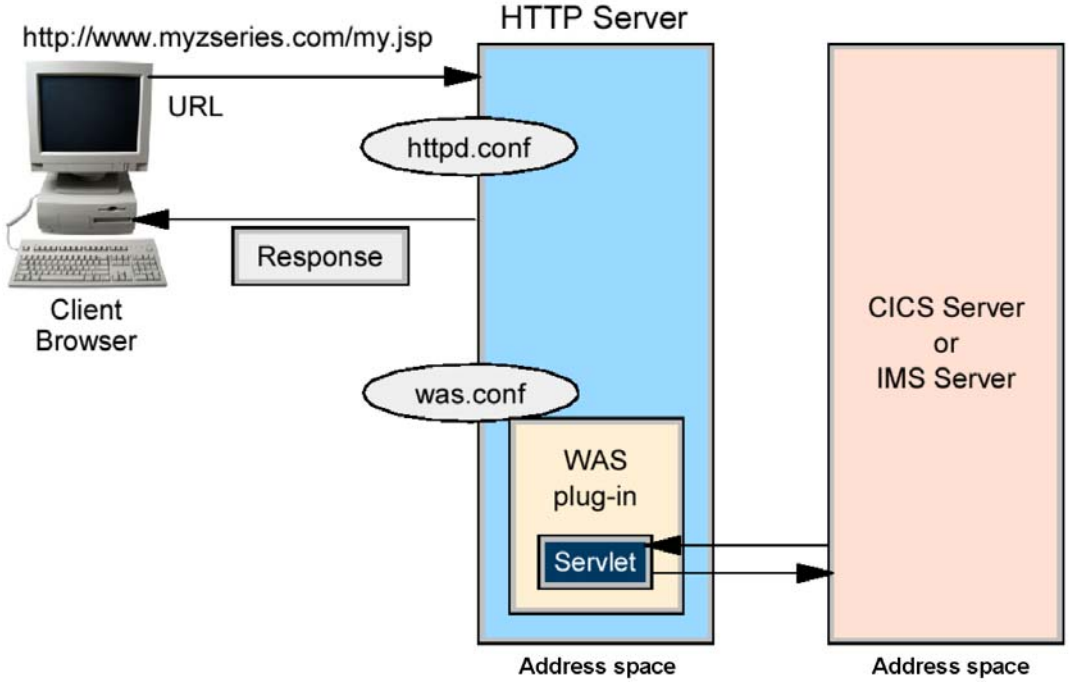


Abb. 15.4.4
Kein EJB Container

Diese Abbildung zeigt eine einfache Konfiguration, in der kein JEE-Server benötigt wird. Der Servlet Container befindet sich in dem gleichen Adressenraum wie der http Server. Das Servlet kann CICS, IMS oder DB2 über JDBC aufrufen. Allerdings wird eine Kodierung von Geschäftslogik innerhalb des Servlets nicht empfohlen.

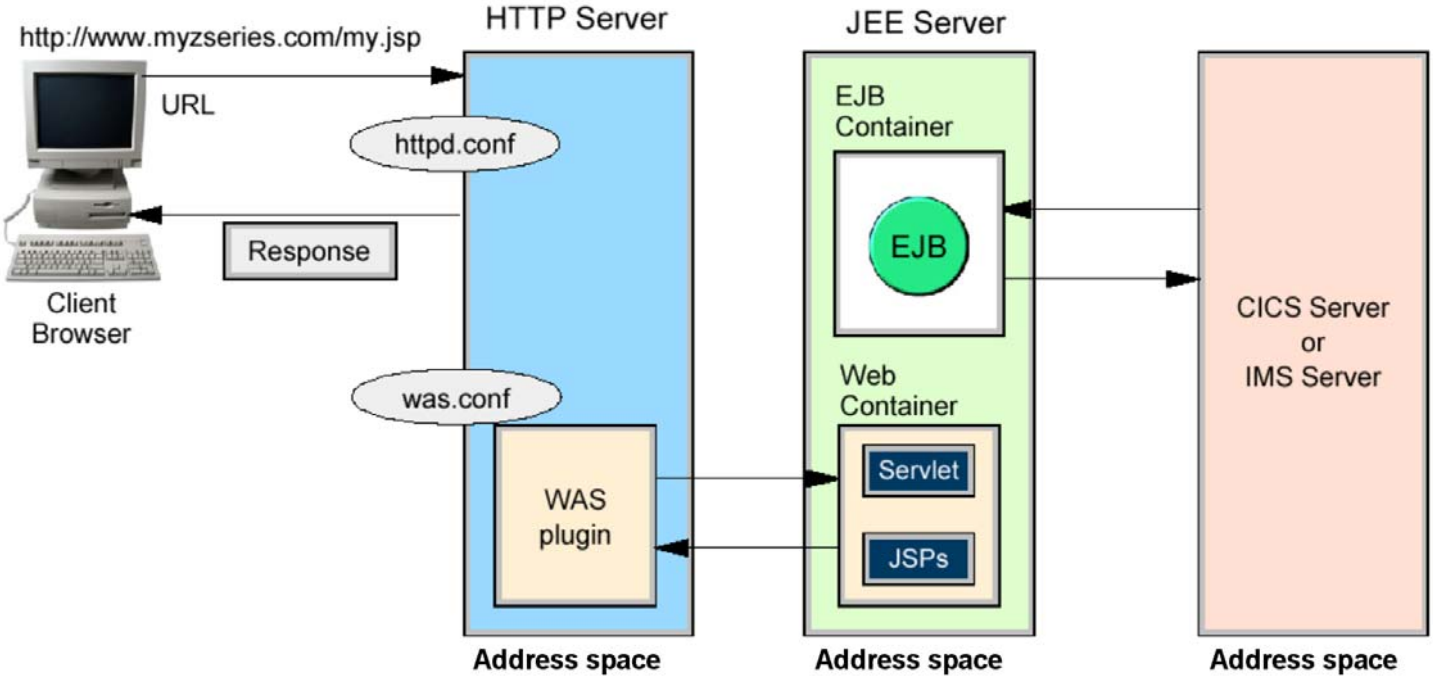


Abb. 15.4.5

Sie können auch das WebSphere Plug-in benutzen, um Servlets remote in einem „Web-Container“ auszuführen. Dies ermöglicht es, Servlets und EJBs in dem gleichen Adressraum auszuführen, so dass keine Remote EJB Aufrufe (über das RMI-Protokoll) benötigt werden.

Web Container ist eine andere Bezeichnung für Servlet Container. Der Name rührt daher, dass in den Web Container auch anwendungsspezifische statische HTML Seiten untergebracht werden können.

15.4.3 Ablaufsteuerung einer JEE Server Anwendung

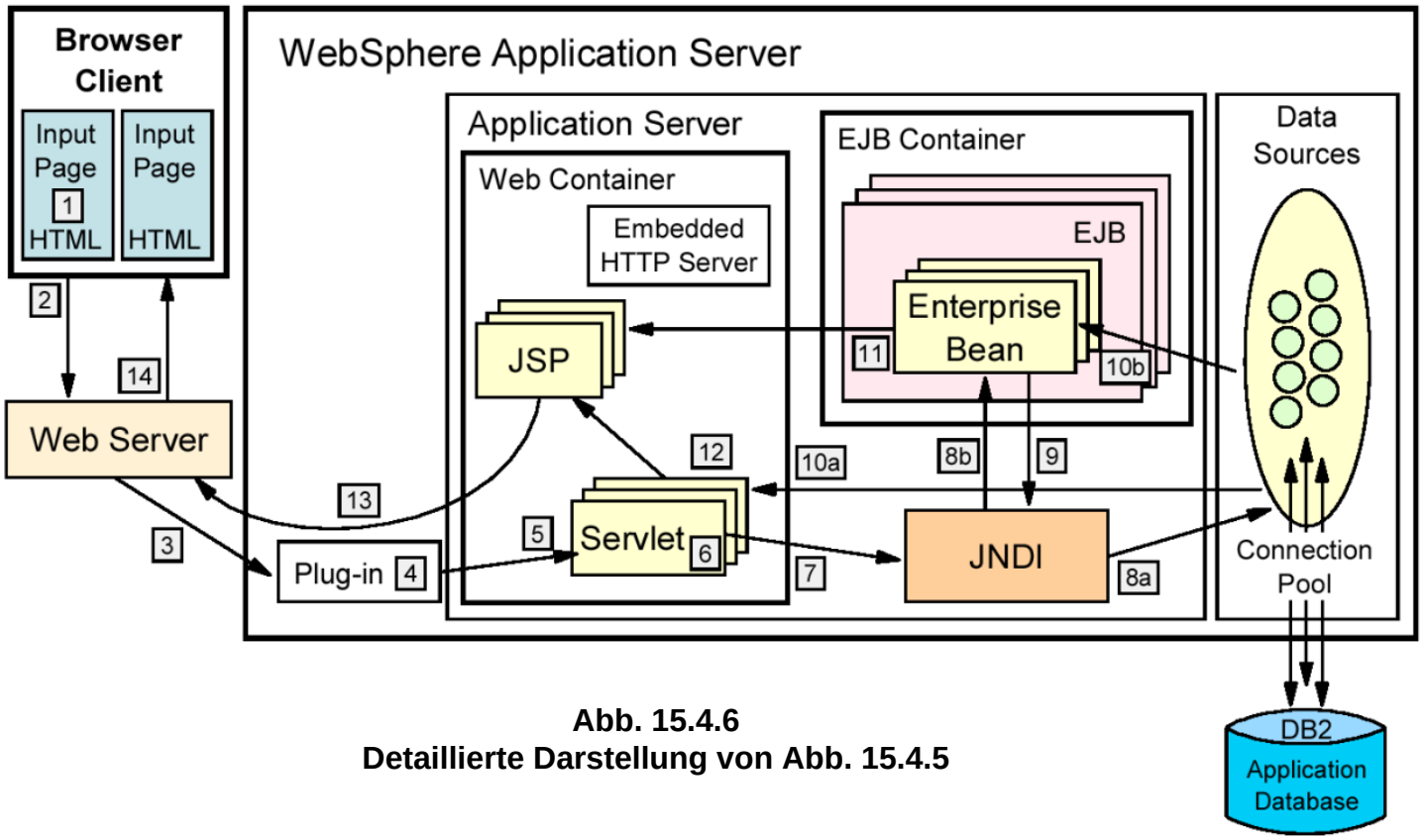


Abb. 15.4.6
Detaillierte Darstellung von Abb. 15.4.5

Der typische Ablauf der Ausführung einer JEE Server Anwendung ist wie folgt:

1. **Ein Web-Client ruft eine URL in seinem Browser auf (Input Page).**
2. **Die Anfrage wird über das Internet an den Web Server (http Server) geroutet.**
3. **Der Web-Server leitet die Anfrage an das Web Server Plug-in weiter.**
4. **Das Web Server Plug-in prüft die URL, und wählt einen Server aus, um die Anfrage zu bearbeiten.**
5. **Ein Stream wird erzeugt. Ein Stream ist eine Verbindung mit einem Web-Container. Es ist möglich, eine Verbindung (Stream) über eine Anzahl von Anfragen aufrecht zu erhalten. Der Web-Container erhält die Anfrage und reicht sie an das richtige Servlet weiter.**
6. **Wenn die Servlet-Klasse nicht geladen ist, lädt der dynamische Klassenlader das Servlet (Servlet init() Methode), und ruft dann die doGet() oder doPost() Methode auf.**
7. **JNDI wird benutzt, um die Lokation entweder der Data Source (Datenbank, VSAM) oder der EJBs zu finden, die von dem Servlet benötigt werden.**
8. **Je nachdem, ob eine Data Source angegeben ist oder eine EJB angefordert wird, directet JNDI das Servlet zu:**
   - **der entsprechende Datenbank. Hierfür wird eine bereits existierende Verbindung von dem Connection Pool zugeordnet.**
   - **dem entsprechenden EJB-Container, welcher dann die EJB instanziiert.**
9. **Wenn die EJB eine SQL-Transaktion beinhaltet, benutzt sie JNDI zum Nachschlagen der Data Source.**

10. Die SQL-Anweisung wird ausgeführt, und die aufgerufenen Daten werden entweder zu dem Servlet oder der EJB zurück gesendet.
11. Data Beans werden erstellt und im Falle einer EJB an eine JSP weiter gereicht.
12. Das Servlet sendet Daten an die JSP.
13. Die JSP generiert eine HTML Seite, die über das Plug-in an den Web-Server weiter gereicht wird.
14. Die Web-Server sendet die Output-Seite (Ausgabe HTML) an den Browser. Diese wird dort zur Input Page für den nächsten Schritt.

15.4.4 Web Application Server als Tranaktionsmonitor Front End

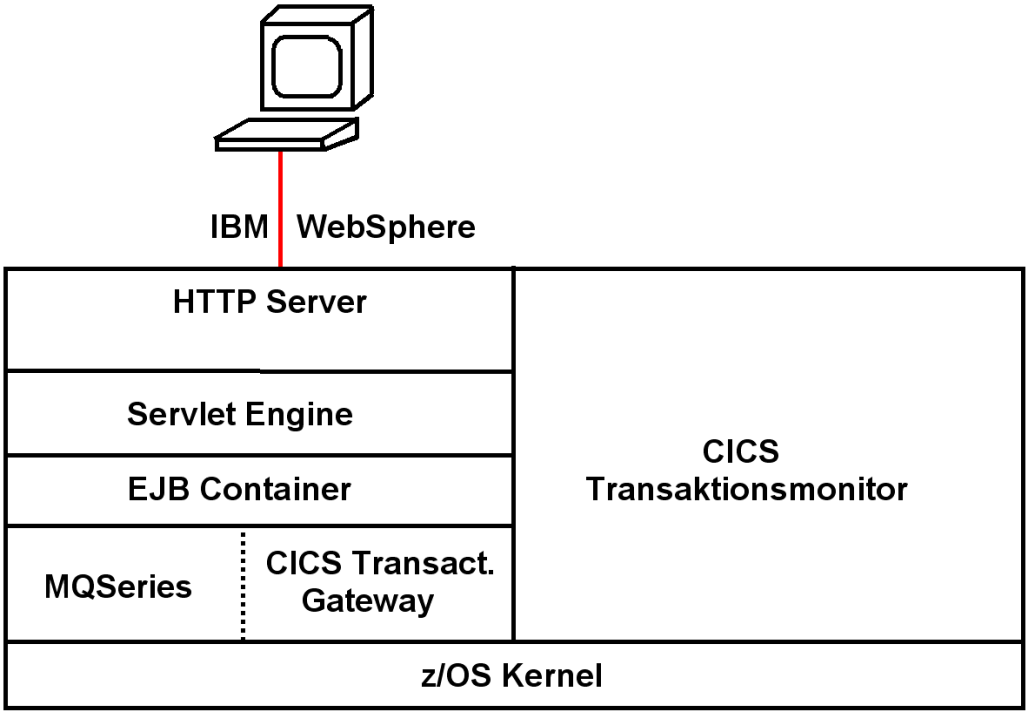


Abb. 15.4.7
WebSphere und CICS laufen unter z/OS in getrennten Adressenräumen

Obwohl EJBs Transaktionseigenschaften haben, und die gesamte Business Logik implementieren könnten, setzt man in vielen Fällen einen traditionellen Transaktionsmonitor wie CICS für den Kern der Business Logik ein.

15.4.5 Message Driven Bean

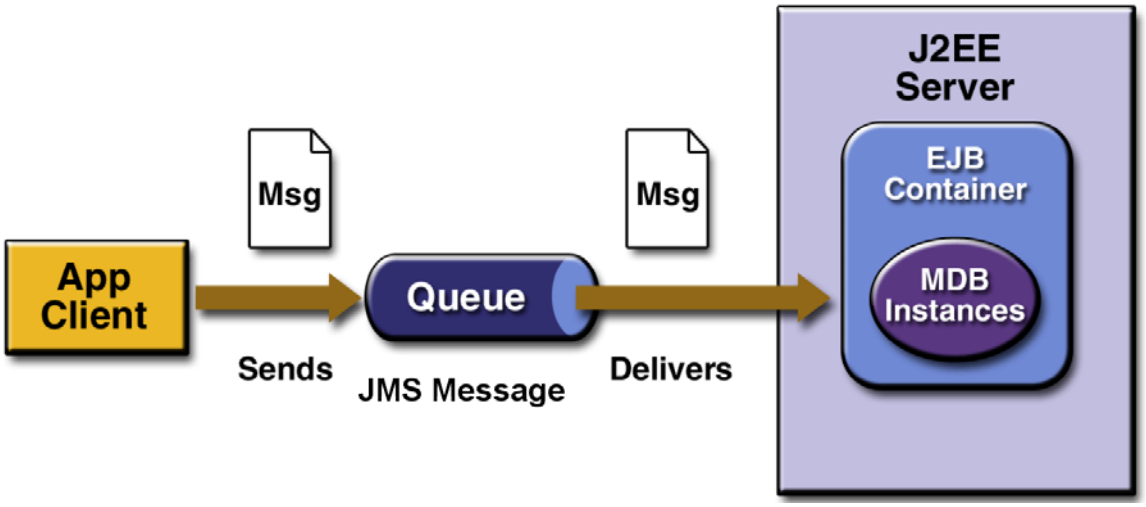


Abb. 15.4.8
WebSphere benutzt MDBs für die Java Implementierung von MBQ

Message based Queuing (MBQ) (Band 1, Abschnitt 10.1.5) ist ein Verfahren zur Program to Program Kommunikation. Es kann verwendet werden, um einen asynchronen RPC implementieren. MQSeries ist ein Beispiel. Programme werden in die Lage versetzt, ohne eine direkte Verbindung Informationen zu senden oder zu empfangen. Programme kommunizieren durch das Hinterlegen von Nachrichten in Message-Queues, und Abrufen von Nachrichten von Message-Queues.

Der JEE Java Message Service (JMS) ist eine Java spezifische Implementierung von MBQ. Es ermöglicht eine Nachrichtenübermittlung durch asynchrone Methodenaufrufe

Der Service, der einen MBQ Nachricht empfängt kann als reguläre Java-Klasse implementiert werden. Alternativ kann der Service als EJB, einer Message Drive Bean (MDB), implementiert werden.

Eine Message Driven Bean (MDB) ist eine spezielle Art einer Enterprise Java Bean (EJB). Eine Message Driven Bean ist ein potentieller Empfänger eines asynchronen Methodenaufrufes. Dieser Methodenaufruf kann von einer beliebigen anderen Java Klasse ausgehen. Die sendende Klasse wartet nicht auf eine Antwort.

Im Gegensatz zu Session Beans und Entitiy Beans erfolgt der Zugriff auf eine MDB nicht über eine Schnittstelle (Interface).

Alle Instanzen eines MDB – Typs sind gleich, da sie nicht direkt für den Client sichtbar sind und keine Zustände annehmen. Daraus resultiert die Möglichkeit für den EJB Container, Pools aus MDB – Instanzen zu bilden, um Skalierbarkeit zu erzeugen.

Der Websphere Web Application Server (WAS) benutzt das MQSeries Produkt, um die MDB Unterstützung zu implementieren. MQSeries existiert in 2 Versionen:

- **Die reguläre MQSeries Version, die alle Sprachen unterstützt. Diese Installation erfordert keinen Websphere Application Server, wird aber dennoch heute als WebSphere MQ bezeichnet.**
- **Eine spezielle MQSeries Version, die MDBs unterstützt und die in WebSphere integriert ist Diese unterhält Queues für die MDBs, sowie eine Listener Komponente, welche eine spezifische MDB benachrichtigt, dass in ihrer Queue eine Nachricht eingetroffen ist, die von der onMessage() Methode der MDB empfangen werden kann.**

Beide Versionen verwenden den gleichen Code. IBM hat daher beschlossen, beide Versionen in WebSphereMQ umzubenennen. Die reguläre MQSeries Version kann (und häufig wird) ohne WebSphere verwendet werden.

Message Driven Beans implementieren eine Enterprise Java Bean Variante einer Message Oriented Middleware wie z.B. MQSeries. Der WebSphere Application Server verwendet MQSeries Software Komponenten für die Implementierung der Message Driven Beans Infrastruktur.

Ein wesentlicher Unterschied zwischen den beiden MQ Varianten besteht darin, dass MDBs an Stelle des MQ Channels und der MQI API Enterprise Java Bean spezifische Konstrukte benutzen.

15.4.6 Message Listener Service

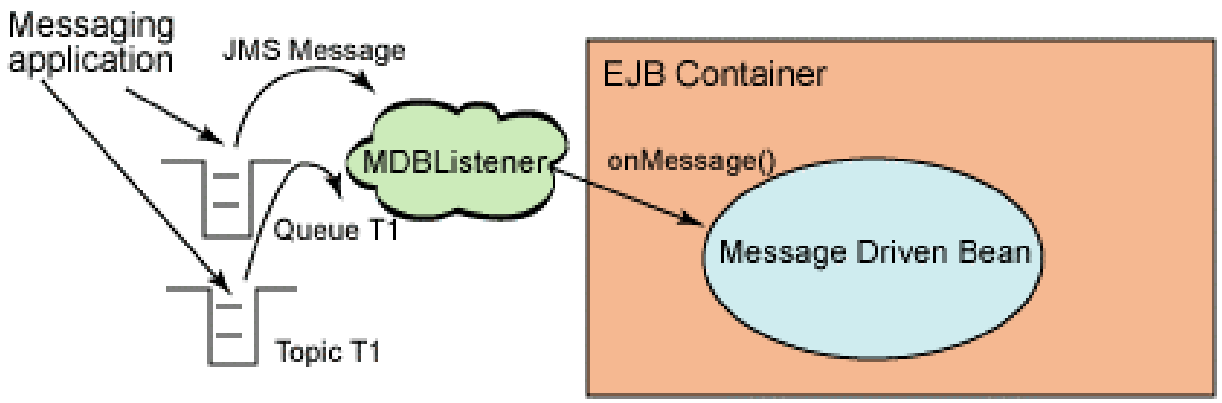


Abb. 5.4.9
Der Message Listener Service ist eine Funktion des Web Application Servers

MDBs hören (listen) ein bestimmtes JMS-Ziel ab. Dies kann zum Beispiel eine Queue sein. Die Message-Client sendet Nachrichten an die jeweilige Queue, auf die die Message Driven Bean zuhört.

Ein MDB-Listener ist ein Service eines WebSphere Application Servers. Der Message Listener Service verwaltet für jede MDB einen Listener. Ein MDB-Listener bewirkt, dass eine MDB (Message-Driven Bean) informiert wird, wenn eine Nachricht an einem bestimmten Ziel z.B. eine Queue angekommen ist . Dem Listener ist ein spezifischer Port zugeordnet. Wenn die Queue eine Nachricht empfängt, ruft der EJB-Container die `onMessage()`-Methode der Message-Driven Bean auf. Die `onMessage()` Methode kann auch als einzige Schnittstelle zu einem MDB betrachtet werden.

Ein Listener-Port ist eine Komponente des Applikationsservers, der ein JMS-Ziel und das Eintreffen von Nachrichten überwacht. MDBs sind an einen bestimmten Listener-Port gebunden. Wenn eine Nachricht an dem Listener-Port ankommt, wird sie an die entsprechende MDB (Message-Driven Bean) weitergeleitet..

In EJB 3.0 wird der MDB-Listener durch die „Aktivierungsspezifikationen“ ersetzt.

15.4.7 Web Application Server Implementierungen

WebSphere (IBM), Web Logic (Oracle) und Netweaver(SAP) sind die mit Abstand führenden Web Application Server Produkte. Nur WebSphere ist verfügbar unter z/OS.

Für einen Vergleich, siehe

http://www.cedix.de/VorlesMirror/Band2/WasCompet01.pdf

und

http://www.cedix.de/VorlesMirror/Band2/WasCompet02.pdf

Jboss, Geronimo und GlassFish (Oracle) sind Opensource-Implementierungen eines Web Application Servers.

Die eigenständige Servlet-Laufzeitumgebung Tomcat (ebenfalls Open Source) ist als Webcontainer (Servlet Container) standardmäßig in JBoss und Geronimo integriert.

JBoss und Geronimo sprechen ihren Transaktionsmanager (bis auf den Import und Export des Transaktionskontextes) allein über die JTS-Schnittstellen an, sodass sich verhältnismäßig einfach beliebige JTS-konforme Transaktionsmanager integrieren lassen.

Jboss, Geronimo und GlassFish haben nicht den Reifezustand und Funktionsumfang von kommerziellen Web Application Servern wie Oracle WebLogic oder IBM WebSphere. JBoss besteht aus knapp 4.300 Klassen und belegt installiert 50 Mbyte auf dem Festplattenspeicher, Websphere besteht aus über 20.000 Klassen und belegt installiert 460 MByte.

15.5 Weiterführende Information

Das Java Cult Buch „Java ist auch eine Insel“ ist kostenlos zum Download verfügbar unter http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/JavaInsel.pdf.

Kurzbeschreibung von JEE 7
http://www.heise.de/newsticker/meldung/Oracle-gibt-Java-Enterprise-Edition-7-offiziell-frei-1887250.html

Zum Thema Java Stand-alone Anwendungen existieren zwei Redbooks:

http://www.redbooks.ibm.com/abstracts/sg247177.html
http://www.redbooks.ibm.com/abstracts/sg247291.html

Ein Java Video Tutorial unter Benutzung von Eclipse ist zu finden unter

http://www.youtube.com/watch?v=le_-tsHyLXU

Es existieren mehrere Video Tutorials zum Thema JEE, z.B.

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

http://www.youtube.com/watch?v=hD2L9AE9sDc

http://www.youtube.com/watch?v=0q-HC3PDX5s

Nicht alles ist positiv mit Java. See:
http://www.inventus.org/posterous/file/2011/12/8168678-083-085_COM.pdf ,
oder Mirror

http://www.cedix.de/VorlesMirror/Band2/JavaWithers.pdf

Eine Dalvik Slide Presentation ist verfügbar unter

https://14b1424d-a-62cb3a1a-s-sites.googlegroups.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attachauth=ANoY7coirObuO3E_oo_ONpPrvNv0FtwVAZbiaqy0_83GabP2Eg40TuIG09FZWN5zabZY3OtMV3w1yjrzKYAp_SYhUDoAevRE7Lbd9meZMb2cBKc-QZH4XzPC8g9OUkMZAiDp908K7X_QnZno21DuDYphAibQ5AGuNBAZs0wCQK7TnjBfPvVM0n4CUOCeosFrXS4S--dUyrIRE6LB3q7VK21QpN0mRfVChLZXN50PNlM--Z6PQEfCWjhYI72IOS8LxMLWK3B9WrSzyANfKOXuJCgQE62HuiXxdw%3D%3D&attredirects=0

oder als Mirror

http://www.cedix.de/VorlesMirror/Band2/DalvikVM.pdf

Ein Video, welches die Android Dalvik virtuelle Maschine beschreibt, ist verfügbar unter

https://sites.google.com/site/io/dalvik-vm-internals/

16. Remote Method Invocation

16.1 Object Request Broker

16.1.1 Remote Procedure Call

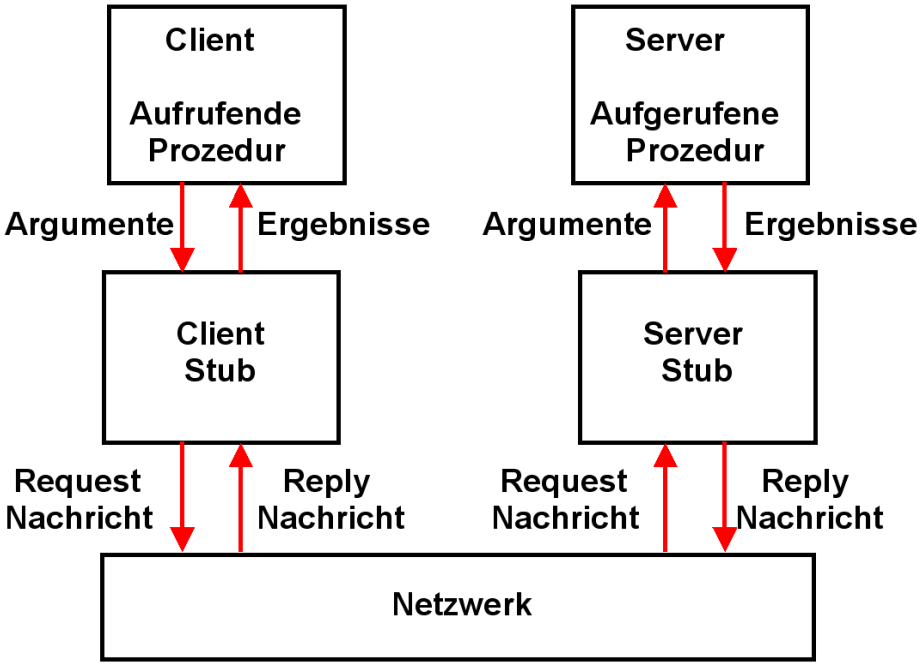


Abb. 16.1.1
Server und Client Stubs

Siehe hierzu auch Band 1, Abschnitt 10.1.1.

Client und Server laufen als zwei getrennte Prozesse.

Die beiden Prozesse kommunizieren über Stubs. Stubs sind Routinen, welche Prozedur-Aufrufe auf Netzwerk RPC Funktionsaufrufe abbilden.

Ein Server-seitiges RPC Programm stellt den Klienten seine Dienste über eine Interface (Schnittstelle) zur Verfügung. Es definiert seine Interface unter Benutzung einer Interface Definition Language (IDL).

Ein Stub Compiler liest die IDL Schnittstellenbeschreibung und produziert zwei Stub Proceduren für jede Server Procedure (Client Side Stub und Server Side Stub).

Der klassische RPC benutzt die Bezeichnung Server Stub. Modernere RPCs verwenden statt dessen die Bezeichnung Skeleton. Die Bezeichnungen Server Stub und Skeleton sind austauschbar.

Stubs bzw. Skeletons implementieren eine RPC Runtime. Sie übersetzen Aufrufe der Client API in Sockets und dann in Nachrichten, die über das Netz übertragen werden.

Beim CICS Distributed Program Link (DPL) benötigt der Aufruf eines entfernten Systems lediglich dessen Name (TRID) und die Bezeichnung der COMMAREA. Eine IDL ist nicht erforderlich.

16.1.2 Sockets und RPC

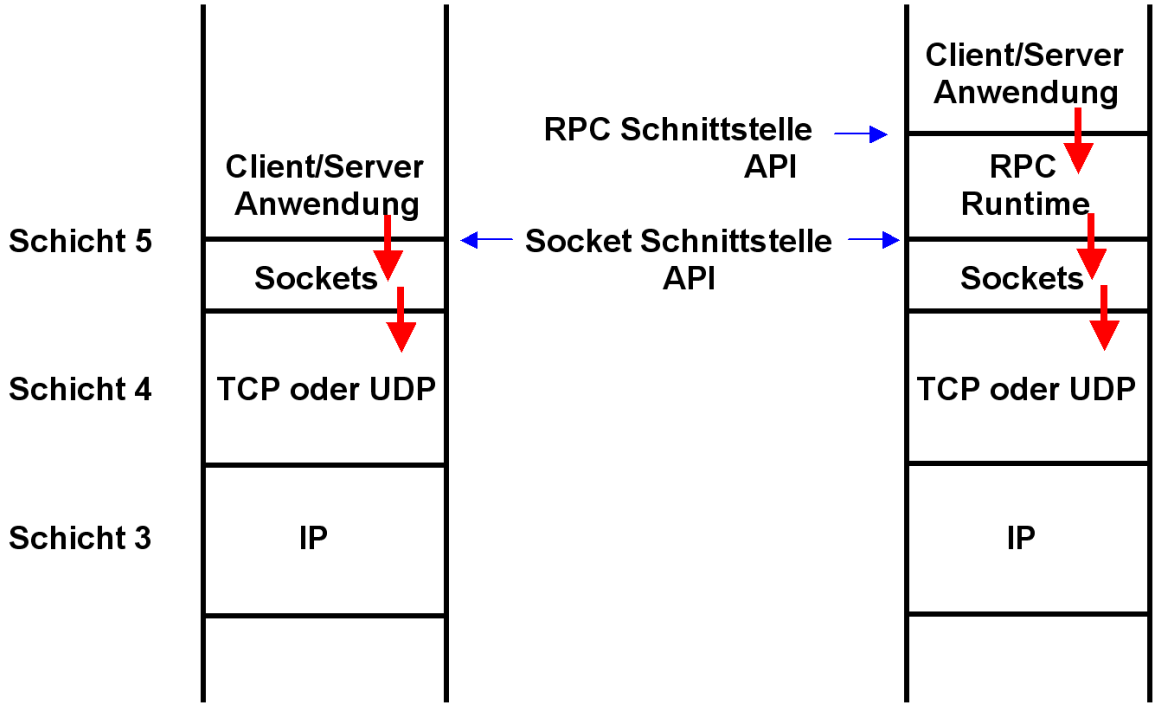


Abb. 16.1.2
Der RPC ist oberhalb der Socket Schicht einzuordnen

De Facto alle Client/Server Verbindungen benutzen heute TCP oder UDP in der Schicht 4 des OSI Modells. In der darüberliegenden Schicht 5 wird meistens das Socket Protokoll eingesetzt. Sockets sind leistungsfähig und vielseitig einsetzbar. Das Anwendungsprogramm des Klienten benutzt die Socket Schnittstelle, um mit dem Anwendungsprogramm (Service) des Servers zu kommunizieren. Sockets sind jedoch schwierig zu programmieren.

Deswegen wird universell der Remote Procedure Call (RPC) eingesetzt. Der RPC enthält bereits viele Funktionen, die bei der Nutzung von Sockets von Hand programmiert werden müssen. Das Anwendungsprogramm des Klienten benutzt die RPC Schnittstelle, um mit dem Anwendungsprogramm (Service, hier auch als Procedure bezeichnet) des Servers zu kommunizieren. Die RPC Runtime übersetzt die RPC Aufrufe in Socket Aufrufe.

Die RPC Runtime ist eine Sammlung von Routinen, welche die RPC Schnittstelle (API) implementieren.

16.1.3 IDL (Interface Definition Language)

Ein Server-seitiges RPC Programm stellt den Klienten seine Dienste über eine Interface (Schnittstelle) zur Verfügung. es definiert seine Interface unter Benutzung einer Interface Definition Language (IDL).

Die Prozedur eines Servers kann vom Klienten über eine Schnittstelle (Interface) in Anspruch genommen werden. Wenn man eine Server-seitige Anwendung entwickelt, erstellt man zwei Komponenten:

- **Die Anwendung selbst, oft als „Implementation“ bezeichnet,**
- **Die Schnittstelle (Interface), welche beschreibt, wie die Anwendung von einem Klienten aufgerufen werden kann, einschließlich der Beschreibung der übergebenen Parameter.**

Die Anwendung kann in einer beliebigen Programmiersprache geschrieben werden. Die Schnittstelle wird in einer spezifischen Sprache, der IDL (Interface Definition Language) beschrieben.

Mit Hilfe eines „IDL Compilers“ wird die in der IDL beschriebene Schnittstelle übersetzt. Das Ergebnis ist ausführbarer binary Code für Skeleton und Stub. Skeleton und Stub werden somit mit Hilfe des IDL Compilers automatisch generiert. Sie müssen die Eigentümlichkeiten der Schnittstelle der Server Prozedur kennen, nicht aber die Art, in der die Server Prozedur implementiert wurde. Spezifisch brauchen sie nicht zu wissen, in welcher Sprache (Cobol, C++, Java) die Server Prozedur implementiert wurde.

Die unterschiedlichen RPC Implementierungen verwenden unterschiedliche IDL Sprachen und damit unterschiedliche IDL Compiler.

Java nimmt eine Sonderstellung ein. Die Schnittstelle einer Java Klasse, z. B. die Remote Interface einer EJB, wird mit Hilfe eines Java Sprachelements, der Java Interface, beschrieben. Eine zusätzliche Java IDL ist in den meisten fällen nicht erforderlich.

16.1.4 Business Objekt

Business Objekte sind Repräsentationen von Dingen des täglichen Lebens.

Beispiel: Das Business Objekt „Fritz Meier“ ist eine Instanz (Ausprägung) der Objektklasse „Bankkonto“. Franz Müller, Barbara Schneider und Else Schmidt sind weitere Instanzen der Objektklasse Bankkonto.

Services der Objektklasse Bankkonto können in Anspruch genommen werden, indem man ihre Methoden aufruft. Derartige Methoden der Objektklasse Bankkonto können z.B. sein:

- **Kontostand abfragen**
- **Geld einzahlen**
- **Geld abheben**
- **Postanschrift ändern**
- **Vollmacht für Ehepartner erteilen**

usw.

Eine Methode wird aufgerufen, indem man an ihre Interface eine Nachricht schickt. Die Nachricht enthält einzelne Parameter, z.B.

- **Name/Adresse der Objektklasse, z.B. Bankkonto Fritz Meier**
- **Name der Methode, z.B. Kontostand abfragen**
- **übergebene Parameter, z.B. stand vom 1.Januar letzten Jahres**
- **Parameter der Antwort, z.B. € 638,15 .**

Komponente		
1	Name, Vorname	
2	Titel, Anrede	
3	Geburtsdatum	
4	PLZ	Firmenanschrift 1
5	Ort	
6	Straße	
7	PLZ	Firmenanschrift 2
8	Ort	
9	Straße	
10	PLZ	Privatanschrift
11	Ort	
12	Straße	
13	Kunden-Nr. = f(Objekt Nr., Objekt Identifikation, ...	
14		

ca. 500 weitere Attribute, z.B.:

- Fax Nr.
- Tel. Nr.
- e-mail Adresse
- Info über Partner
- Info über Kinder
- Zeiger auf Kinder
- Arbeitgeber
- Stellung im Betrieb
- Mitglied im Tennisclub xyz
- Rentendaten

•
•
•
•

Abb. 16.1.3
Beispiel: Business Objekt “Kunde”

Abb. 8.2.3 stellt ebenfalls ein Business Objekt dar.

16.1.5 Wiederverwendbarkeit von Code

Vorbild: Entwurf / Bau einer Brücke im Bauingenieurwesen

Objekttechnologie ermöglicht Code Blöcke mit fest definierter Funktionalität, die in Klassen gekapselt werden. Ein Objekt als Instanz einer Klasse stellt sich dem Entwickler als Black Box mit einer öffentlichen Schnittstelle dar.

Wird nur in einer einzigen Sprache mit identischem Compiler entwickelt ist die Wiederverwendbarkeit von Klassen am leichtesten erreichbar. Beim Einsatz mehrerer Programmiersprachen muss die Objektphilosophie der Programmiersprache beachtet werden. Z.B. unterstützt C++ die Mehrfachvererbung, Java aber nur die Einfachvererbung.

Um objektorientierte Bibliotheken mit unterschiedlichen Sprachen und Compilern zu realisieren, ergeben sich eine Reihe von Problembereichen:

- **Sprachenabhängigkeit: Smalltalk- und C++-Objekte verstehen einander nicht.**
- **Compilerunabhängigkeit: Objekte zweier Compiler (z.B. Watcom C++ und *GNU* C++) können nicht miteinander kommunizieren, weil die Verwaltung interner Informationen nicht standardisiert ist.**
- **starre Kopplung zwischen Objekten: Wenn sich die Implementierung einer Klasse ändert, müssen alle Teile, die diese Klasse in irgendeiner Form nutzen, neu kompiliert werden. Beispiel: C++ Compiler benutzen Konstanten beim Zugriff auf Daten und Methoden.**
- **Beschränkung auf einen Prozessraum (Objekte können nicht über Prozessgrenzen hinweg kommunizieren).**

Zielsetzung: Unterschiedliche objektorientierte Klienten-Implementierungen, die auf unterschiedlichen Plattformen (z.B. HP-UX, AIX, Solaris, Linux, XP, z/OS) laufen, sollen mit unterschiedlichen objektorientierten Server-Implementierungen (ebenfalls unterschiedliche Plattformen) in Binärform kompatibel zusammen arbeiten.

Klienten sollen maximal portierbar sein, und ohne Änderungen auf Rechnern/Betriebssystemen unterschiedlicher Hersteller lauffähig sein.

Klienten sollen keine Kenntnis benötigen wie ein Objekt auf einem Server implementiert ist.

16.1.6 Objekt orientierter RPC

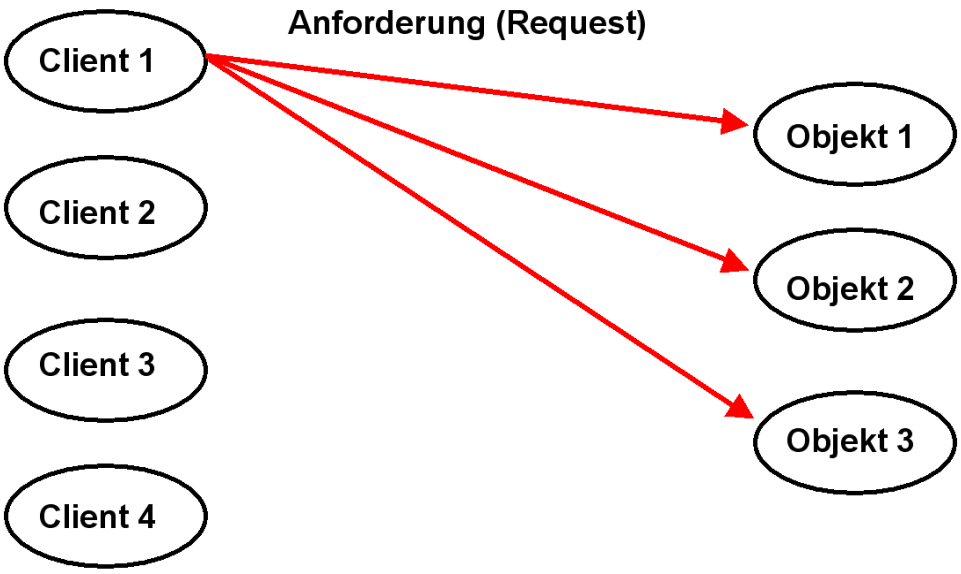


Abb. 16.1.4
In einem Objekt-orientierten RPC ersetzten Server Objekte die Services des Servers

In einem Objekt-orientierten RPC ist der Service (Dienstleistung eines Servers) als Objekt implementiert. Der Service wird in Anspruch genommen, indem die Methoden des Objekts aufgerufen werden.

Ein Klient ist eine Software-Einheit, die eine Operation (eine Methode) eines Objektes aufruft

Problematisch ist, dass Objekte in unterschiedlichen Sprachen implementiert werden, aber in binärer Form auf dem Server installiert sind. Die binäre Implementierung eines Client Objects ist ohne Anpassung nicht in der Lage mit der binären Implementierung eines Server Objects zu kommunizieren.

16.1.7 Kommunikations-Alternativen für den Objekt-orientierten RPC

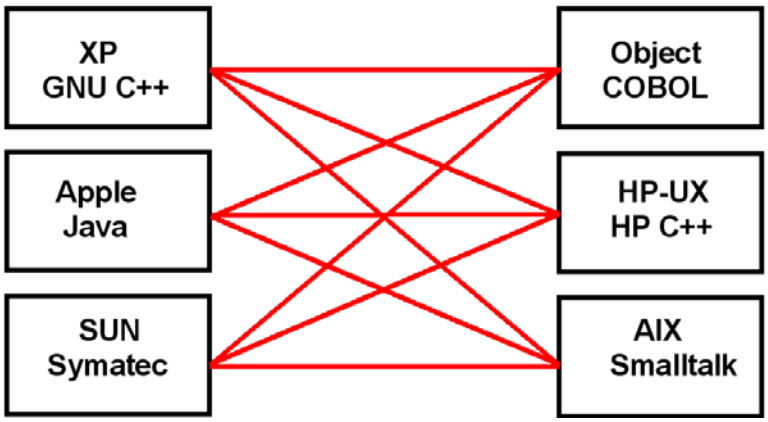


Abb. 16.1.5
Any to any Kommunikation

Ohne zusätzliche Vorrichtungen muss die Kommunikation jedes Klienten an jedes Server Objekt je nach verwendeter Sprache und darunterliegender Platform getrennt angepasst werden.

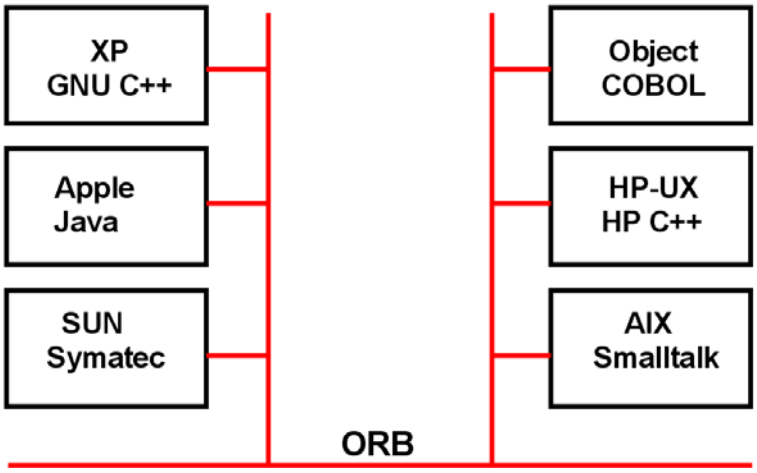


Abb. 16.1.6
Der ORB vereinheitlicht die Kommunikation

Unter Benutzung eines Object Request Brokers (ORB) können in unterschiedlichen Programmiersprachen entwickelte binäre Objekte plattform-unabhängig miteinander kommunizieren.

16.1.8 Object Request Broker

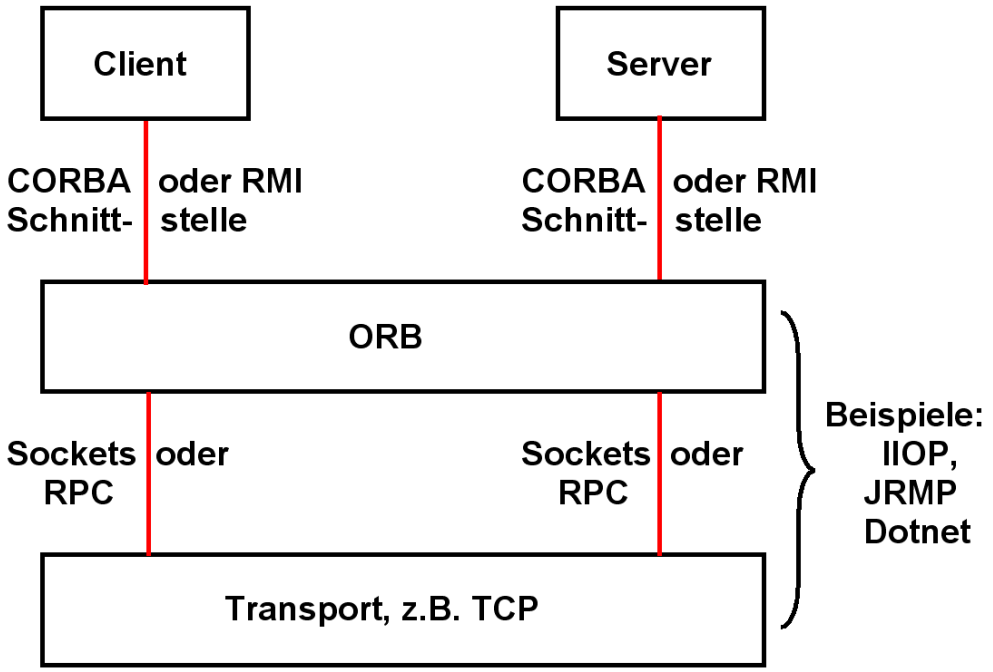


Abb. 16.1.7
Objektorientiertes Client/Server-System

In einem Objekt orientierten Client/Server System wird eine „Object Request Broker“ (ORB) Schicht zwischen dem Client und Server auf der einen Seite, und der Transport Schicht auf der anderen Seite zwischen geschaltet.

Ein ORB (Object Request Broker) ist eine Komponente eines Betriebssystems. Er ermöglicht es, dass Objekte, die in unterschiedlichen Programmiersprachen entwickelt wurden, in binärer Form miteinander zusammenarbeiten. Dies kann über Prozess- und physische Rechnergrenzen hinweg geschehen.

Hierfür stellt der ORB ein einheitliches Klassenmodell sowie Standards für die Organisation von Klassen-Bibliotheken und die Kommunikation zwischen Objekten zur Verfügung.

Es existieren mehrere Alternativen einen ORB zu implementieren:

- **CORBA Standard der OMG (Open Management Group),**
- **Remote Method Invocation (RMI), Teil des Java JDK der Fa. SUN**
- **DotNet der Fa. Microsoft.**

In eine ähnliche Richtung geht die Internet orientierte SOAP / UDDI / WSDL Initiative.

16.1.9 Interoperable Object Reference, IOR

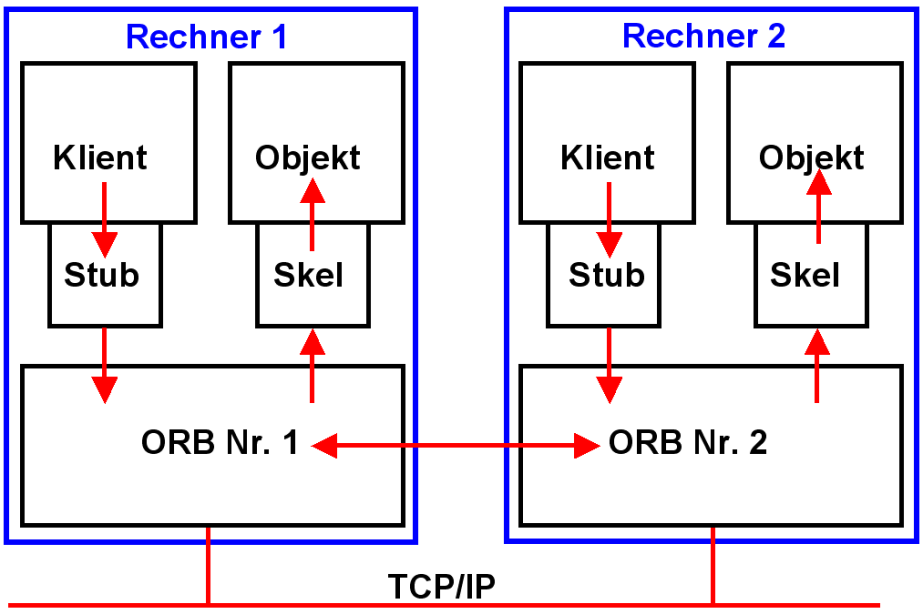


Abb. 16.1.8
Stubs und Skeleton Schnittstellen mit der lokalen ORB Komponente

Der ORB stellt eine Schnittstelle zwischen Client und Server dar, die ähnlich der Socket- oder RPC-Schnittstelle arbeitet, aber

- **auf einer höheren Ebene arbeitet (und deshalb evtl. Sockets oder RPC für die eigentliche Kommunikation verwendet,**
- **objektorientiert arbeitet,**
- **für die Kommunikation das IIOP oder das JRMP Protokoll verwendet (oder andere).**

Klienten und Server Objekte kommunizieren mit ihrem ORB über Stubs und Skeletons. Sie können transparent auf dem gleichen oder auf entfernten Rechnern arbeiten.

Eine eindeutige Objekt Bezeichnung (16-8, IOR) wird dem Objekt bei der Entstehung zugeordnet. Der Client nimmt die Dienste eines Objektes in Anspruch, indem er es zunächst mit Hilfe seiner IOR lokalisiert, und dann einen Methodenaufruf ausführt. Der Methodenaufruf enthält:

- **IOR**
- **Methodennamen**
- **Parameter**
- **Kontext (enthält Information über die Position des Aufrufers und nimmt Fehlermeldungen entgegen)**

16.1.10 Inter ORB Protokolle

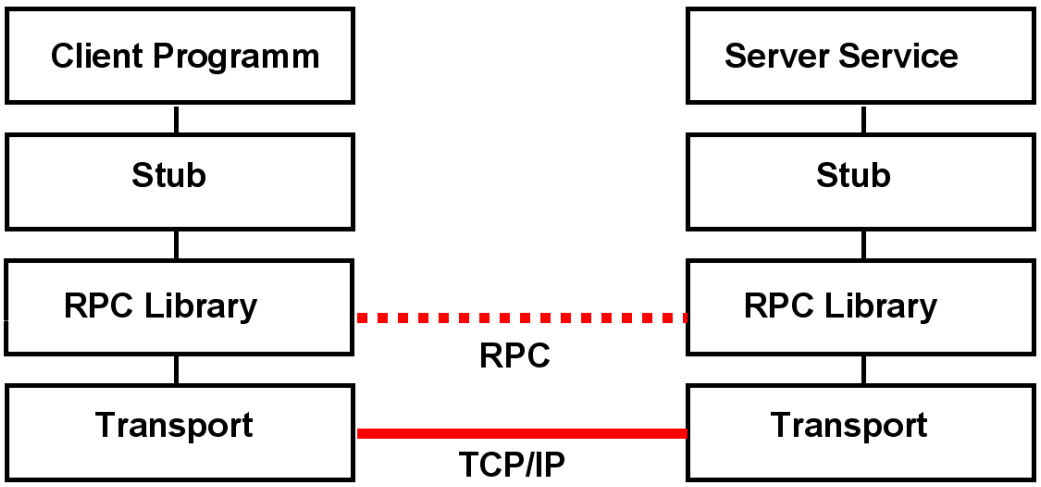


Abb. 16.1.9
RPC Library Kommunikation

Beim klassischen RPC kommuniziert die RPC Library des Klienten mit der RPC Library des Servers mittels eines RPC spezifischen Protokolls

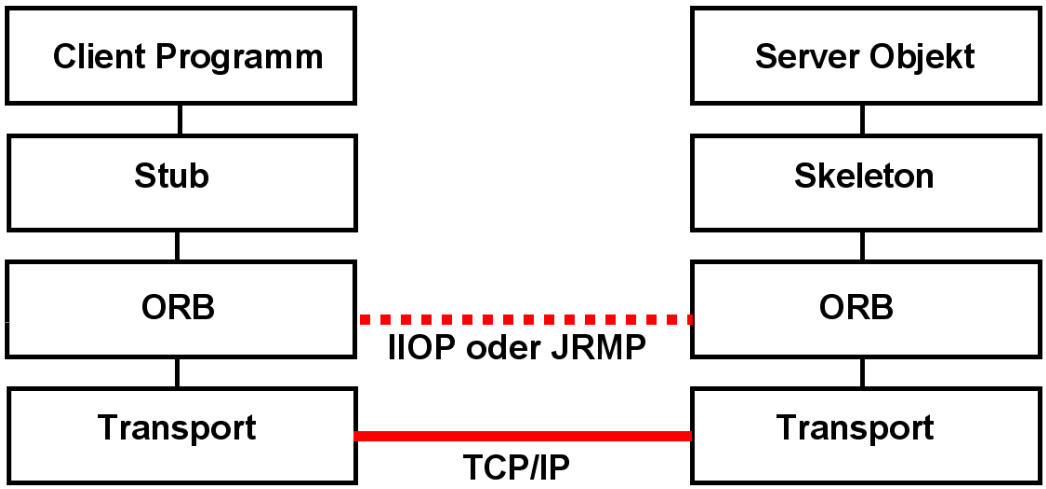


Abb. 16.1.10
ORB Kommunikation

Beim objektorientierten RPC kommuniziert die ORB Komponente des Klienten mit der ORB Komponente des Servers mittels der IIOP oder JRMP Protokolle.

IIOP (Internet Inter-ORB Protokoll) und JRMP (Java Remote Method Protokoll) sind auf der gleichen Ebene einzuordnen wie die http, SOAP, 3270 oder FTP Protokolle.

16.1.11 Integration von unterschiedlichen Object orientierten Client/Server Protokollen

Es existieren drei bedeutende objektorientierte Client/Server Standards:

- **CORBA**
- **JEE Remote Method Invocation (RMI)**
- **DotNet von Microsoft**

CORBA (Common Object Broker Architecture) ist ein Standard der OMG (Open Management Group). Die OMG ist ein 1989 gegründeter, internationaler, nicht profit-orientierter Zusammenschluss von zunächst 8 (Anfang 1996: Ca. 600) Softwareentwicklern, Netzbetreibern, Hardwareproduzenten und kommerziellen Anwendern von Computersystemen (ohne Microsoft).

CORBA unterstützt viele Sprachen, darunter C/C++, COBOL, PLI, ADA und Java. CORBA Produkte werden von zahlreichen Herstellern implementiert. Für alle von CORBA unterstützten Sprachen existiert eine einheitliche Interface Definition Language (IDL). IDL ist der OMG Standard um Sprach-neutrale APIs zu definieren. IDL ermöglicht eine Platform-unabhängige Beschreibung der Interfaces von verteilten (distributed) Objekten.

Der RMI-Standard bietet zwei alternative Protokolle: JRMP und RMI/IIOP. RMI/IIOP wurde entwickelt, um CORBA und RMI miteinander zu verbinden. Einige Hersteller von JEE Software bieten beide Alternativen in ihren Produkten an. IBM hat sich entschieden, ausschließlich RMI/IIOP einzusetzen, was konform mit dem RMI Standard ist.

Die Koexistenz Charakteristika von Corba und RMI sind sehr gut.

Auf der anderen Seite sind die Microsoft DotNet Produkte sehr unvereinbar mit der Corba/RMI Welt. Es ist möglich, eine Kommunikation zwischen beiden Alternativen mittels ORB Brücken zu erreichen. Allerdings ist die Leistung nicht optimal und die Integration erfordert Expertenwissen. Aus diesem Grund ist DotNet bei größeren Unternehmen, die Java häufig verwenden, nicht sehr beliebt. Es wird vor allem in kleineren Unternehmen eingesetzt, die hauptsächlich Microsoft Server Produkte einsetzen.

Web Services und ihr SOAP RPC sind eine moderne Alternative zu RMI und Corba.

16.1.12 RMI

Remote Method Invocation (RMI, deutsch etwa „Aufruf entfernter Methoden“), ist der Aufruf einer Methode eines entfernten Java Objekts und realisiert die Java-eigene Art des Remote Procedure Calls. „Remote“ bedeutet dabei, dass sich das Objekt in einer anderen Java Virtual Machine befinden kann, die ihrerseits auf einem entfernten Rechner oder auf dem gleichen lokalen Rechner laufen kann. Dabei sieht der Aufruf für das aufrufende Objekt (bzw. dessen Programmierer) genauso wie ein lokaler Methoden Aufruf aus; es müssen jedoch besondere Ausnahmen abgefangen werden, die zum Beispiel einen Verbindungsabbruch signalisieren können.

Auf der Client-Seite kümmert sich der Stub um den Netzwerktransport. Vor dem Erscheinen der Java Standard Edition (JSE) in Version 1.5.0 war der Stub eine mit dem RMI-Compiler rmic erzeugte Klasse. Seit der Version 1.5 ist es nicht mehr notwendig, den RMI-Compiler aufzurufen. Das Erstellen des Stubs wird von der Java Virtual Machine übernommen.

Für die erste Verbindungsaufnahme werden aber die Adresse des Servers und ein Bezeichner (z.B. eine RMI-URL) benötigt. Für den Bezeichner liefert ein Namensdienst auf einem Server eine Referenz auf das entfernte Objekt zurück. Damit dies funktioniert, muss sich das entfernte Objekt im Server zuvor unter diesem Namen (IOR oder String Name) beim Namensdienst registriert haben. Der RMI Namensdienst wird über statische Methoden der Klasse java.rmi.Naming angesprochen. Der Namensdienst ist als eigenständiges Programm implementiert. Es existieren zwei Arten des Namensdienstes, der RMI Registry für lokale Netze und der „Java Naming and Directory Service“ (JNDI) für weltweite Netze.

Eine Client/Server Anwendung, bei der sowohl der Client als auch der Server in Java programmiert sind, kann sowohl RMI als auch eine andere RPC Art verwenden (z.B. CORBA oder DCE). RMI ist aber ein besonders einfaches Verfahren, Client/Server Anwendungen zu erstellen.

16.2 RMI

16.2.1 Java Method Invocation

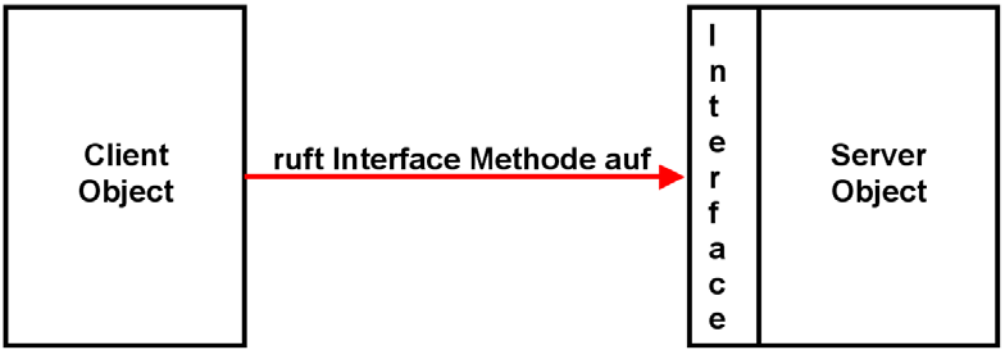


Abb. 16.2.1
Ein Java Client Objekt ruft eine Methode eines Java Server Objektes auf

Eine JVM kann viele Klassen (Objekte) speichern.

Eine Klasse kann mittels eines Methodenaufrufs eine Methode einer anderen fremden Klasse aufrufen. Der Methodenaufruf überträgt die Aktivität auf das durch eine Referenz angegebene (fremde) Objekt (Klasse).

Bei einem lokalen Aufruf ruft der Klient eine Methode (von potentiell mehreren) auf, die in der Interface des aufgerufenen Objektes beschrieben ist.

Was passiert beim Methodenaufruf in einer Zeile wie

```
String datei=gibtWertfuerParameter("-datei",args);
```

Auf der rechten Seite steht ein Funktions- oder Methodenaufruf. Ein Methodenaufruf ähnelt in seiner Form der ersten Zeile einer Methodendefinition: Auf einen Namen folgt ein Paar runder Klammern. Zwischen den Klammern können Parameter stehen (es gibt auch Methoden ohne Parameter).

In der großen Mehrzahl der Fälle erfolgen Java Methodenaufrufe nur innerhalb einer einzelnen JVM (lokaler Methodenaufruf). Beim lokalen Methodenaufruf rufen Sie eine Methode einer Klasse auf, die sich in der gleichen JVM befindet. Mit Hilfe der Remote Method Invocation (RMI) sind Objekte in einer bestimmten (lokalen) JVM in der Lage, Methoden von Objekten in einer entfernten JVM aufzurufen. Beim entfernten (remote) Methodenaufruf kann sich die angesprochene JVM entweder auf dem gleichen Rechner wie die aufrufende JVM befinden (meistens in einem anderen Address Space), oder sie kann sich auf einem beliebigen anderen Rechner im Internet befinden. Syntax und Semantik des Methodenaufrufs sind in beiden Fällen identisch; die JVMs managen die Unterschiede.

RMI ist ein Java spezifischer entfernter Methodenaufruf, ähnlich zum klassischen RPC (Remote Procedure Call), CORBA (Common Object Request Broker Architecture) RPC und dem Web Services (SOAP) RPC.

16.2.2 Arten des Methoden Aufrufs

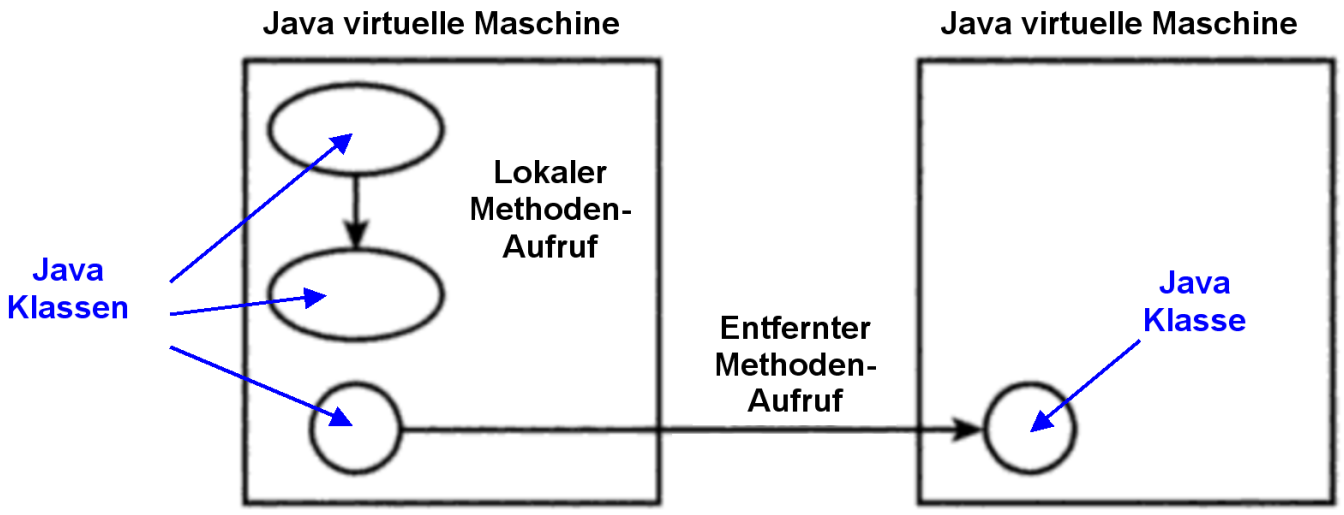


Abb. 16.2.2
Der entfernte Methodenaufruf erfolgt mittels des RMI Protokolls

Der entfernte Methodenaufruf, Remote Method Invocation (RMI) ist einer der Eckpfeiler von Enterprise JavaBeans und eine ausgesprochen praktische Möglichkeit, verteilte Java-Anwendungen zu erstellen.

Lokale und entfernte Methodenaufrufe haben unterschiedliche Performance-Eigenschaften

- **Der Zugriff auf ein Remote Object erfordert die Erstellung von Stubs und Skeletons. Dies braucht CPU Zyklen.**
- **Bei der Übertragung von Variablen bestehen Performance Unterschiede. Unterschiedliche Arten von Daten haben einen unterschiedlichen Marshalling Overhead. Marshalling (von engl. to marshal, aufstellen, anordnen) ist das Umwandeln von strukturierten oder elementaren Daten in ein Format, das die Übertragung über das Netz und die Übermittlung an andere Prozesse ermöglicht.**

16.2.3 Remote Method Invocation

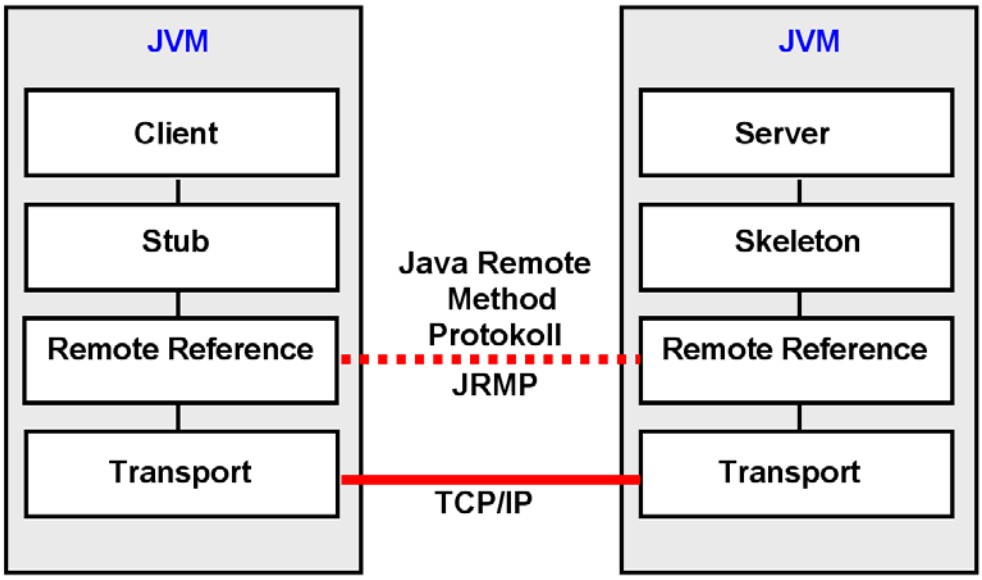


Abb. 16.2.3
JRMP ist das ursprüngliche RMI Protokoll

Remote Method Invocation (RMI) ist eine Client-Server-API für den Aufruf von Java Programmen auf geographisch entfernten Rechnern

Unter Benutzung von RMI können Objekte einer bestimmten JVM die Methoden von Objekten in einer entfernten JVM aufrufen.

Zur Realisierung wird ein Stellvertreter (Client-Stub) des entfernten Objekts in der lokalen JVM erzeugt. Dieser kommuniziert mit einem Stellvertreter (Server-Skeleton) des entfernten Objektes in dessen JVM.

16.2.4 RPC Server und RPC Service

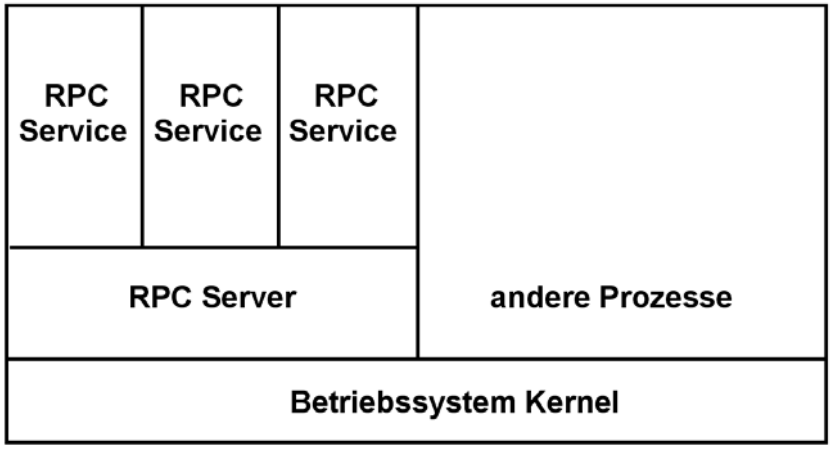


Abb. 16.2.4
RPC Server als Laufzeitumgebung für RPC Services

Eine aufgerufene Prozedur wird als RPC-Service (oder Implementation) bezeichnet. Ein RPC-Server bietet in der Regel viele unterschiedliche Arten von RPC Services an. Die RPC Services (RPC Dienstprogramme) können entweder in getrennten virtuellen Adressenräumen laufen, oder alternativ als Threads innerhalb eines einzigen virtuellen Adressenraums implementiert werden. In CICS ist der CICS Nucleus der RPC-Server; einzelne Transaktionen, die durch ihre TRID gekennzeichnet sind, sind die RPC Services.

Der RPC-Server liefert eine Laufzeitumgebung für die RPC-Services. Er stellt Verwaltungsdienste für seine RPC Services zur Verfügung, z.B.:

- **Registry Funktionen,**
- **Binding (Address Resolution),**
- **Sicherheits-Funktionen,**
- **Kommunikations-Funktionen, usw.**

Die RPC Services können in beliebigen Sprachen implementiert werden, auch als Java Objekte.

Ein physischer Rechner kann mehrere RPC-Server beherbergen, von denen jeder eine andere Laufzeitumgebung für seine RPC-Services zur Verfügung stellt. Ein Beispiel ist ein z/OS-System, welches einen CICS Server, einen DCE-Server mit relaxten Sicherheitsmerkmalen, und einen weiteren DCE-Server mit strengen Sicherheitsanforderungen enthält.

Häufig laufen Hunderte oder Tausende unterschiedlicher RPC Services auf dem gleichen Rechner.

Java RPC Services können als Threads innerhalb einer JVM implementiert werden. Alternativ kann ein Java RPC-Server auch mehrere JVMs, jeweils für Gruppen von Java Services, unterhalten.

16.2.5 Home Banking Beispiel

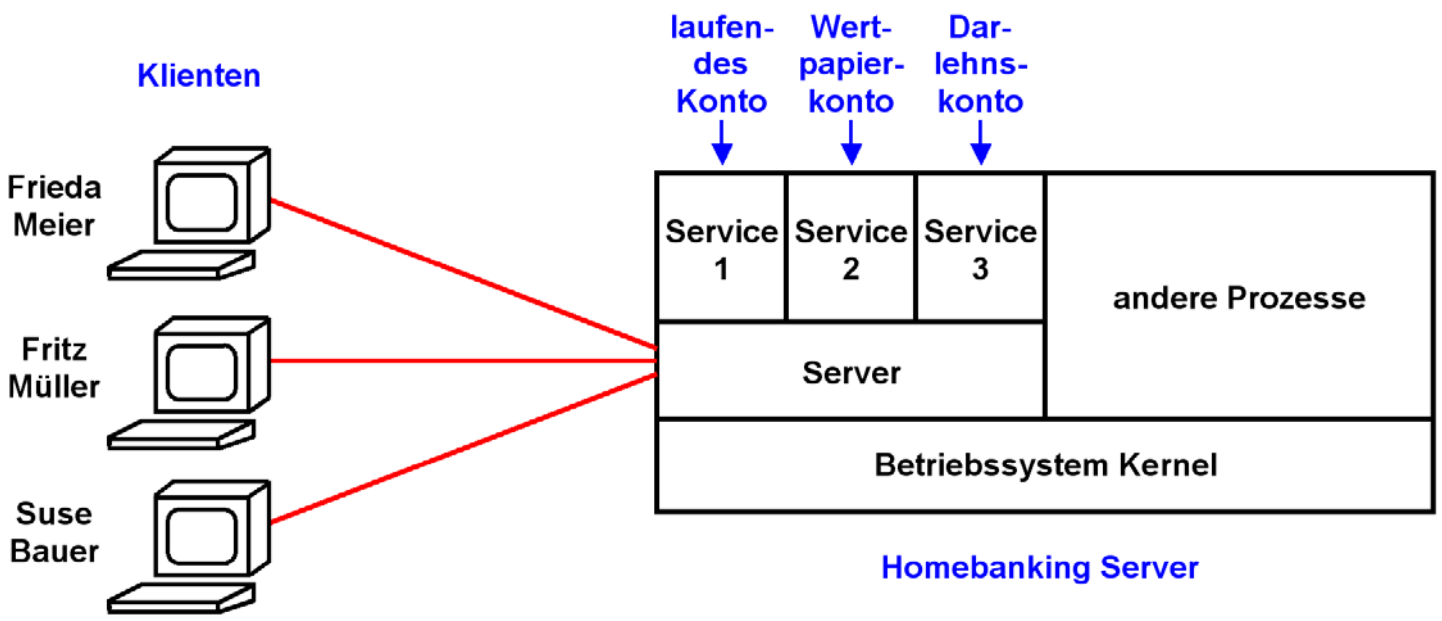


Abb. 16.2.5
Ein Service kann von mehreren Klienten gleichzeitig in Anspruch genommen werden

Dies sei an Hand eines Homebanking Beispiels erläutert. Angenommen, Klienten und Server sind in Java implementiert. Die drei Kunden Frieda Meier, Fritz Müller und Suse Bauer unterhalten bei einer Bank je ein laufendes Konto, ein Wertpapierkonto und ein Darlehnskonto. Auf ihren PCs ist ein Java Client in einer JVM installiert. Auf dem Server laufen drei Services als drei getrennte Prozesse, jeweils mit einer eigenen JVM. Services werden von den Klienten mittels RMI aufgerufen. Service 1 implementiert das laufende Konto mit Methoden wie kontostandabfragen, geldüberweisen, daueraufträgstornieren usw. Service 2 implementiert das Wertpapierkonto mit Methoden wie wertpapierkaufen, wertpapierverkaufen, aktienkursentwicklung usw. Service 3 implementiert das Darlehnskonto mit Methoden wie tilgungsstatus abfragen, sondertilgungvornehmen, usw.

Der Server implementiert für seine Services Dienstleistungen wie Authentification, Load Balancing, Errror Recovery, Namensdienste, Transaktionsdienste, Journalling usw.

16.2.6 Threads auf der Server-Seite

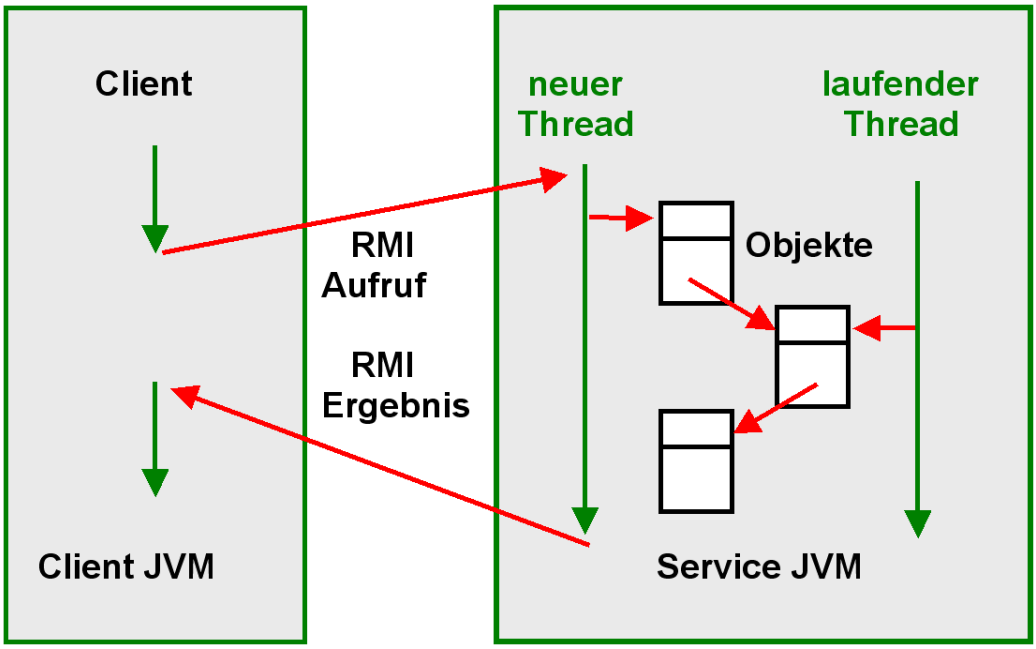


Abb. 16.2.6
Parallele Verarbeitung durch mehrere Java Threads

Ein Client ruft einen von mehreren Services mittels RMI auf. Die Services sind als Java Klassen (Java Objekte) implementiert. Jeder RMI-Aufruf erzeugt auf der Server-Seite einen neuen Thread. Jeder Thread ist eine Instanz der Service Klasse, ein Java Object. Das bedeutet, dass in der Server JVM mehrere solcher Threads gleichzeitig laufen können, zusammen evtl. mit weiteren dauerhaft laufenden Threads des Servers (z.B. Garbage Collector, Compute-Server,...). Wenn Frieda Meier und Fritz Müller gleichzeitig ihr laufendes Konto abfragen, wird also für beide je ein Java Thread mit den Objekt Variablen Frieda Meier und Fritz Müller angelegt.

Man muss sich also in jedem Fall Gedanken über eine notwendige Synchronisation (Concurrency) machen (mit synchronisierter Blockierung, ggf. auch mit wait() und notify()), um ein Zugriff mehrerer Threads auf gemeinsam genutzte Daten zu steuern. Auch wenn keine vollständigen ACID Eigenschaften vorhanden sind, dürfen die Threads sich nicht gegenseitig beeinflussen.

16.2.7 String Name und Objekt Referenz

In unserem Beispiel haben die Klienten die Option, auf einen von mehreren Services zuzugreifen: Laufendes Konto, Wertpapier Konto oder Darlehnskonto. Klienten selektieren den Service mit Hilfe des Namens des Services, z.B. laufendeskonto. Namen sind üblicherweise Zeichenketten (Strings), weshalb auch die Bezeichnung „String Name“ gebräuchlich ist.

Ein bestimmter Service ist gekennzeichnet durch zwei Bezeichner: Seinen Namen sowie die Adresse, über die er erreichbar ist. Wenn in unserem Beispiel jeder Prozess über eine eigene IP Adresse erreichbar ist, könnte dies z.B. die IP Adresse 134.2.205.55, Port Nr. 2012 sein. Diese Adressse wird als Objekt Referenz bezeichnet.

Wenn ein Klient auf einen Service zugreift, ist es daher erforderlich, den Namen in die Objekt Referenz zu übersetzen. Dies geschieht mit Hilfe eines getrennten Server Prozesses. Hier existieren zwei Alternativen:

- **Registry**
- **Namens- und Verzeichnis Dienstes**

Eine Registry ist ein einfacher Namensdienst. Er läuft als getrennter Prozess auf dem gleichen physischen Server, auf dem auch die Services laufen, und ist typischerweise auf eine LAN Umgebung begrenzt. Ein Namens- und Verzeichnisdienst (Naming and Directory Service) ermöglicht es, diesen (und auch die Services) auf physisch unterschiedlichen Servern unterzubringen, die sich irgendwo im weltweiten Netz befinden.

16.2.8 Remote Method Invocation

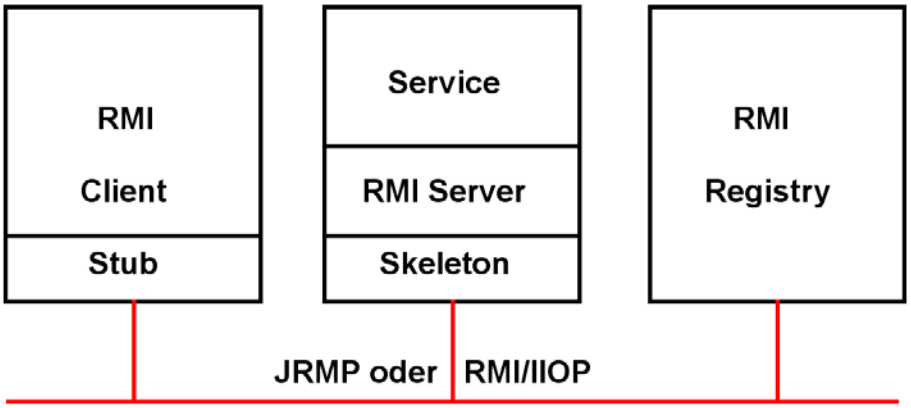


Abb. 16.2.7
Die Namensauflösung erfordert ein Registry

RMI erfordert drei verschiedene Prozesse, die auf dem gleichen Rechner oder auf entfernten Maschinen laufen können: Klient, Server und Namensdienst. RMI Registry ist ein einfacher RMI Namensdienst. Die Alternative ist JNDI

- **Der Klient besorgt sich eine Referenz (Handle) für das entfernte Objekt, indem er RMI Registry aufruft.**
- **Eine Referenz auf das entfernte Objekt wird zurückgegeben. Jetzt kann eine Methode des entfernten Objektes aufgerufen werden.**
- **Dieser Aufruf erfolgt zum lokalen Stub, der das entfernte Objekt repräsentiert.**
- **Der Stub verpackt die Argumente (Marshalling) in einen Datenstrom, der über das Netzwerk geschickt wird.**
- **Das Skeleton unmarshals die Argumente, ruft die Methode des RMI Services (Implementation) auf, marshals die Ergebniswerte und schickt sie zurück.**
- **Der Stub unmarshals die Ergebniswerte und übergibt sie an das Klientenprogramm.**

16.2.9 Zeitlicher Ablauf der RMI Remote Class Programmausführung

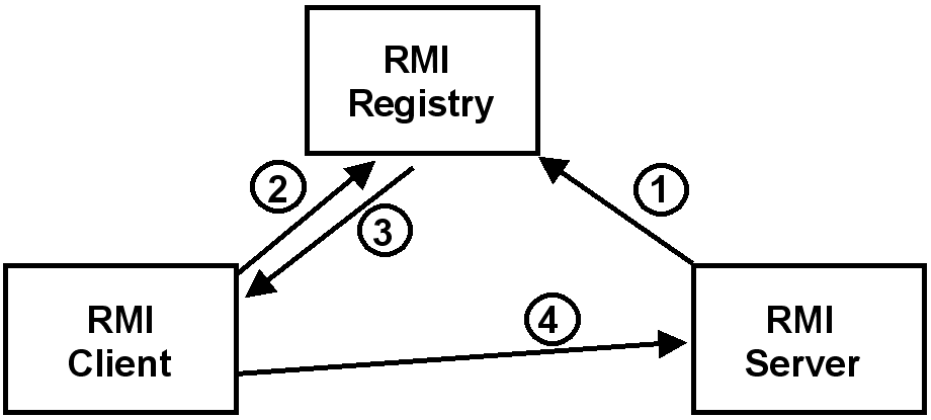


Abb. 16.2.8
Ein Server registriert jeden neuen Service in seinem Registry

Der RMI Server speichert zahlreiche Java Server Objekte (Services), auf die ein RMI Client zugreifen möchte. Hierzu:

1. **RMI Server speichert Objekt Namen und dazugehörige Objekt Referenz im RMI Registry Server. (Dieser läuft im Gegensatz zu anderen JNDI Implementierungen auf dem gleichen physischen Server wie der RMI Server).**
2. **Client verbindet sich mit dem Registry Server, und übermittelt den Objekt Namen (String Name) des gewünschten remote Objektes.**
3. **Registry stellt die Objekt Reference (Adresse) auf das remote Objekt zur Verfügung.**
4. **Zugriff auf den RMI Service.**

Ein Objekt Name ist vergleichbar mit einer URL wie z.B.
http://leia.informatik.uni-leipzig.de

Eine Objekt Referenz ist vergleichbar mit einer dazugehörigen IP Adresse wie z.B.
139.18.4.30

16.2.10 Registry Alternativen

Ein Namensdienst verwaltet Zuordnungen von Namen zu bestimmten Objekten. Alle Objekte werden nach einem standardisierten Namen angesprochen.

Ein Verzeichnisdienst (Directory Dienst) ist eine für Lese-Zugriffe optimiertet Datenbank, die Informationen in einem hierarchischen Informationsmodell speichert. Die in einem Namensdienst definierten Namen können in einem Verzeichnisdienst gespeichert werden; ein Verzeichnis Dienst ist eine Art Datenbank, die im Gegensatz zu einem Namensdienst weitere Informationen speichern kann, z.B. die geografische Lokation eines Services oder Servers.

Ein Namens- und Verzeichnisdienst ermöglicht es, diesen (und auch die Services) auf physisch unterschiedlichen Servern unterzubringen, die sich irgendwo im weltweiten Netz befinden.

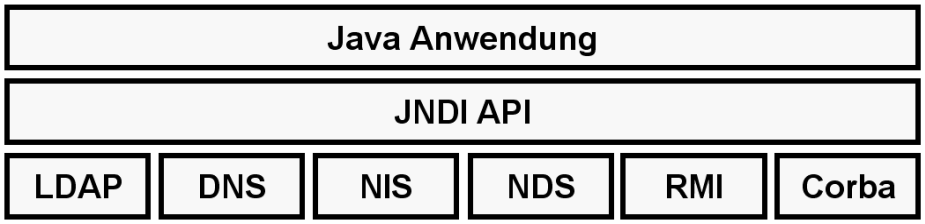


Abb. 16.2.9
Hinter der JNDI können sich beliebige Verzeichnisdienste verbergen

JNDI (Java Naming and Directory Interface) ist eine API, über die ein Java Programm einen Namens- und Directory Dienst in Anspruch nehmen kann. Der Namens- und Directory Dienst ermöglicht einen Lookup von RMI Objekten (Services) zwischen entfernten JVMs. Mögliche Implementierungen der JNDI Schnittstelle sind z.B. z.B. LDAP Server, Corba Common Object Services Naming (COSNaming) Server oder der rmiregistry Server. rmiregistry ist ein Bestandteil des JDK. JRMP verwendet standardmäßig einen rmiregistry Server.

Wie in Java RMI, erfolgt der Zugriff auf ein CORBA Server Objekt mit Hilfe einer Objekt-Referenz. Ein CORBA Server ORB ermöglicht das Registrieren einer Objekt-Referenz mit einem Corba konformen Verzeichnisdienst (Common Object Service (COS) Naming Server). Der CORBA COS (Common Object Services) Naming Service verfügt über eine Baum-artige Directory Struktur für Object Referencen.

Da CORBA im Gegensatz zu RMI sprachenabhängig ist, ist eine CORBA Objekt-Referenz eine abstrakte Entität. Diese wird von einem Client ORB in eine Sprachen-spezifische Objekt-Referenz abgebildet. CORBA-Objekt Referenzen werden als Interoperable Object References (IORs) bezeichnet.

Aus Kompatibilitätsgründen verlangt RMI/IIOP (siehe unten) die Benutzung von COS Naming für Zugriffe auf einen Corba Server.

16.2.11 Erstellen einer RMI Remote Class

Um entfernte Objekte mit ihren Methoden in Java-Programmen zu nutzen, müssen wir die folgenden Schritte durchführen:

1. **Wir definieren eine remote Schnittstelle eines geplanten Server Objektes, welche die Methode(n) des Server Objekts definiert.**

2. **Wir implementieren eine Klasse, die diese Schnittstelle implementiert und die Methoden mit Leben füllt. Dies bildet das entfernte Objekt (Service, Implementation)**

3. **Existiert die Implementierung, benötigen wir ein Exemplar dieses Objekts. Wir melden dieses bei einem Namensdienst an, damit andere es finden können. Dies bildet den Service.**

Wir implementieren einen Klienten und greifen damit auf die entfernte Methode zu.

RMI benötigt (wie Corba) einen RMI Server, unter dem die RMI Implementation (der Service) läuft.

Zu kodieren sind:

- **Interface**
- **Implementation**
- **Server**
- **Client**

Die Remote Interface enthält die Namen aller Methodenaufrufe und die dazugehörigen Parameter. Beispiel für die Interface einer Methode Addition, die zwei Zahlen a und b addiert:

```
public interface Addition extends
                java.rmi.Remote
{
     public long add(long a, long b)
     throws java.rmi.RemoteException;
}
```

Schritte zur Erstellung und Ausführung einer Anwendung:

1. **Interface definieren, mit der das Remote Object aufgerufen wird.**
   > extends interface java.rmi.Remote .

2. **Implementierung der Server Anwendung schreiben. Muss die Remote Interface implementieren. .**
   > extends java.rmi.server.UnicastRemoteObject

3. **Klassen kompilieren.**

4. **Mit dem Java RMI Compiler Client Stubs und Server Skeletons erstellen. > rmic xyzServerImpl**

5. **Klient implementieren und übersetzen.**

6. **Start Registry, Server starten, Klienten starten.**

16.2.12 RMI Performance

Table 1: Passing bytes by value

Function}	# of args	# of results	Local Call(sec)	Remote Call(sec)
Null	0	0	15.172	17.345
sendbyte	1	0	15.332	17.054
recvbyte	0	1	15.292	17.105

Table 2: Passing fixed length byte arrays

Function}	# of bytes (args)	# of bytes(results)	Local Call(sec)	Remote Call(sec)
senddata	1440	0	25.537	34.209
recvdata	0	1440	27.099	33.408

Abb. 16.2.10
RMI Performance Ergebnisse

RMI benötigt viele CPU Zyklen für die Ausführung. Die gezeigten Daten stammen von einer Untersuchung
http://www-csag.ucsd.edu/individual/achien/cs491-f97/projects/hprmi.html, oder
http://www.cedix.de/VorlesMirror/Band2/Perform02.pdf

Gemessen wurde die benötigte Zeit für 10 000 RMI Aufrufe zwischen 2 JVMs auf dem gleichen Rechner (hier als Local Call bezeichnet) und 2 JVMs auf 2 getrennten Rechnern verbundenen über ein Ethernet (Remote Call). Auf den Rechnern lief Windows NT; die Messungen erfolgten 1997.

Die Zeit pro Aufruf liegt im Millisekunden Bereich. Die Schlussfolgerung ist: "Java RMI performs poorly in comparison to other RPC systems" (z.B. DCE RPC).

Auch wenn heutige Rechner deutlich schneller sind, ist der Overhead beträchtlich. Im Internet existieren zahllose Untersuchungen und Vorschläge zum Thema RMI Performance Tuning – nicht ohne Grund.

Dennoch ist Java RMI ein bequemes und populäres Verfahren, um Remote Method Calls in verteilten (distributed) Object Systemen zu implementieren. Auch ist die RMI Performance deutlich besser als die Web Service RPC Performance, see:
„Comparison of Performance of Web services, WS-Security, RMI, and RMI–SSL". Journal of Systems and Software 79 (2006) 689–700.
http://www.cedix.de/VorlesMirror/Band2/Perform03.pdf

16.2.13 DCE RPC mit Java Klienten ohne Objektorientierung

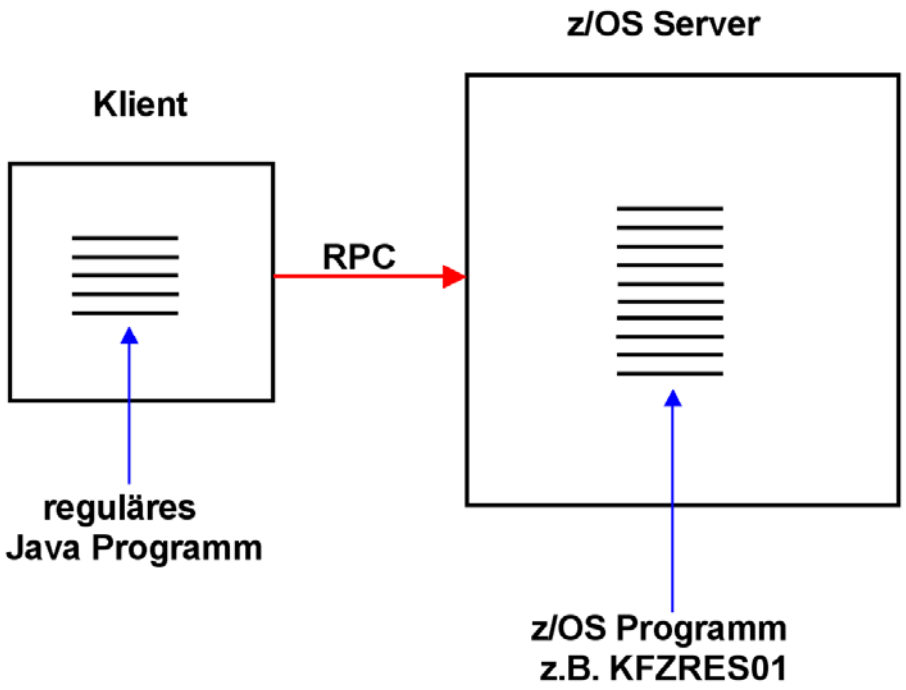


Abb. 16.2.11
Eine Java zu Java Kommunikation kann auch über den klassischen RPC erfolgen

Klassische Remote Procedure Calls (RPC) sind der Sun und der DCE Remote Procedure Calls.

Corba, RMI und Web Services (SOAP) sind moderne Versionen des RPC. Corba und RMI sind inhärent objekt-orientiert.

Es muss nicht alles RMI sein. Man kann auch Client/Server Anwendungen in Java schreiben ohne Benutzung von RMI. Hierbei gehen die Vorteile der Objektorientierung allerdings verloren.

Gezeigt ist ein Beispiel, wo ein Java Programm einen z/OS Service unter Benutzung des klassischen Remote Procedure Calls aufruft.

Der DCE RPC ist der bevorzugte klassische RPC unter z/OS (und auch Windows). Die DCE Software ist Bestandteil von z/OS.

16.2.14 Aufruf eines z/OS Programms durch einen Java Klienten

```
/** *Folgende Methode ruft einen RPC
     auf dem Mainframe auf. */

private boolean rpcCall(String inputString,
                        StringBuffer outBuffer){
  RPC m_rpc =new RPC();
  try {
    //Mainframe braucht UserID und Password!
    m_rpc.setUser("k3216 ");
    m_rpc.setPwd("TESTPW ");
    //Name des aufzurufenden Programms setzen
    m_rpc.setRpcName("KFZRES01 ");
    //remote procedure call ausführen
    m_rpc.execute(inputString,outBuffer);
    return true;
  }
  catch (Exception e) {
    m_stateField.setText("Execute-Error:"
                         +m_rpc.getApiMessage());
    return false;
  }
}
```

Abb. 16.2.12
Client Programm für den Aufruf eines z/OS Service

Die Integration von z/OS Mainframe-Programmen in Java-basierte Client/Server-Systeme funktioniert prinzipiell recht einfach. In der folgenden Methode wird ein z/OS RPC-Programm namens „KFZRES01“ aufgerufen. Diesem Programm werden zwei Zeichenketten (Strings) übergeben, einer für die Eingabeparameter, der zweite für die durch KFZRES01 erzeugte Ausgabe.

Seitens der z/OS Mainframe-Programme ändert sich rein gar nichts gegenüber der Nutzung per 3270-Terminal. Das Programm KFZRES01 „weiß “also nicht, ob es durch ein Java-Programm benutzt wird oder über konventionelle z/OS Mainframe-Terminals aufgerufen wird.

16.3 RMI over IIOP

16.3.1 Warum RMI zusätzlich zu CORBA ?

Neben der reinen Java-Lösung RMI gibt es auf dem weiten Feld der Standards noch das komplexere CORBA.

RMI setzt voraus, dass Client und Server in Java geschrieben sind. Im Gegensatz zu RMI definiert CORBA ein großes Framework für unterschiedliche Programmiersprachen. Die Definition von CORBA durch die OMG (Object Management Group) geht in das Jahr 1991 zurück, also auf die Zeit vor RMI.

Die Frage nach dem Sinn von RMI ist also erlaubt. Die Antwort liegt in der Einfachheit und Integration von RMI.

Seit 2001 hat die Firma Sun RMI an den CORBA Standard angepasst. Die Stellvertreter-Objekte (Stubs, Skeletons) unterstützen mittlerweile nicht nur das Java eigene JRMP Protokoll, sondern zusätzlich auch das CORBA eigene Inter-ORB Protocol (IIOP). Diese Lösung heißt RMI/IIOP („RMI over IIOP“).

Mit RMI/IIOP lässt sich sowohl eine Verbindung zwischen zwei in Java geschriebenen Objekten, als auch eine Verbindung zwischen Java Objekten und nicht-Java Objekten herstellen.

Vergleich CORBA – RMI:

- **Corba Anwendungen können in vielen unterschiedlichen Sprachen geschrieben werden, solange für diese Sprachen ein „Interface Definition Language (IDL) mapping“ vorhanden ist. Dies ist der Fall für Cobol, PL/1, C/C++, Ada, Fortran, SmallTalk, Perl, Ruby, Python und viele weitere Sprachen. RMI in der JRMP Version ist auf Java beschränkt.**

- **Mit der IDL (Interface Definition Language) ist die Interface von der Implementierung sauber getrennt. Es können unterschiedliche Implementierungen unter Benutzung der gleichen Interface erstellt werden. RMI Anwendungen sind einfacher zu programmieren als Corba Anwendungen, weil die Notwendigkeit der IDL Definition entfällt. Die Interface ist bereits ein Sprachkonstrukt von Java.**

- **RMI ermöglicht serialisierbare Klassen. Code und Objekte können über das Netz übertragen werden (can be marshaled), solange der Empfänger über eine Java Virtuelle Maschine (JVM) verfügt. CORBA erlaubt keine Übertragung von Code oder Objekten; es können nur Datenstrukturen übertragen werden.**

- **Corba hat ein besseres Leistungsverhalten als RMI (keine Interpretation).**

Die Vorteile beider Ansätze lassen sich durch den Einsatz des RMI/IIOP Protokolls kombinieren.

16.3.2 Vor dem Erscheinen von RMI/IIOP

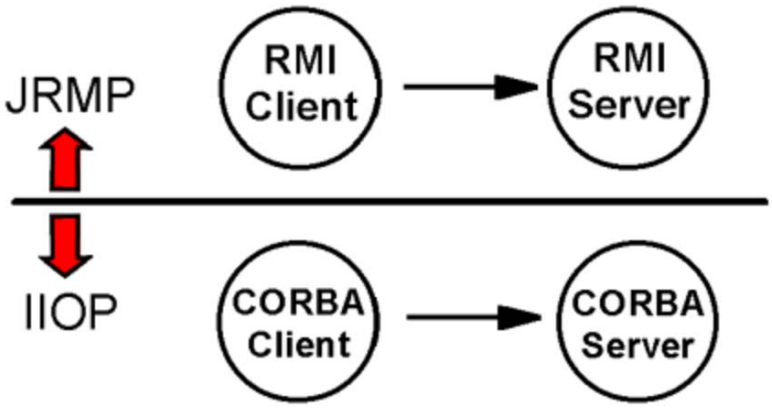


Abb. 16.3.1
Mit JRMP ist keine Kommunikation zwischen RMI und Corba möglich

Schauen Sie sich Abb. 16.3.1 an. Der Raum über der mittleren horizontalen Linie stellt die ursprünglichen Domäne von RMI und dessen JRMP Protokoll dar; der untere Bereich stellt die Welt von CORBA und IIOP dar. Diese beiden getrennten Welten, die unabhängig voneinander entwickelt wurden, sind historisch nicht imstande gewesen, miteinander zu kommunizieren. Zum Beispiel kann das native RMI Protokoll JRMP (Java Remote Method Protocol) sich nicht mit IIOP (dem Corba Protokoll), oder anderen Protokollen verbinden.

16.3.3 Interoperability mit CORBA

Wenn Java die einzige Programmiersprache ist, die Sie in einem neuen Projekt benötigen, können Sie RMI und JRMP (siehe Abb. 16.3.1) verwenden.

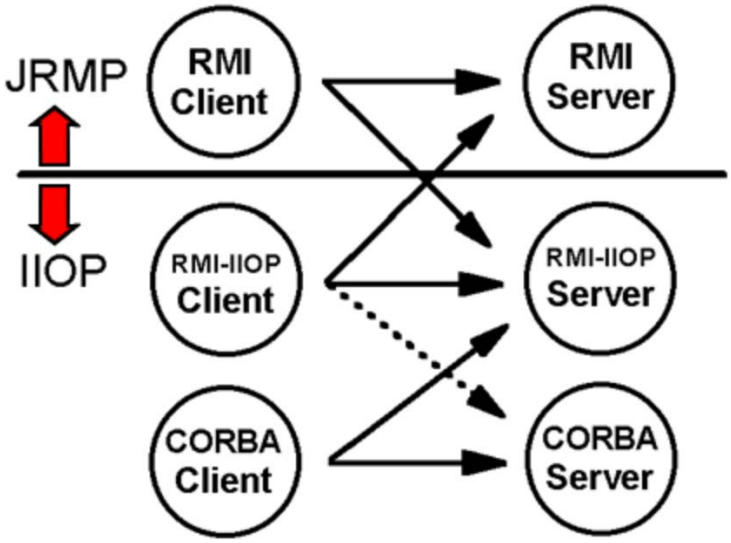


Abb. 16.3.2
Kommunikation zwischen RMI und Corba

CORBA auf der anderen Seite ermöglicht bei der Programmierung von Anwendungen mit verteilten Objekten die Benutzung unterschiedlicher Programmiersprachen. Verteilte Objekte werden nicht nur in neuen Projekten benutzt. Sie treten auch bei der Erweiterung älterer Software-Ressourcen (Legacy Software) auf. Legacy-Software ist in den meisten Fällen in anderen Sprachen als Java programmiert. In solchen Situationen müssen die Entwickler das Corba IIOP Protokoll, nicht RMI/JRMP einsetzen.

Um die komplexe Corba Programmierung zu vermeiden, können Java Programmierer an Stelle von JRMP das Corba kompatible RMI/IIOP Protokoll einsetzen.

In Abb. 16.3.2 stellt der obere Abschnitt der RMI/JRMP Modell, der mittlere Abschnitt der RMI/IIOP Modell, und der untere Abschnitt das CORBA Modell dar. Ein Pfeil stellt eine Situation dar, in der ein Client einen Server anruft. RMI/IIOP gehört in die IIOP Welt unterhalb der horizontalen Linie.

Interessant sind die diagonalen Pfeile, die die Grenze zwischen der JRMP Welt und der IIOP Welt überschreiten. Danach kann ein RMI/JRMP Client auf einen RMI/IIOP-Server zugreifen, und umgekehrt. RMI/IIOP unterstützt sowohl JRMP als auch IIOP Protokolle.

Eine Server Java Binary (d.h. eine Class File), die mit RMI/IIOP APIs erstellt wurde, kann entweder als JRMP oder als IIOP exportiert werden. Beim Wechsel von JRMP nach IIOP, oder umgekehrt, ist es nicht erforderlich, den Java-Quellcode umzuschreiben oder neu zukompilieren. Es ist nur erforderlich, Parameter wie Java System Properties zu ändern. Alternativ können Sie das zu benutzende Protokoll bestimmen, indem Sie es in dem Java-Quellcode spezifizieren.

Die diagonalen Pfeile in der Abbildung oben sind möglich, weil die RMI/IIOP APIs sowohl JRMP als auch IIOP Protokolle unterstützen. Dies bedeutet, dass ein RMI/JRMP Server Objekt ohne Umschreiben des Quellcodes durch einen neuen RMI/IIOP Client aufgerufen werden kann. Ebenso kann ein RMI/JRMP Server Objekt durch ein neues RMI/IIOP Objekt ersetzt werden, ohne Umschreiben des Quellcodes eines RMI/JRMP Clients.

Schauen wir wieder auf Abb. 16.3.2. Der Abschnitt unterhalb der horizontalen Linie ist die IIOP Welt, wo ein RMI/IIOP Client einen CORBA-Server aufruft und ein CORBA Client einen RMI/IIOP Server aufruft. Mit RMI/IIOP Client meinen wir ein Client-Programm, das von einem RMI-Programmierer, der nichts über CORBA oder IDL weis, geschrieben wurde. Ebenso ist ein CORBA-Client ein Client-Programm, das von einem CORBA-Programmierer ohne RMI Kenntnisse geschrieben wurde. Die Trennung der Interface von der Implementierung ist eine gut etablierte Technik. Sie ermöglicht es einem Programmierer auf verschiedene Ressourcen zuzugreifen ohne Wissen wie diese implementiert wurden. Benutzer sowohl von RMI/IIOP als auch von CORBA können die Dienste des anderen Protokolls verwenden , wenn sie Zugang zu seiner Interface bekommen. Ein RMI Java Interface-Datei ist die Interface für RMI/IIOP Benutzer, während IDL die Interface für CORBA Benutzer definiert. Die Interoperabilität zwischen RMI/IIOP und CORBA in der obigen Abbildung wird erreicht, indem jedem Benutzer seine erwartete Interface zur Verfügung gestellt wird.

Als letztes Detail in der obigen Abbildung ist der gepunktete Pfeil zu erklären. Er zeigt einen RMI/IIOP Client, der einen CORBA Server aufruft. Warum ist dieser Pfeil gepunktet ? Ein RMI/IIOP Client kann nicht unbedingt auf alle vorhandenen CORBA Objekte zugreifen. Die Semantik der in IDL definierten CORBA-Objekte kann Funktionen enthalten, die in RMI/IIOP Objekten nicht vorhanden sind. Eine bestehende CORBA Objekt IDL kann nicht immer in einem RMI/IIOP Java-Interface abgebildet werden. Der gepunktete Pfeil zeigt, dass eine Verbindung zu einem bestehende CORBA-Server-Objekt manchmal - aber nicht immer - möglich ist.

Diese Einschränkung gilt nicht für neu entwickelte Nicht-Java-CORBA-Server-Objekte (zum Beispiel geschrieben in C/C++). Es ist einfach, ein RMI/ IIOP Interface hierfür zu erzeugen.

Um zusammenzufassen: Wenn Sie Ihren Server in RMI/IIOP implementieren, haben Sie die größte Auswahl an Klienten. Ebenso, wenn Sie Ihren Klienten in RMI/IIOP implementieren, können Sie mit der größten Auswahl an Servern kommunizieren. Hierbei kann es einige Beschränkungen im Fall von bestehenden CORBA-Objekte geben.

16.3.4 Interoperability mit CORBA

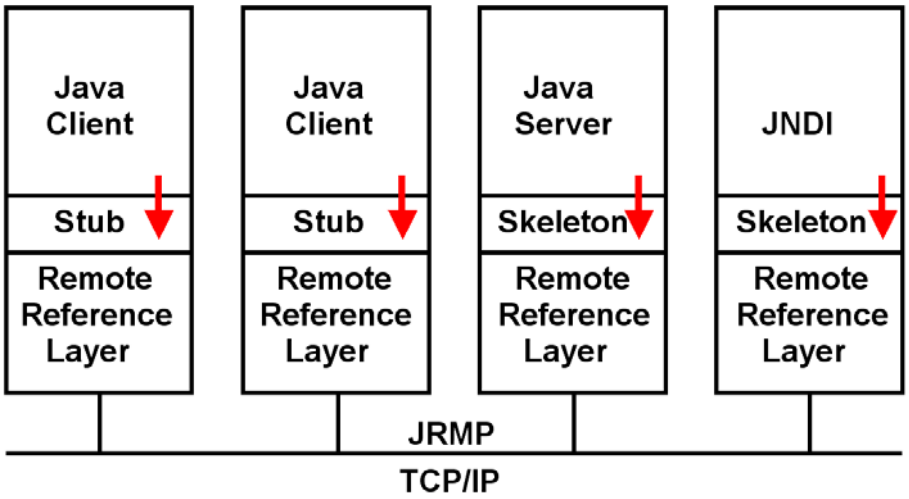


Abb. 16.3.3
JRMP Kommunikation

Java Client und Server benutzen die RMI interface ↓ um mit Stub und Skeleton zu kommunizieren. Der JDK enthält die "Remote Reference Layer" Komponente, welche alle Communication Funktionen implementiert. Es stellt auch die Stub und Skeleton Interfaces mit TCP/IP in OSI Layer 4 zur Verfügung.

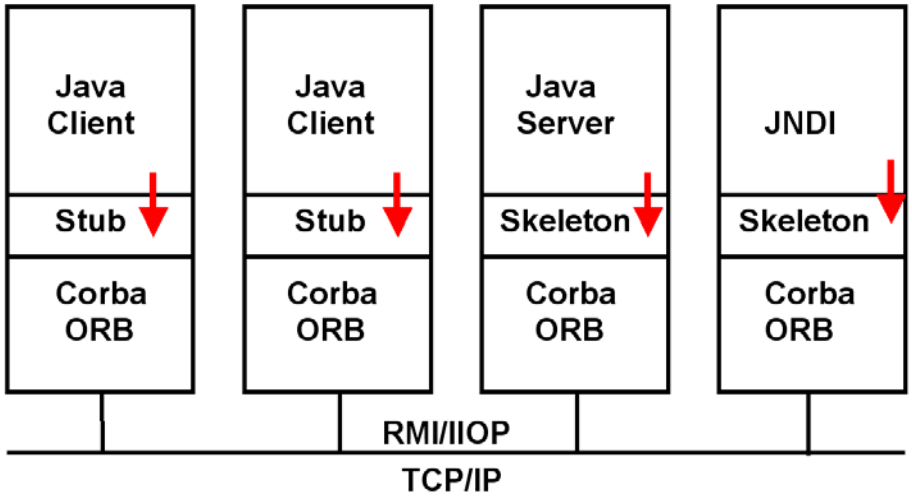


Abb. 16.3.4
RMI/IIOP Kommunikation

Bei der Benutzung von RMI/IIOP wird die Remote Reference Layer durch den Corba Object Request Broker (ORB) ersetzt.

Die RMI Interface, die vom Java Client und Java Server Object Code für die Kommunikation mit Skeleton und Stub benutzt wird, verändert sich nicht.

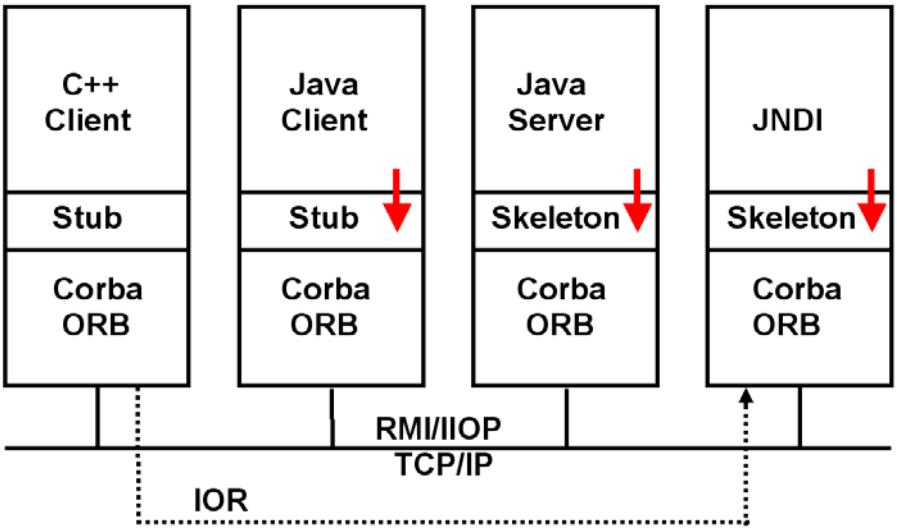


Abb. 16.3.5
Ein C++ Klient greift mittels RMI/IIOP auf einen Java Server zu

Wenn ein Corba Object Request Broker (ORB) für alle Java Komponenten benutzt wird, kann ein non-Java Member der Gruppe beitreten (ein C++ Klient in diesem Beispiel).

Die RMI Interface, die vom Client und Server Object Code für die Kommunikation mit Skeleton und Stub benutzt wird, verändert sich nicht.

Der vorhandene Java JEE Standard legt fest, dass alle Java Software Produkte RMI/IIOP unterstützen. Die Unterstützung für JRMP ist optional.

In Übereinstimmung mit dieser Spezifikation unterstützen IBM WebSphere und CICS unter z/OS nur RMI/IIOP. Ihre EJB Container haben die Funktionalität eines CORBA ORB.

Interessanterweise ist die Performance von RMI/IIOP häufig besser als die von JRMP.

16.3.5 JRMP Entwicklung und Ausführung

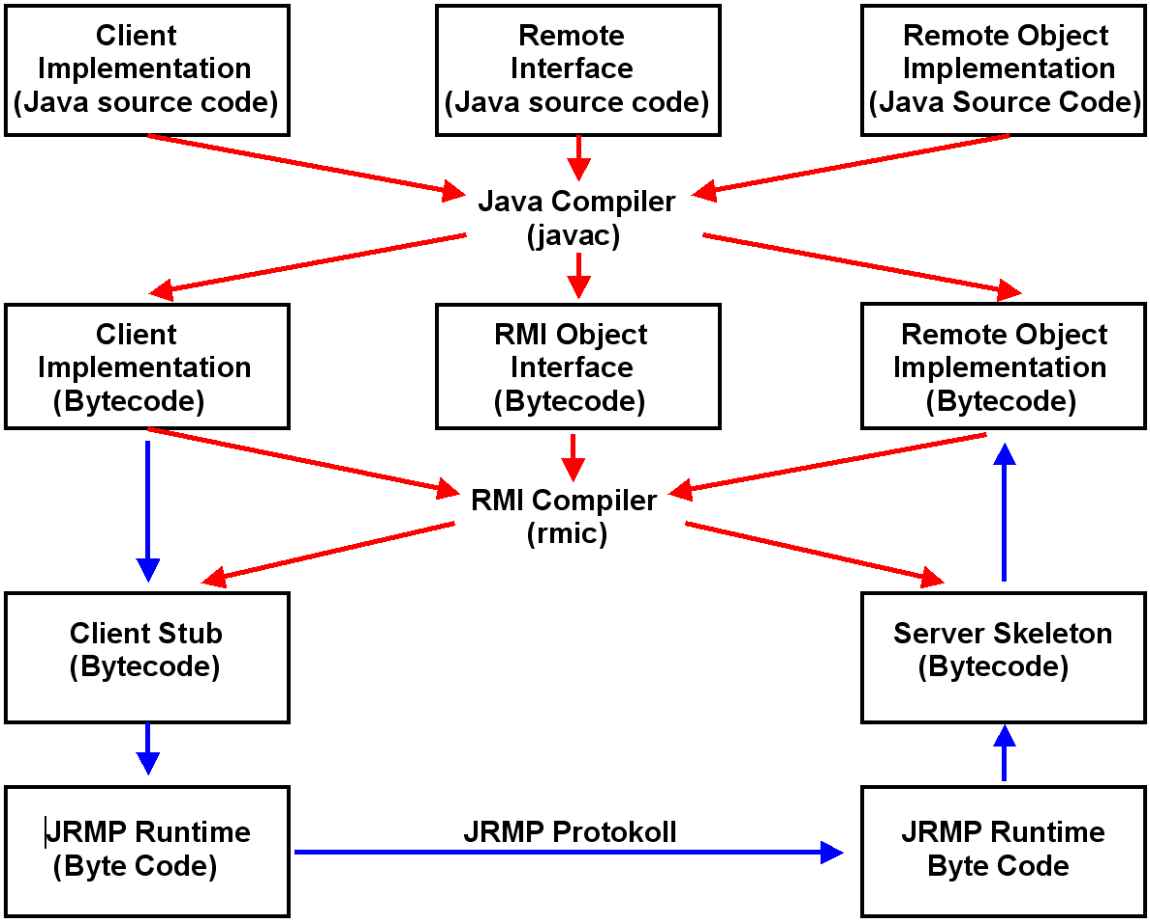


Abb. 16.3.6
Entwicklung und Ausführung mittels JRMP

In Abb. 16.3.6 ist die Entwicklung für den JRMP Server und den JRMP Klienten dargestellt. Folgen Sie den roten Pfeilen:

Der Quellcode der Server-side Object Implementierung und deren Java-Interface werden wie gewohnt mit dem regulären Java-Compiler (`javac`) übersetzt. Dies generiert Byte-Code.

Der Byte-Code wird als Eingabe für den Java RMI-Compiler (`rmic`) verwendet, zusammen mit der Remote-Interface Beschreibung des Java-Server-Objekts. Hiermit wird eine Byte Code Version sowohl des Server Skeletons als auch des Client-Stubs generiert.

Folgen Sie nun den blauen Pfeilen.

Wenn der Bytecode der Client-Implementierung das Remote-Objekt aufruft, überträgt es eine Nachricht an den Client-Stub. Die Client-Stub-Klassen übernehmen die Funktionen, die unique für diese Client-Implementierung sind. Die JRMP Laufzeitklassen führen die generischen JRMP Funktionen aus und starten die Kommunikation durch die Übertragung eines JRMP Nachricht an die TCP-Komponente des Client-Systems.

16.3.6 RMI/IIOP Entwicklung

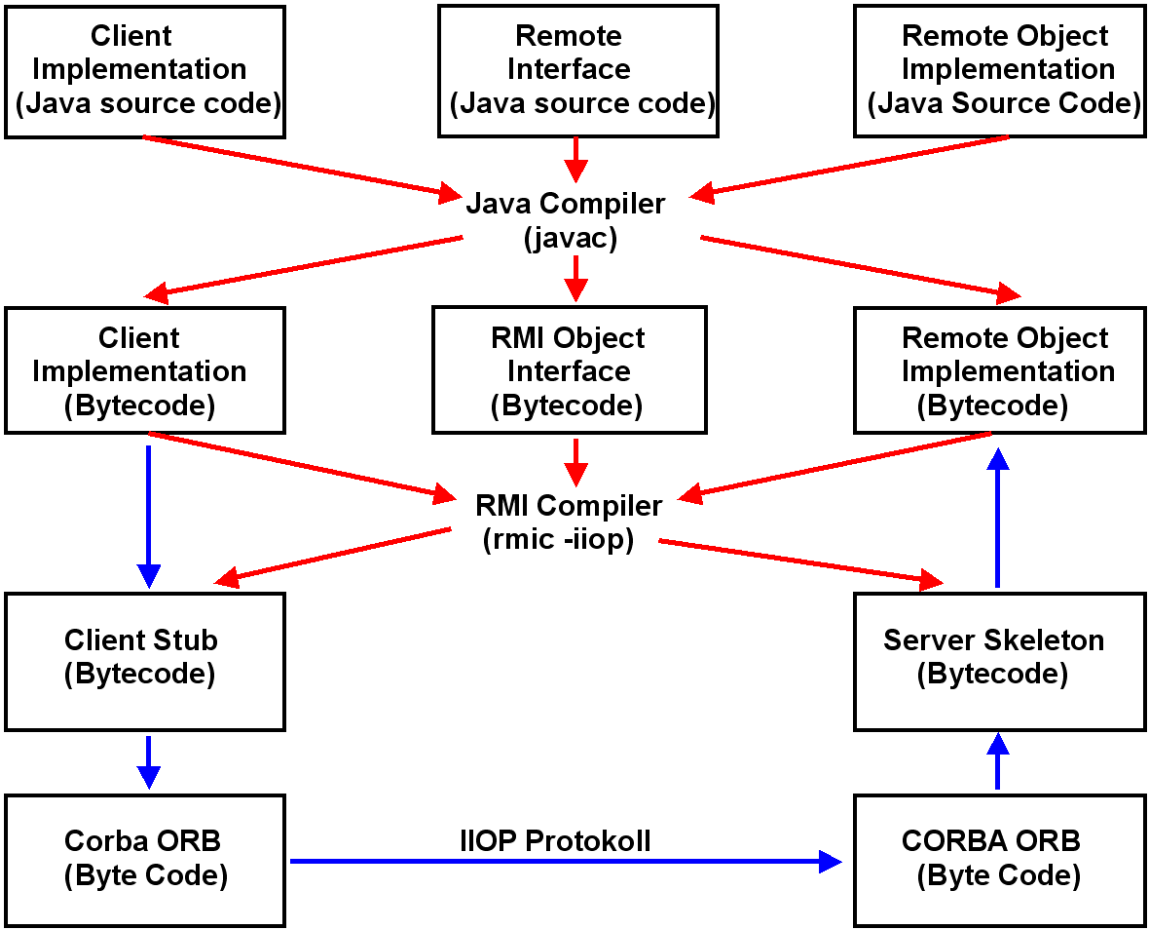


Abb. 16.3.7
Entwicklung und Ausführung mittels RMI/IIOP

Abb.16.3.7 zeigt das Entwicklungs-Verfahren für den RMI/IIOP Server und Klienten. Es entspricht Abb.16.3.6 . Sie werden feststellen, es ist fast das gleiche Verfahren wie bei RMI/JRMP.

Genauso wie bei RMI/JRMP enthält eine RMI Java-Interface die verteilte Server-Objekt-Definition. Ein Unterschied ist der `iiop` Parameter des `rmic` Compilers. Diese Option wird benutzt, damit `rmic` Stubs erzeugt werden, und diese mit dem IIOP Protokoll anstelle des JRMP Protokolls verknüpft werden. Obwohl die Entwicklungsverfahren für RMI/IIOP fast die gleichen wie für RMI/JRMP sind, ist die Laufzeitumgebung verschieden. Die Kommunikation erfolgt über einen CORBA-konformen ORB. IIOP wird für die Kommunikation zwischen Server und Client benutzt.

RMI/IIOP verwendet den Java CORBA Object Request Broker (ORB) und IIOP. Sie können den gesamten Code in der Programmiersprache Java schreiben, und den `rmic` Compiler (mit der `iiop` Option) benutzen. Die resultierende Java Anwendung kann über das Internet InterORB Protocol (IIOP) mit anderen in einem CORBA-kompatiblen Sprache geschrieben Anwendungen kommunizieren.

Der J2EE-Standard bietet zwei alternative Protokolle: JRMP und RMI/IIOP. Einige Hersteller unterstützen beide RMI Protokolle. IBM hat beschlossen, ausschließlich RMI/IIOP in ihren Produkten zu verwenden. Dies ist konform mit dem RMI-Standard.

16.3.7 Coexistence Beispiel: C++ client ruft ein Java RMI Object auf

CORBA und RMI/IIOP verwenden den gleichen Internet Inter-ORB Protocol Kommunikationsstandard. Wenn erforderlich ist es möglich, die IDL-Definitionen für die beteiligten RMI/IIOP Datenstrukturen erzeugen. Diese Definitionen können benutzt werden, um die Interoperabilität zwischen den RMI/IIOP Anwendungen und normalen CORBA-Anwendungen zu gewährleisten.

Mit RMI/IIOP können Entwickler Remote Interfaces in der Java-Programmiersprache schreiben und nur mit Hilfe von Java-Technologie und den Java RMI APIs implementieren. Diese Interfaces können in jeder anderen von CORBA unterstützten Sprache implementiert werden, vorausgesetzt es existiert ein ORB für diese Sprache. Ebenso können Klienten, die in anderen Sprachen geschrieben sind, IDL benutzen, die von den Remote-Java-Technologie basierenden Interfaces abgeleitet ist.

Verwenden Sie den RMI/IIOP Compiler mit der `iiop` Option um Stubs und Skeletons für Remote-Objekte zu erzeugen, welche das IIOP-Protokoll verwenden. Der `rmic` Compiler kann auch benutzt werden, um IDL für CORBA Entwicklungen auf der Java-Plattform zu erzeugen.

Als Beispiel betrachten wir den Fall, wo ein C++ CORBA Client ein RMI-Server-Objekt aufruft

Der `rmic-iiop` Compiler generiert Stubs und Skeletons für Java-Clients und Server.

Die C++ CORBA Client benötigt einen C++ CORBA IIOP Stub um das RMI-Server Objekt aufzurufen. Stubs werden mit Hilfe der Interface Definition des Server-Objektes erzeugt. Hierzu brauchen wir eine normale Corba IDL Beschreibung der Server Interface. Die Beschreibung der Server Interface liegt aber als Java Interface, und nicht als Corba IDL Interface vor.

Um den Client-Stub zu generieren, beginnen wir mit dem Java-Server-Interface.

Abb. 16.3.8
Generierung einer Corba Kompatiblen IDL File

Wie in Abb. 16.3.9 dargestellt, verwenden wir die Java-Interface und den `rmic` Compiler mit der `idl` Option um eine IDL-Beschreibung der Interface (IDL-Datei) zu generieren.

Mit dem IDL Beschreibung der Interface können wir jetzt den Client-Stub generieren.

Die IDL-Beschreibung, zusammen mit dem Client C++-Code wird von dem regulären CORBA-Compiler (der idl2cpp Compiler) verwendet, um den Code für CORBA C++ kompatible Stubs und Skeletons zu erzeugen. Der Stub wird mit dem C++-Client integriert, das Skelett wird verworfen, da das Java RMI-Objekt das regulären Skeleton verwendet, das durch `die rmic-iiop` Compiler generiert wurde.

16.3.8 Generating the Client Stub Code

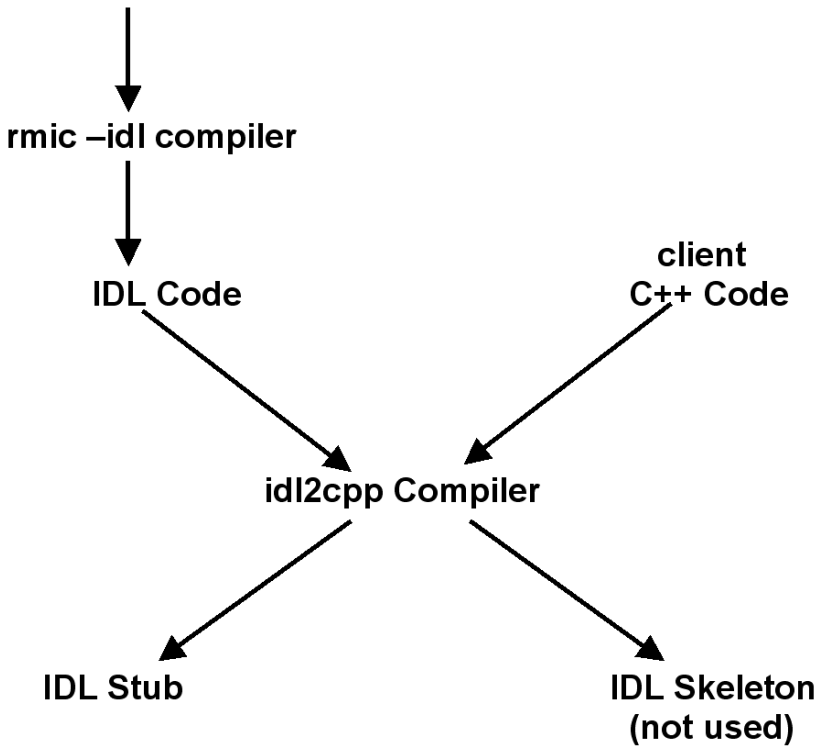


Abb. 16.3.9
Erstellen des Stub Codes für den C++ Client

Der idl2cpp Compiler wird von CORBA standardmäßig benutzt, um für ein C++ Objekt Skeleton und Stub zu erzeugen. Ähnliche Compiler existieren, um für Cobol, ADA, PL/1 Objekte Skeleton und Stub zu erzeugen. Weil der Server die Java Interface benutzt, wird das Server Skeleton vom rmic Compiler mit der iiop Option erzeugt.

Es ist möglich, eine Java Client/Server Anwendung mit Corba an Stelle von RMI zu entwickeln. Hierbei würden Stubs und Skeletons mittels des Corba IDL Compilers für Java erzeugt. Die Entwicklung ist aber nicht mehr so einfach wie unter RMI.

16.3.9 WebSphere CORBA Unterstützung

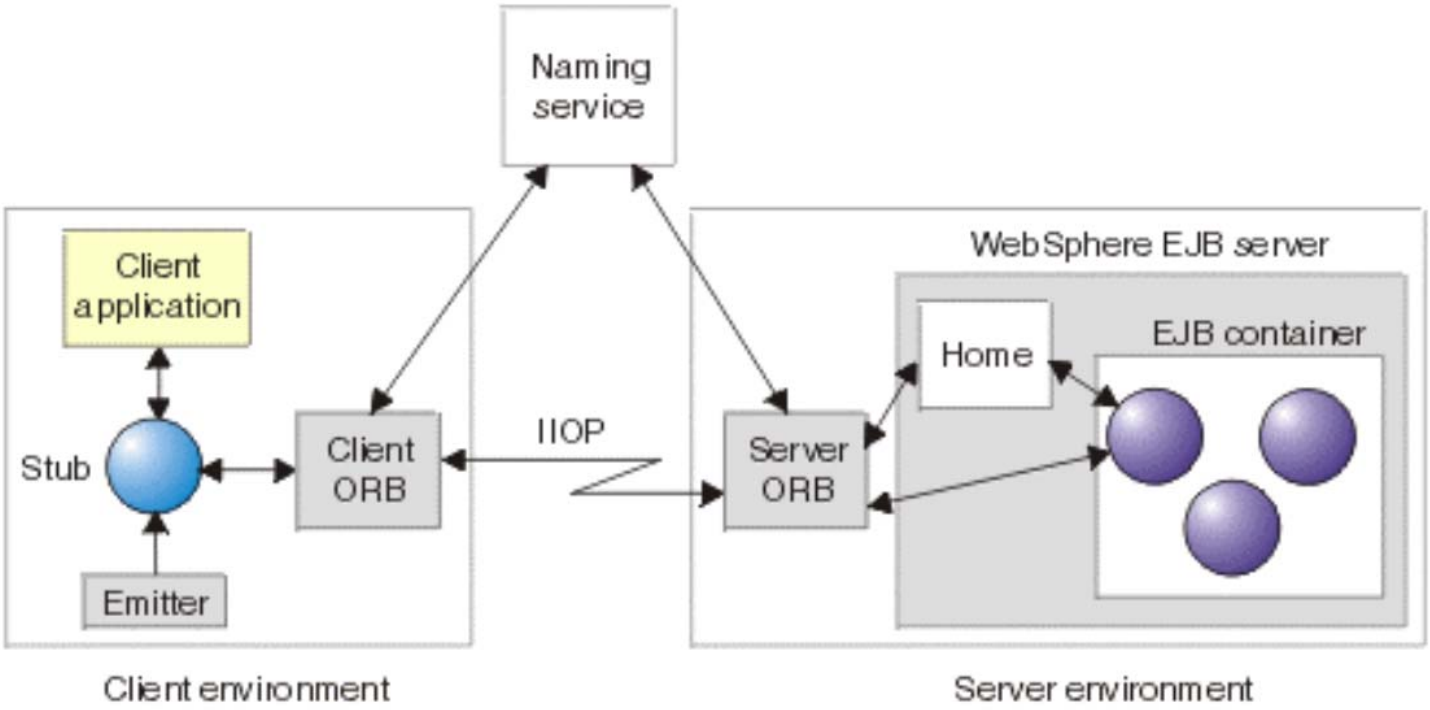


Abb. 16.3.10
WebSphere beinhaltet einen regulären CORBA ORB

Die IBM WebSphere Web Application Server beinhaltet CORBA-Unterstützung. Diese ermöglicht den Einsatz von CORBA-Interfaces zwischen einem Server-Objekt, das einen Service anbietet, und einem Client, der diesen Service benutzt. Diese Option ist zusätzlich zu den normalen Java Interfaces verfügbar. In der Praxis bedeutet dies:

- **WebSphere C++ CORBA-Server und WebSphere EJB-Services können von CORBA-Clienten aufgerufen werden, und**
- **WebSphere CORBA-Clienten können auf beliebige CORBA-Server zugreifen.**

Als Teil der WebSphere JEE-Umgebung beinhaltet die C++ CORBA-Unterstützung eine zusätzliche Basis CORBA Umgebung an. Diese kann in den JEE Namensraum integriert werden, und kann JEE-Transaktionen aufrufen.

16.4 Weiterführende Information

http://www.cedix.de/VorlesMirror/Band2/RmiBeispiel.pdf

enthält ein einfaches, aber detailliertes RMI Programmierbeispiel, welches Sie auf Ihrer Workstation ausführen können.

Ein weiteres RMI Tutorial finden sie unter

http://www.cedix.de/VorlesMirror/Band2/RmiBeispiel02.pdf

Die URL

http://www.youtube.com/watch?v=CLjyhH28AnE

zeigt ein Video, in dem RMI unter Eclipse entwickelt wird

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.

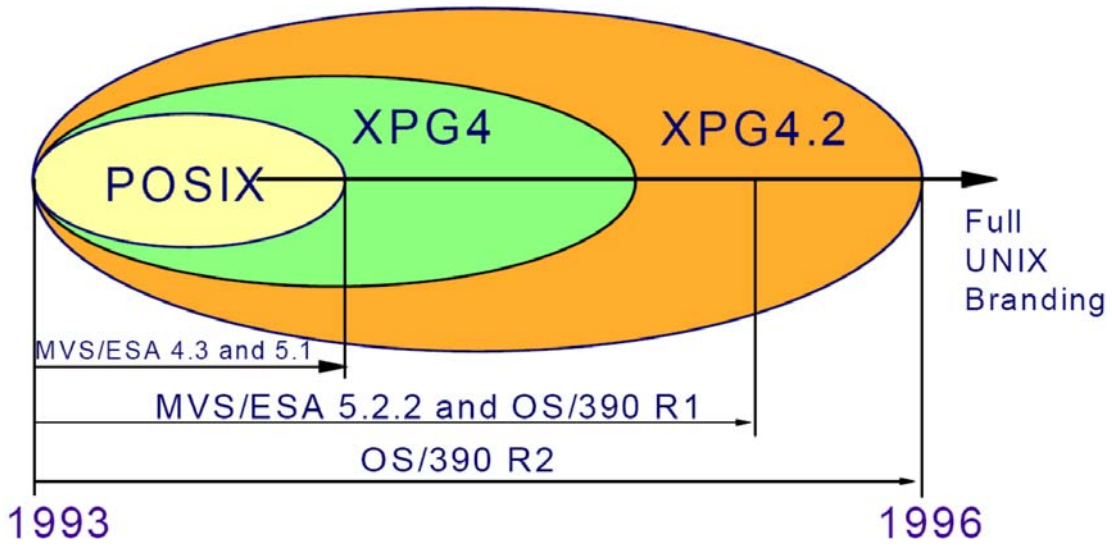


Abb.17.1.1
Posix und XPG4 Modell

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

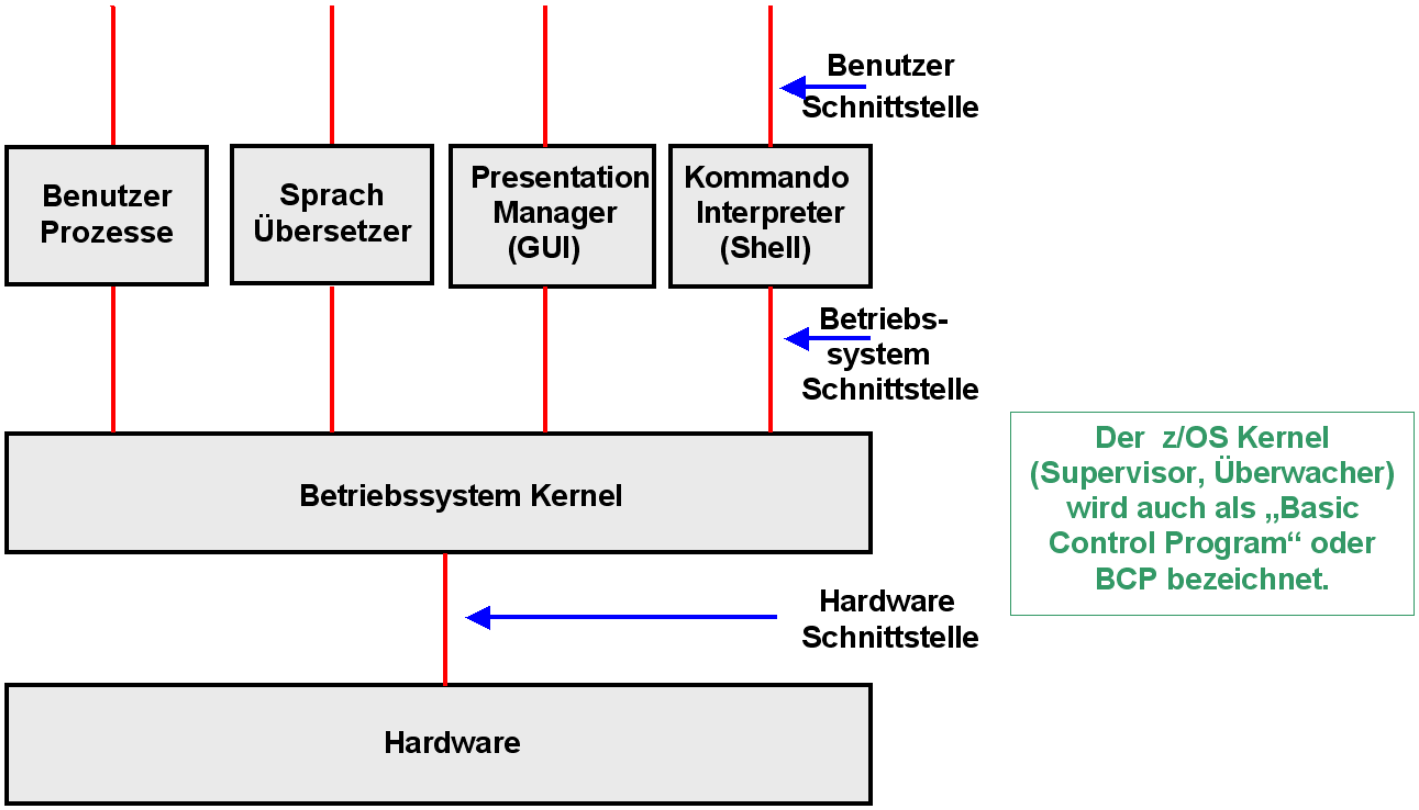


Abb.17.1.2
Schichtenmodell der Rechner Architektur

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

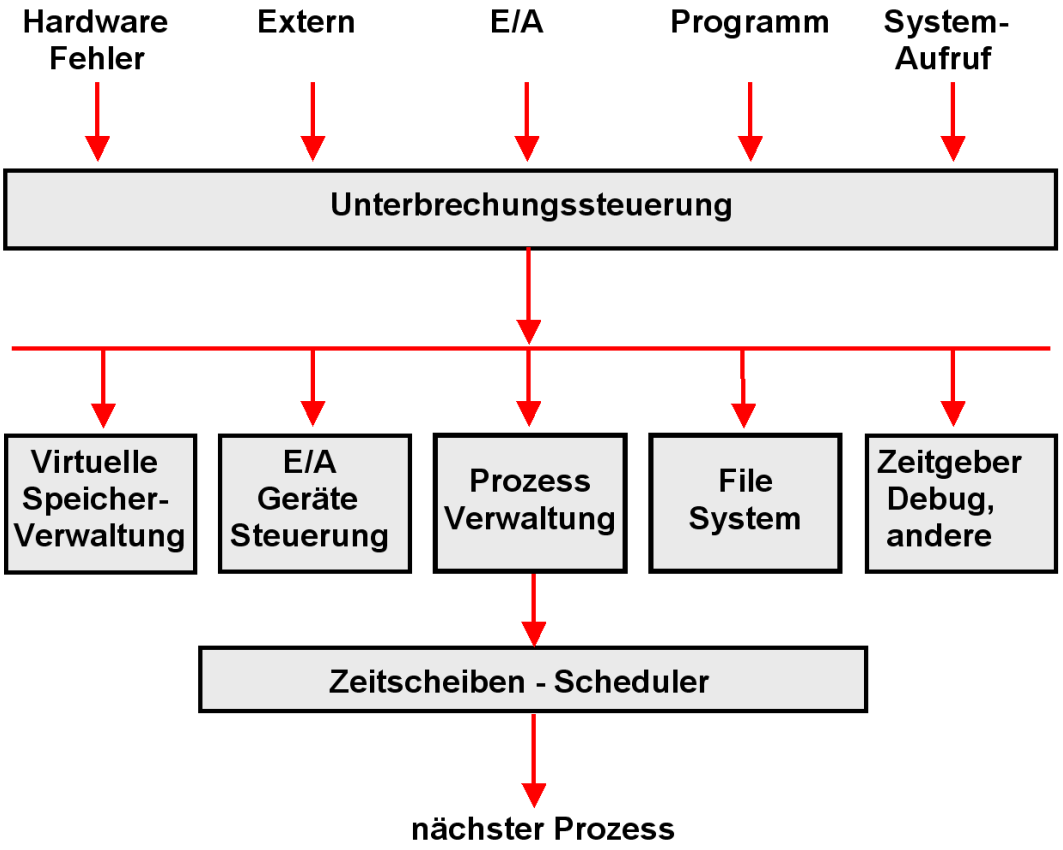


Abb.17.1.3
Struktur des Überwachers

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.

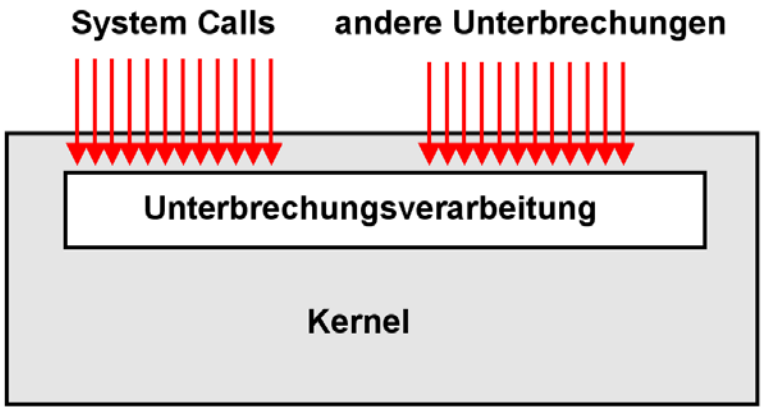


Abb.17.1.4
System Calls werden vom Überwacher wie Unterbrechungen behandelt

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.

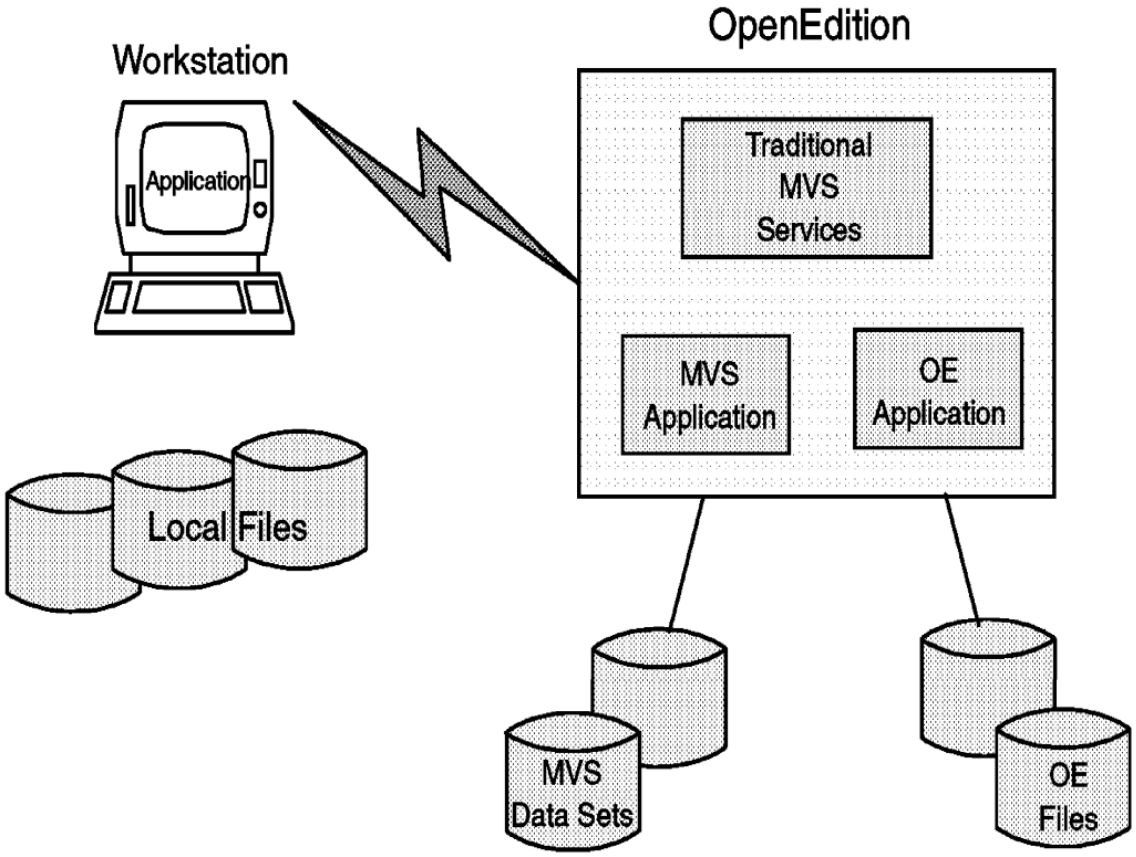


Abb.17.1.5
Normale z/OS Programme können auch auf Unix System Service Files zugreifen

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

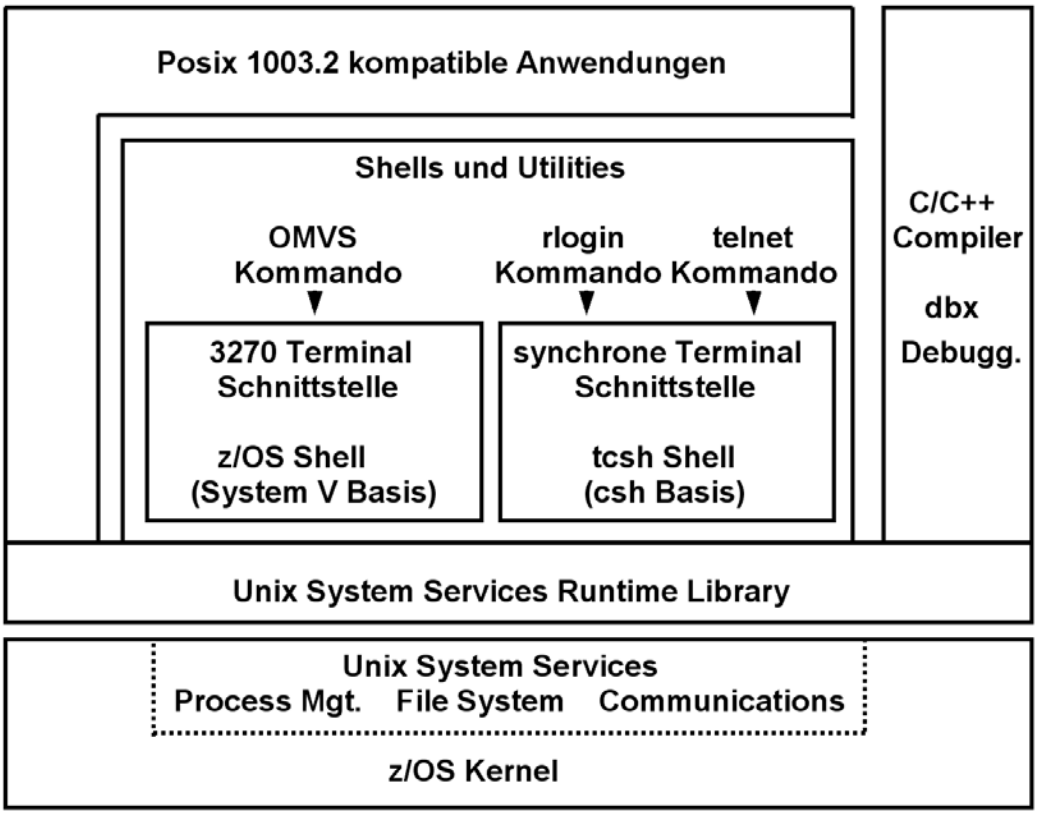


Abb.17.1.6
Komponenten der Unix System Services Erweiterung

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

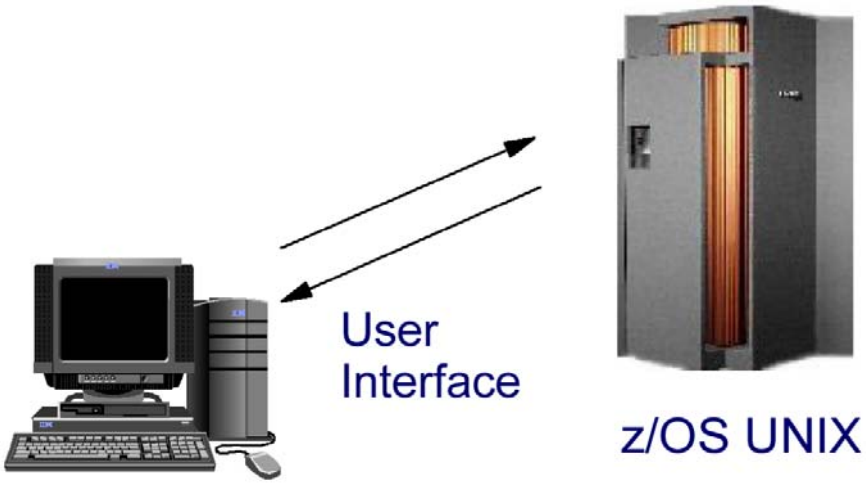


Abb.17.1.7
Zwei unterschiedliche Unix System Service Shells

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

```
  Menu  List  Mode  Functions  Utilities  Help
--------------------------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> omvs  <---


Place cursor on choice and press enter to Retrieve command

=> ish
=> omvs
=>
=>
=>
=>
=>
=>
=>
=>

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

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.

```
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

SPRUTH : /u/spruth >ls -als
total 96
  16 drwxr-xr-x   4 BPXOINIT SYS1        8192 Oct  3 23:40 .
  16 drwxrwxrwx  120 BPXOINIT SYS1        8192 Sep 30 17:32 ..
   8 -rwx------   1 BPXOINIT SYS1         365 Mar 25  2002 .profile
   8 -rw-------   1 BPXOINIT SYS1        2347 Oct  3 23:42 .sh_history
   8 -rw-r-----   1 BPXOINIT SYS1        3715 Jul  7  2001 index.htm
   8 -rw-r-----   1 BPXOINIT SYS1        2806 Jul  7  2001 links01.htm
  16 drwxr-x---   2 BPXOINIT SYS1        8192 Mar 28  2002 sm390
  16 drwxr-xr-x   6 BPXOINIT SYS1        8192 Apr 12 20:06 was_samples
SPRUTH : /u/spruth >
 ===>
                                                                    INPUT
ESC=¢   1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

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

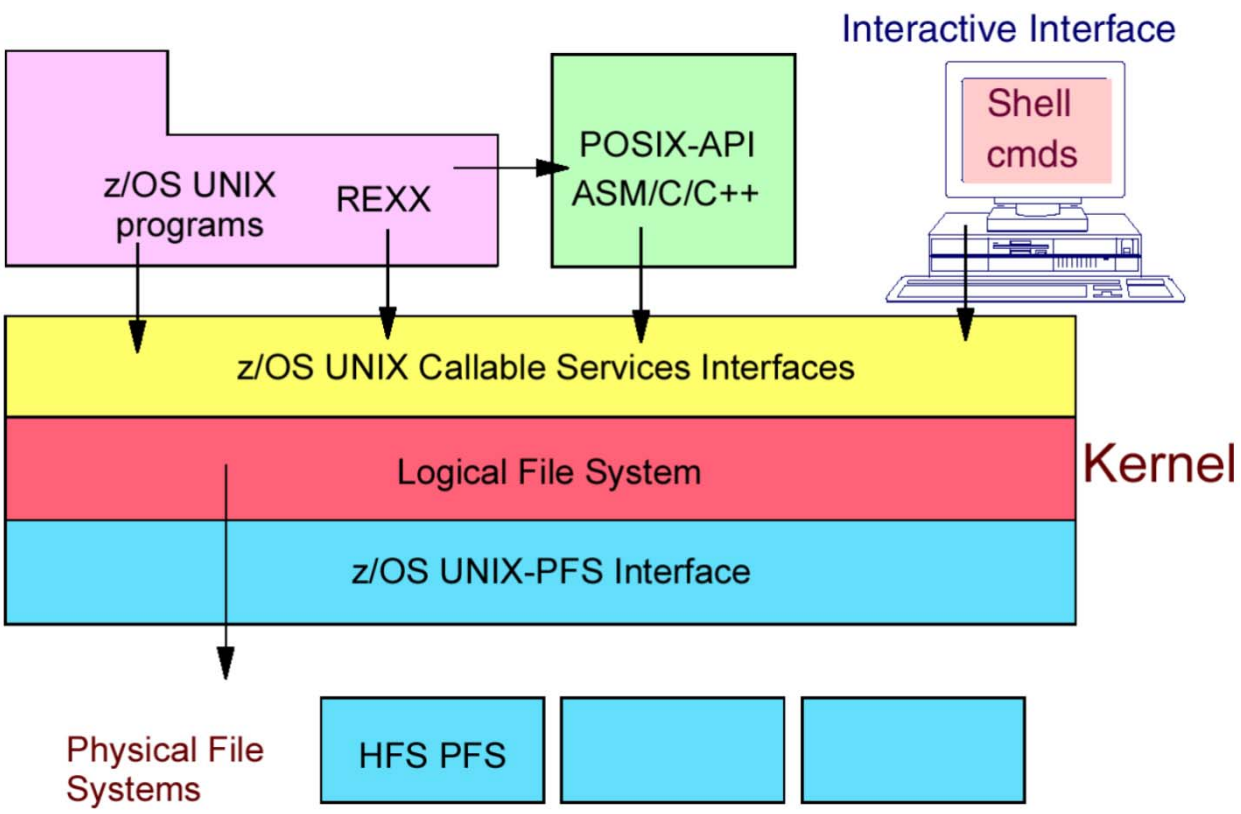


Abb.17.1.10
Hinter dem logischen File System verbergen sich mehrere physische file Systeme

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.

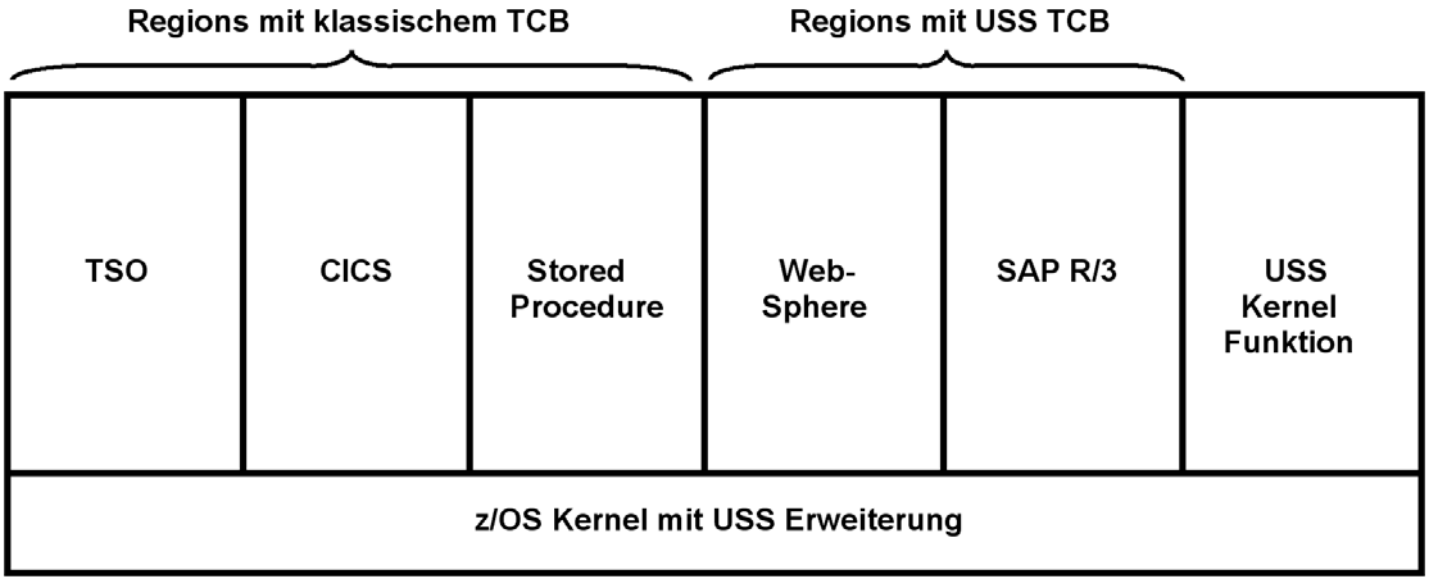


Abb.17.1.11
USS Regions nutzen den USS TCB

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

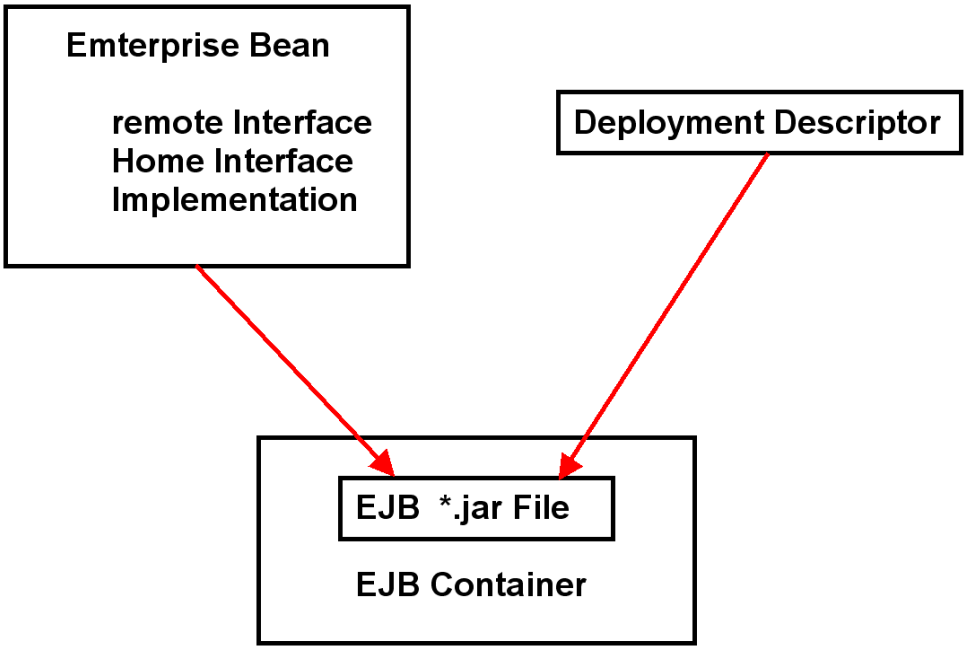


Abb. 17.2.1
Elemente einer *.jar File

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

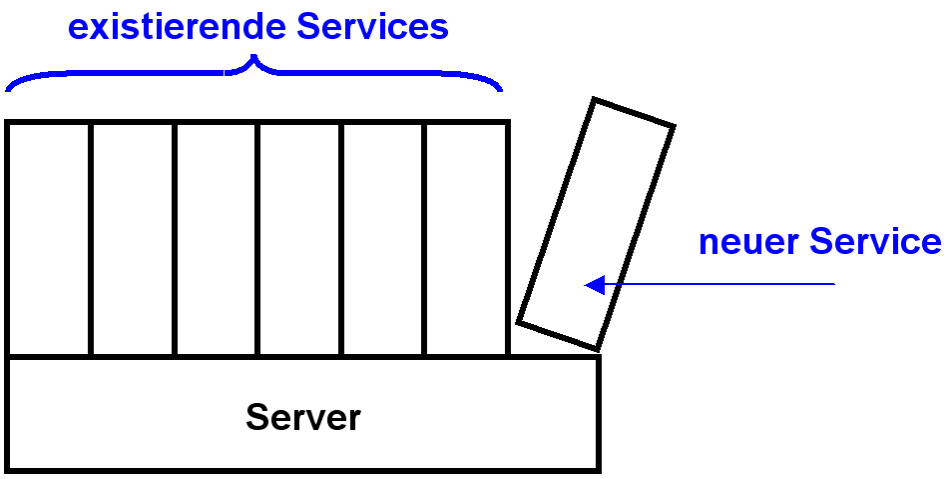


Abb. 17.2.2
Die Installation einer neuen Anwendung braucht mehrere Teilschritte

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.

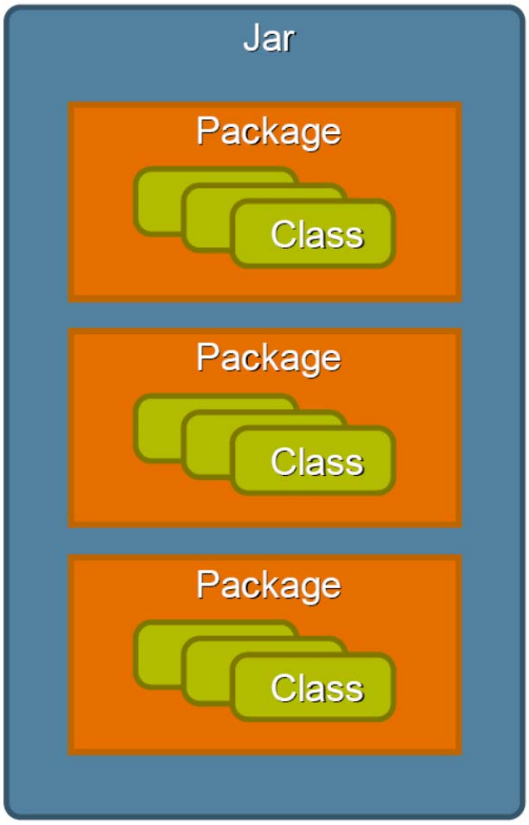


Abb. 17.2.4
Eine Anwendung kann aus mehreren Packages bestehen.

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

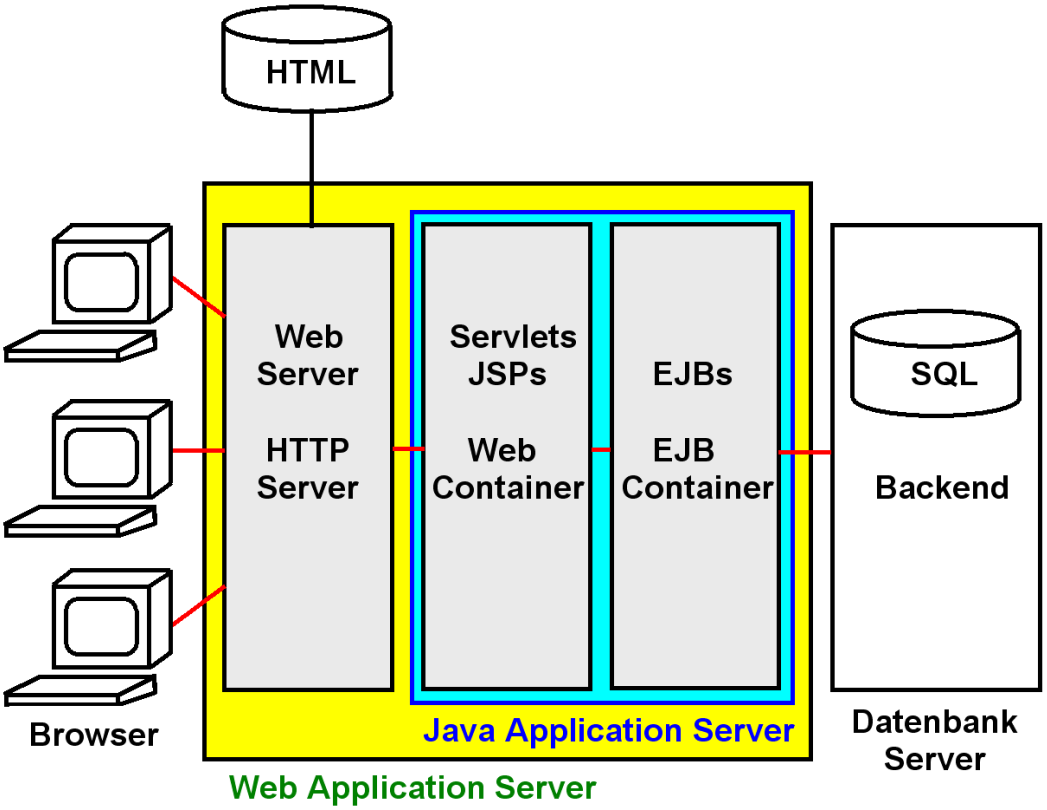


Abb. 17.2.5
Die wichtigsten Komponenten eines Web Application Servers

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.

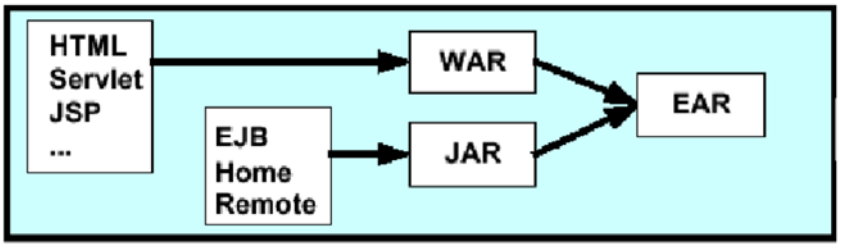


Abb. 17.2.6
Zusammenspiel von WAR, JAR und EAR

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.

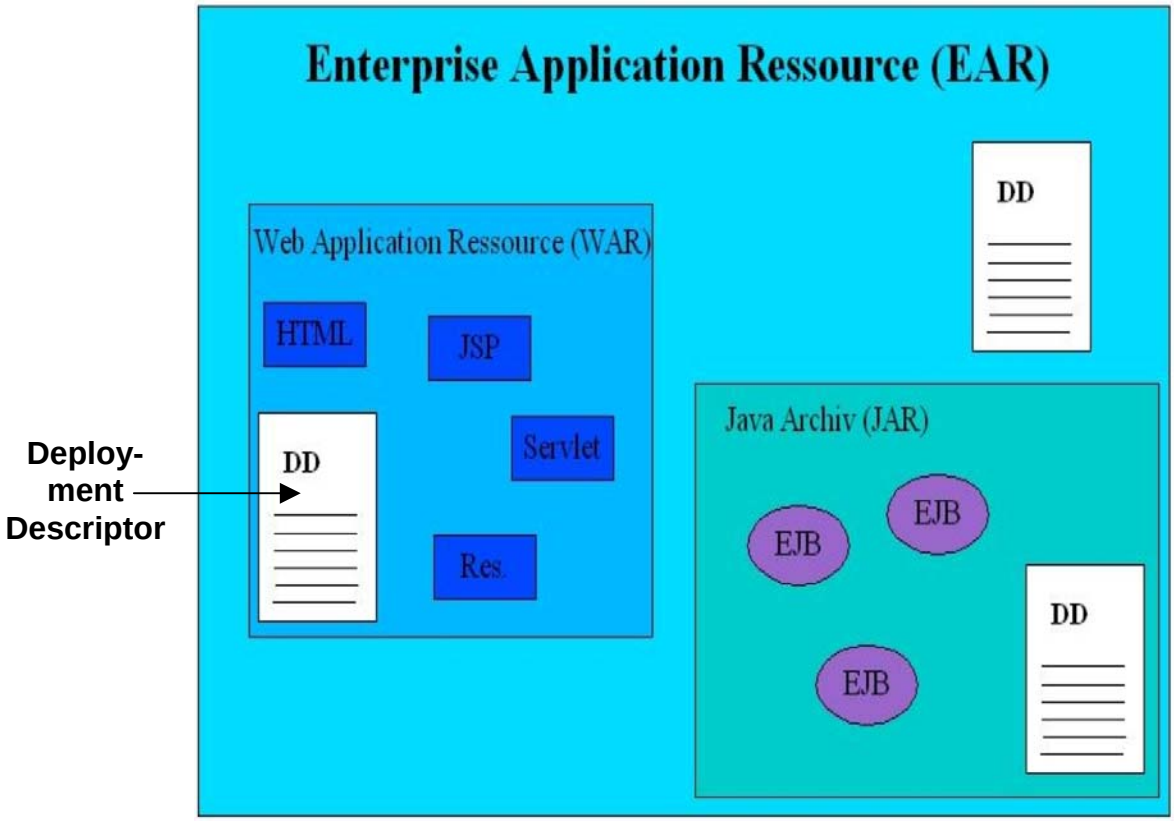


Abb. 17.2.7
Bestandteile der Enteprise Application Resource

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

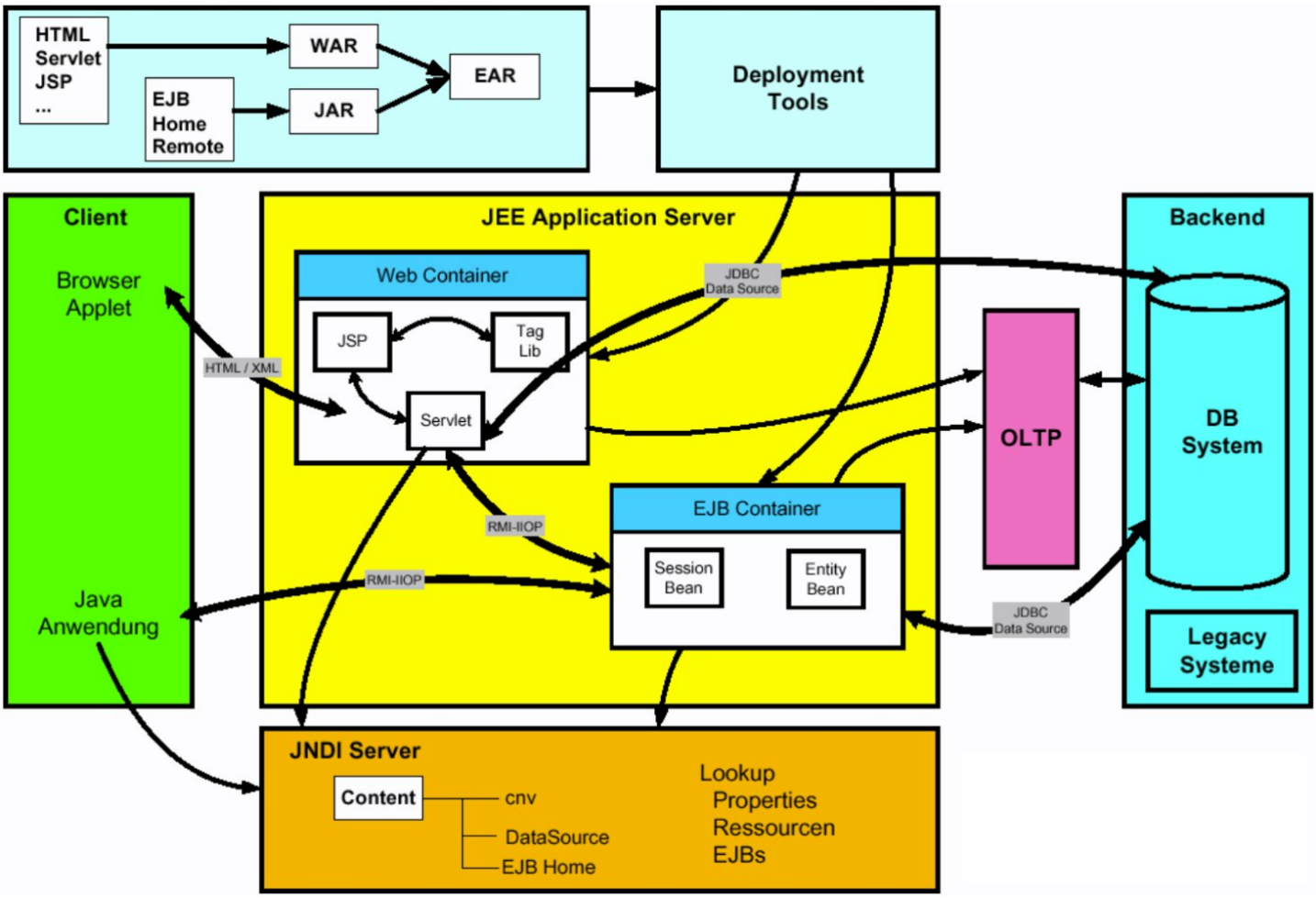


Abb. 17.2.8
Zusammenspiel Web Container, EJB Container und Backend

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).

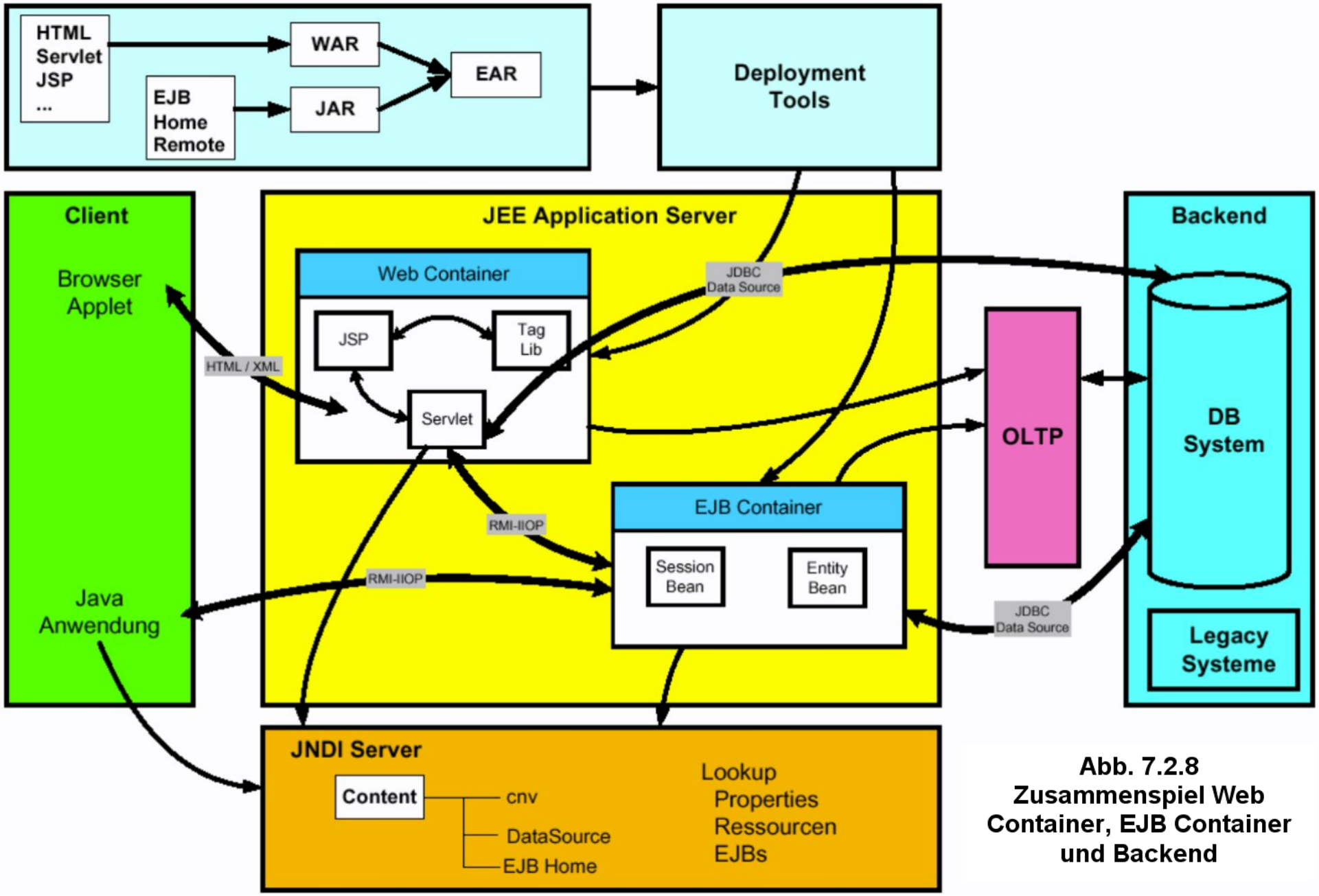


Abb. 7.2.8
Zusammenspiel Web Container, EJB Container und Backend

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.

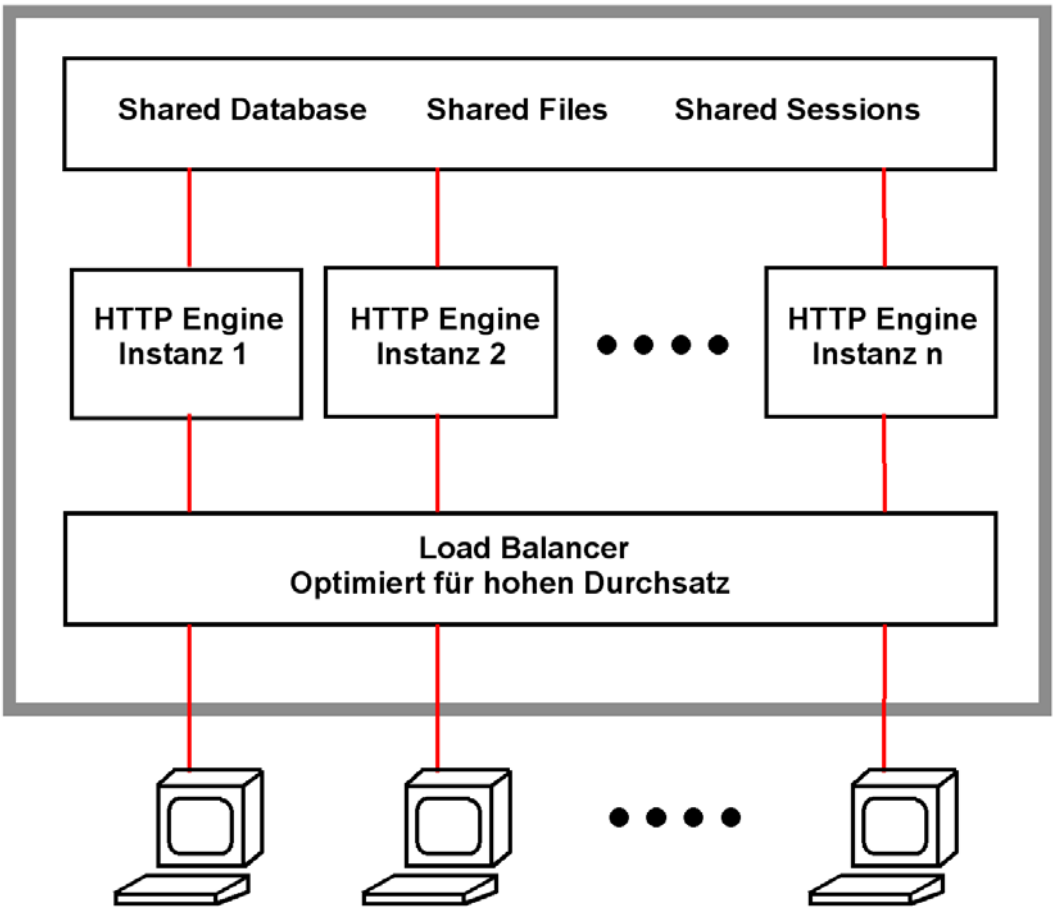


Abb. 17.3.1
Server Farm

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

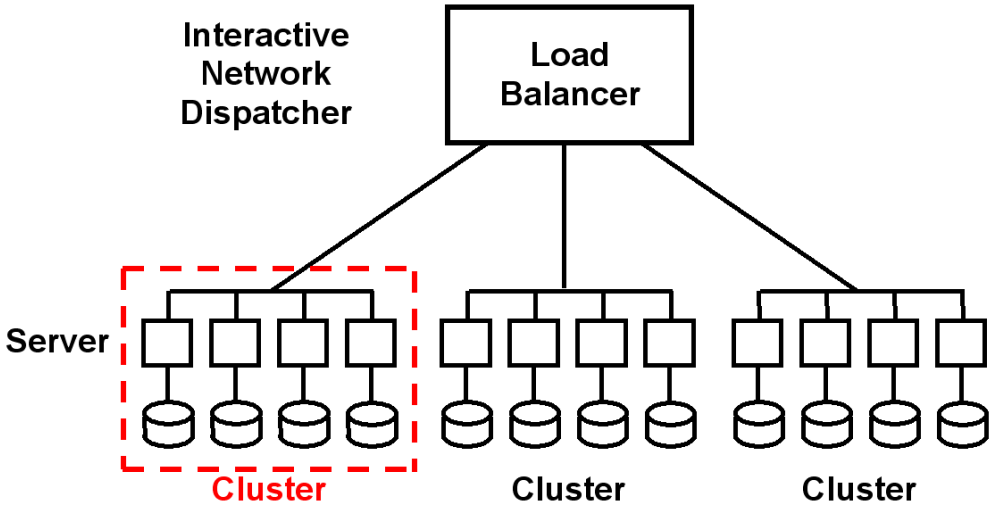


Abb. 17.3.2
Google Cluster

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**

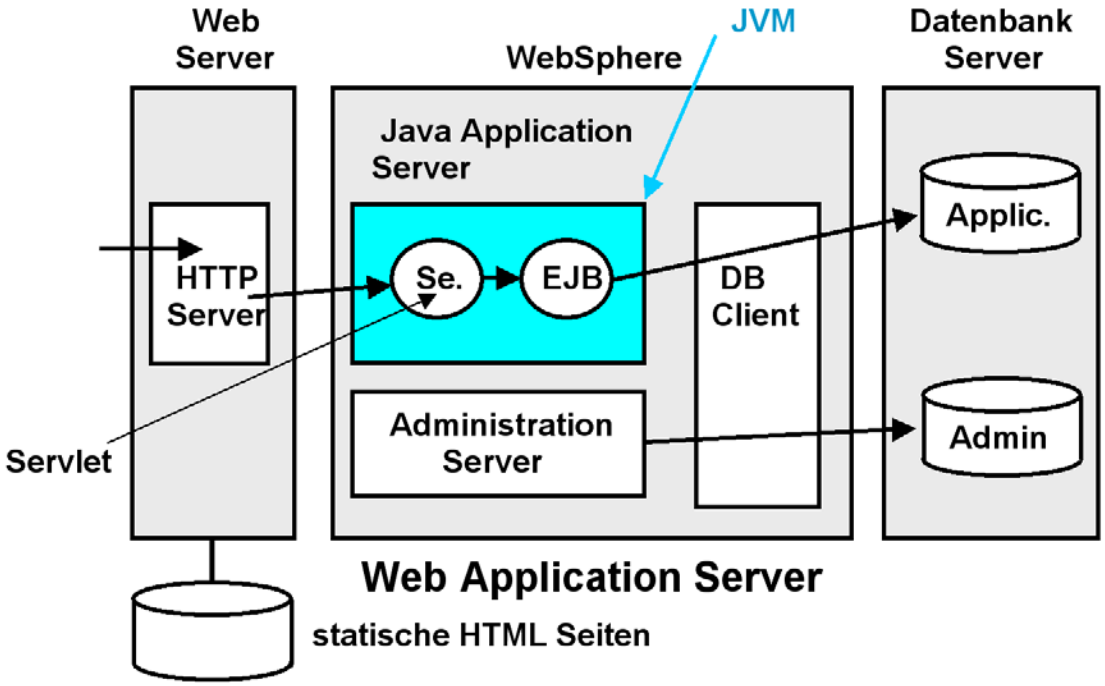


Abb. 17.3.5
WebSphere WAS Grundstruktur

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.

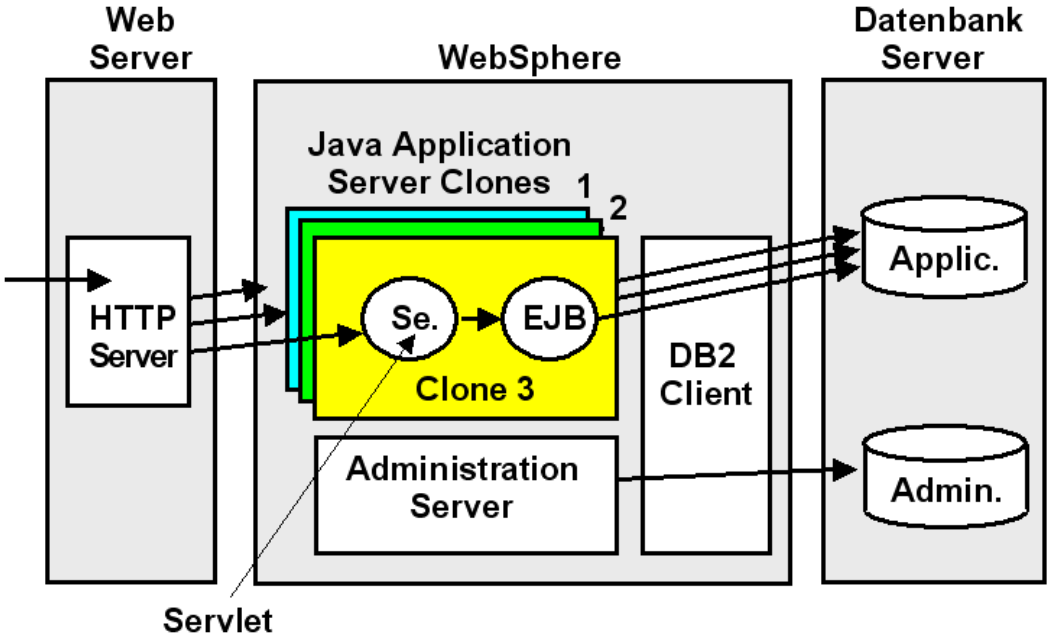


Abb. 17.3.6
Zur besseren Ausnutzung existieren mehrfache Clones der Java Application Server

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.

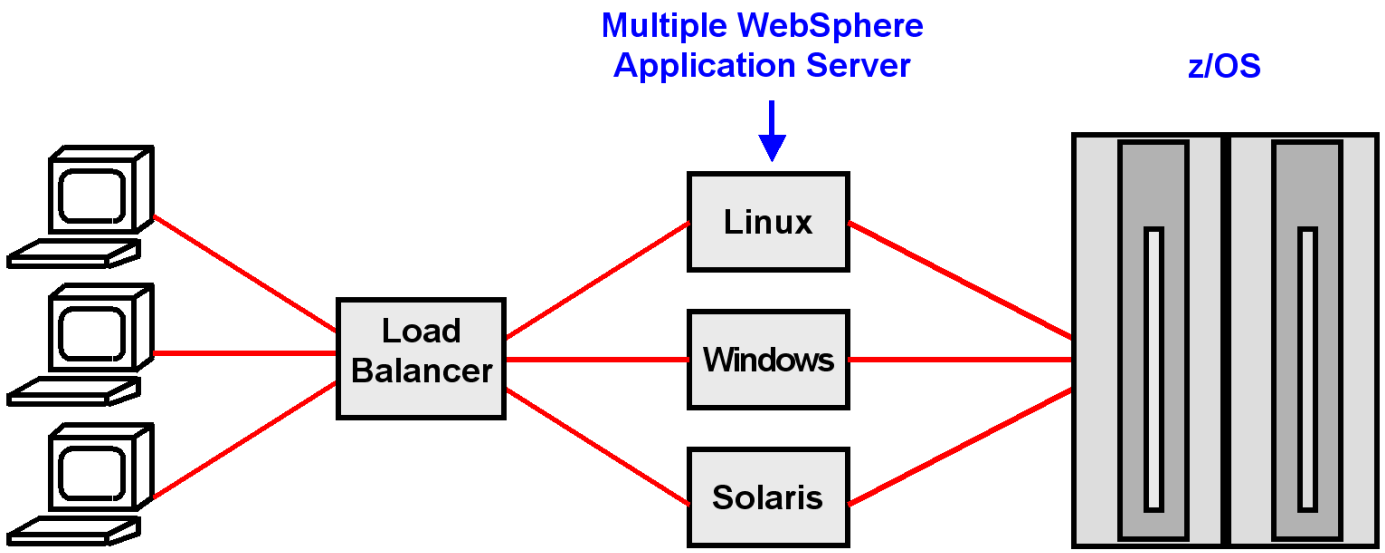


Abb. 17.3.7
Distributed WAS auf unterschiedlichen Platformen

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.

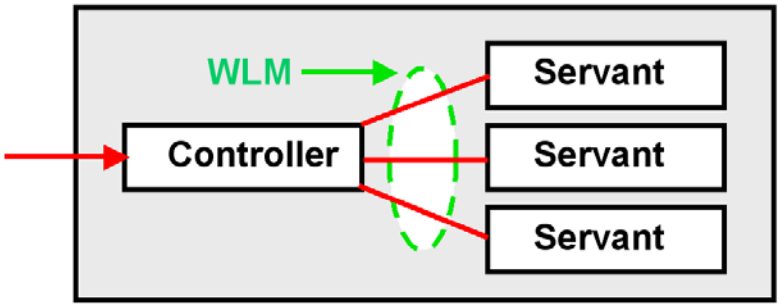


Abb. 17.3.8
z/OS Version des WebSphere Application Server (WAS)

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.

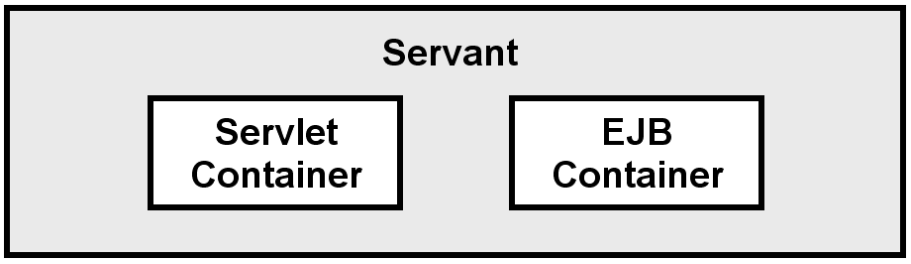


Abb. 17.3.9
WebSphere Runtime Structure on z/OS

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.**

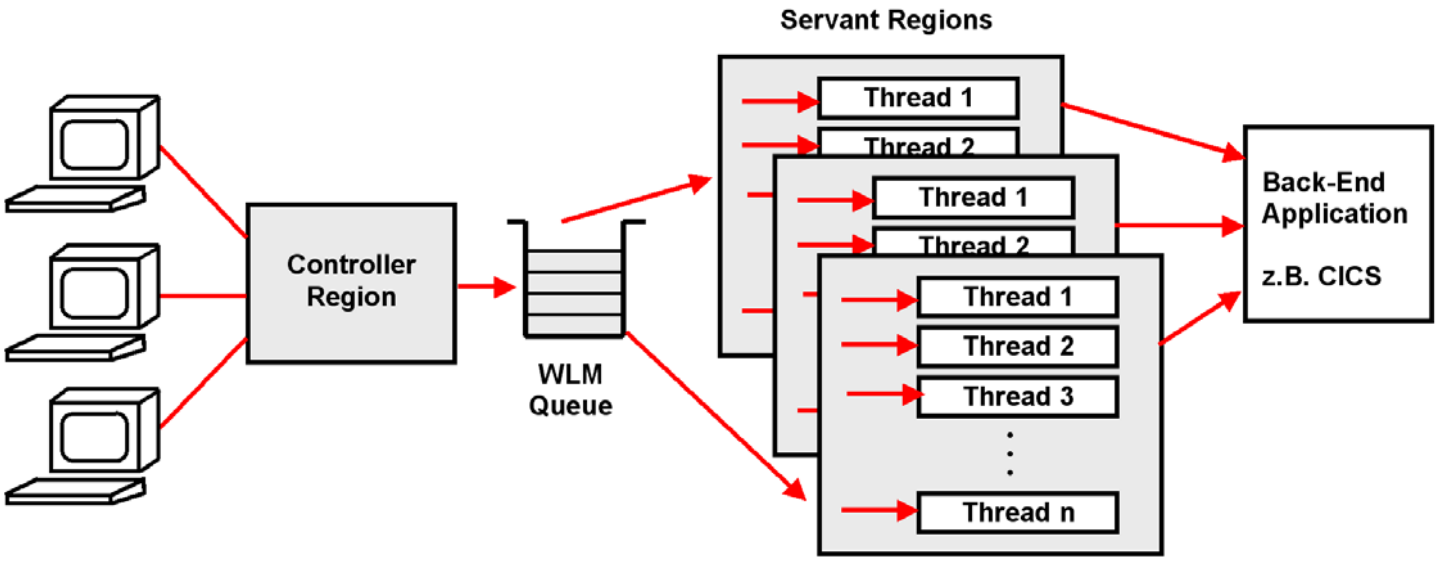


Abb. 17.3.10
z/OS Work Load Manager Steuerung der Input Queue

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

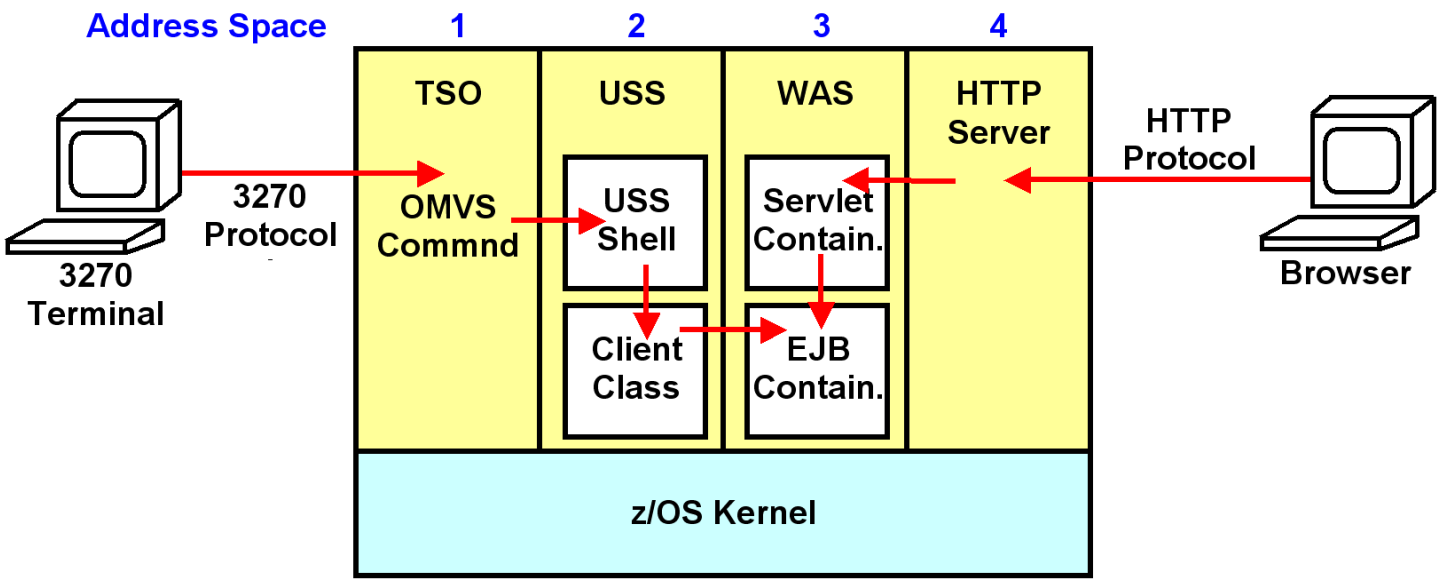


Abb. 17.4.1
Zugriff auf EJBs

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.

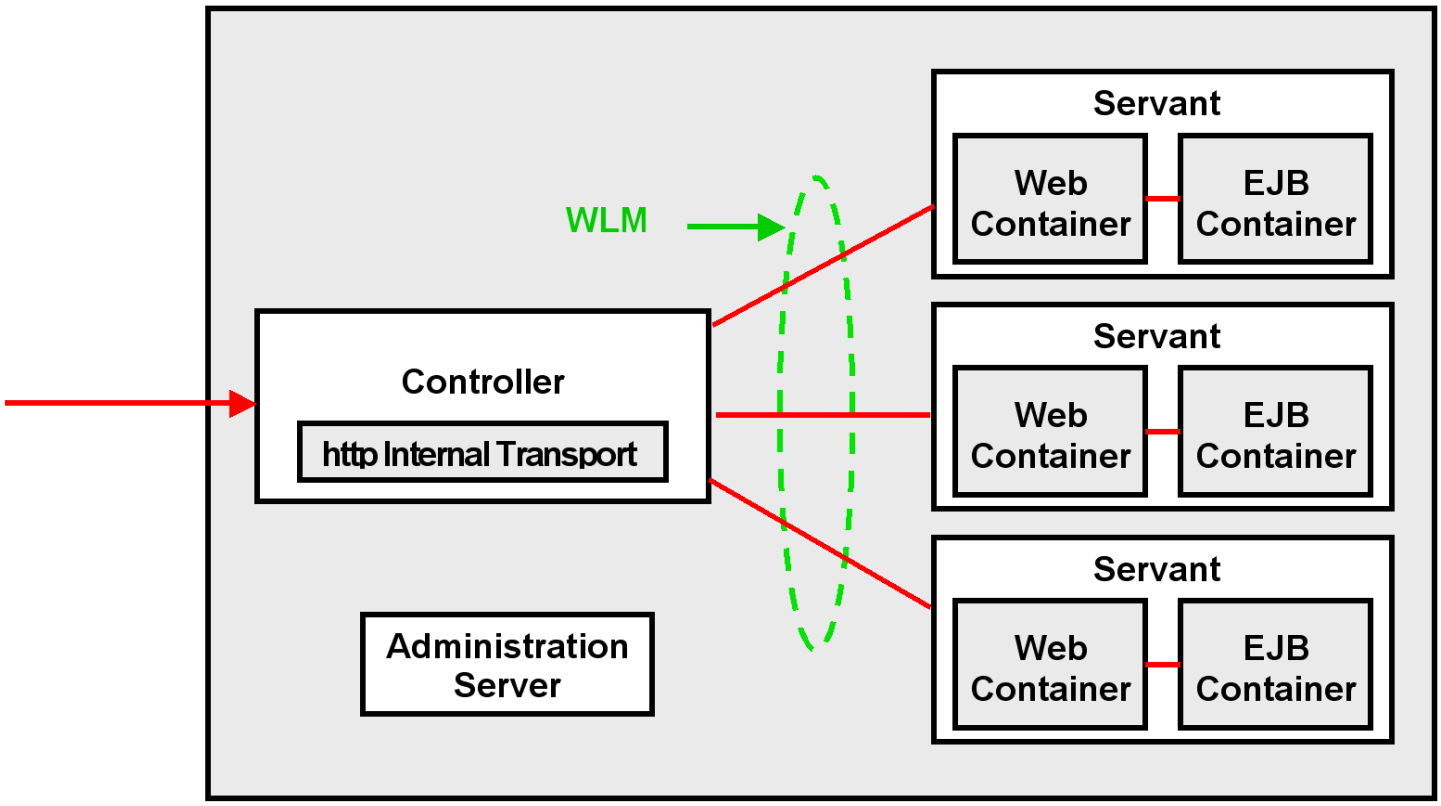


Abb. 17.4.2
WAS Controller mit der http Internal Transport Komponente

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 .

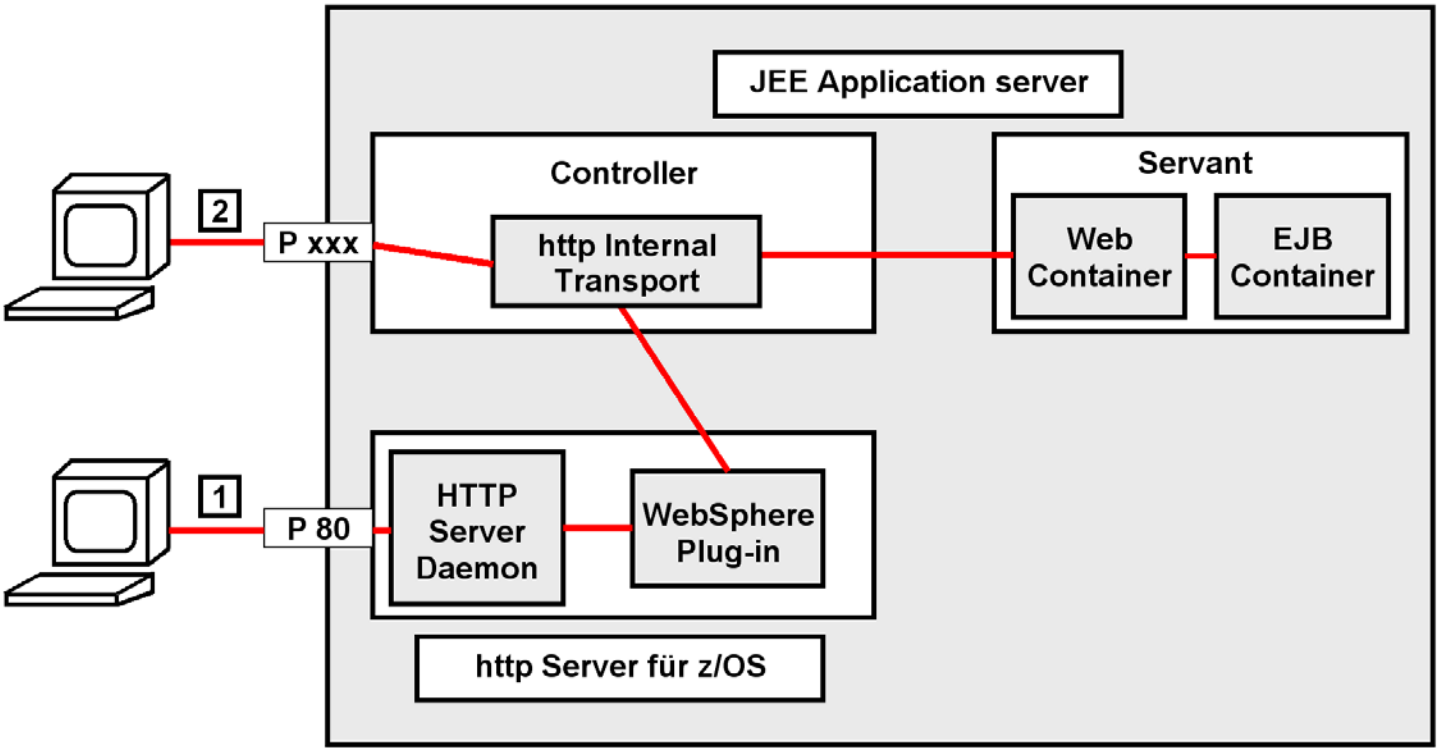


Abb. 17.4.3
http Internal Transport

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

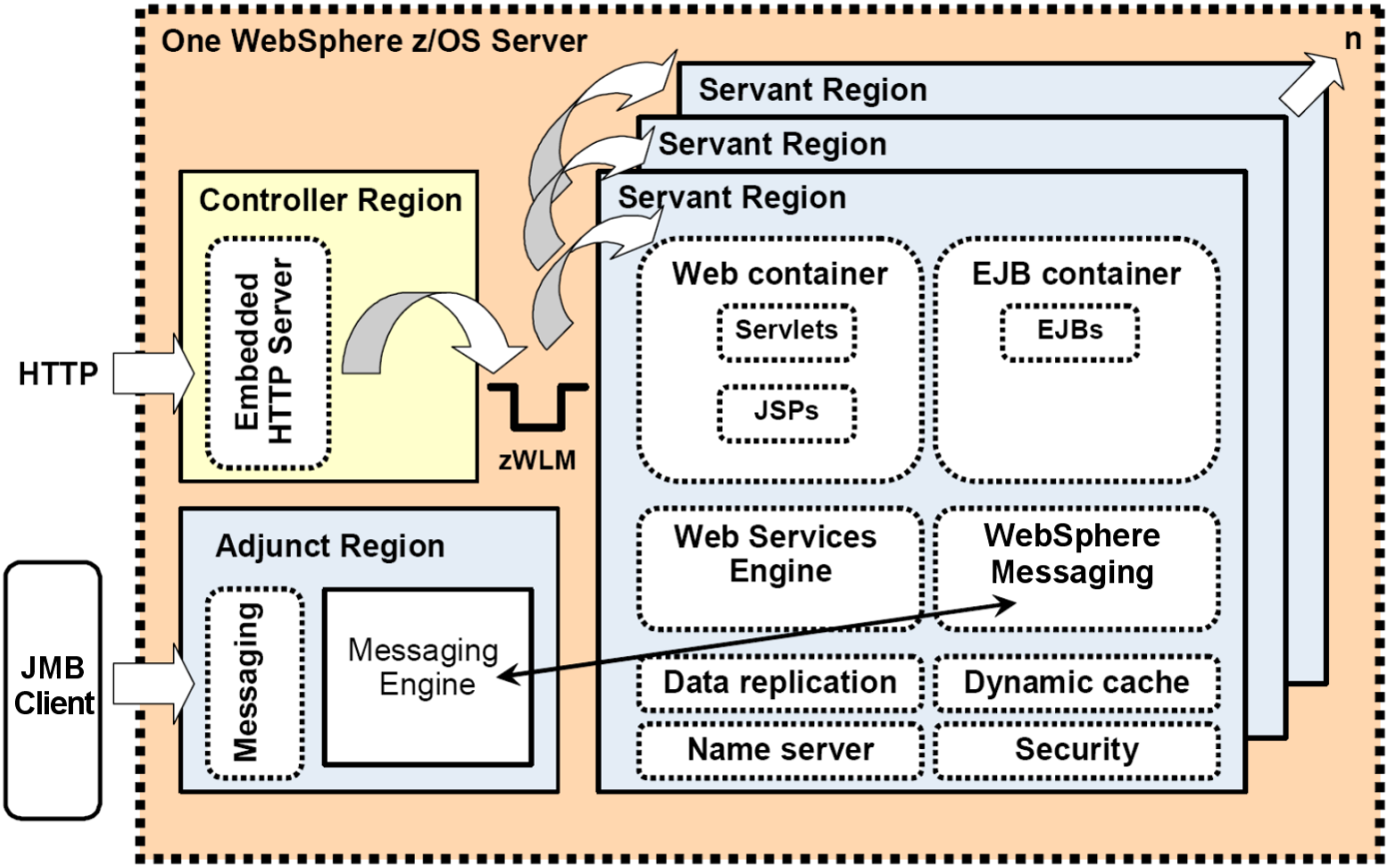


Abb. 17.4.4
Komponenten des WebSphere Application Servers

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.

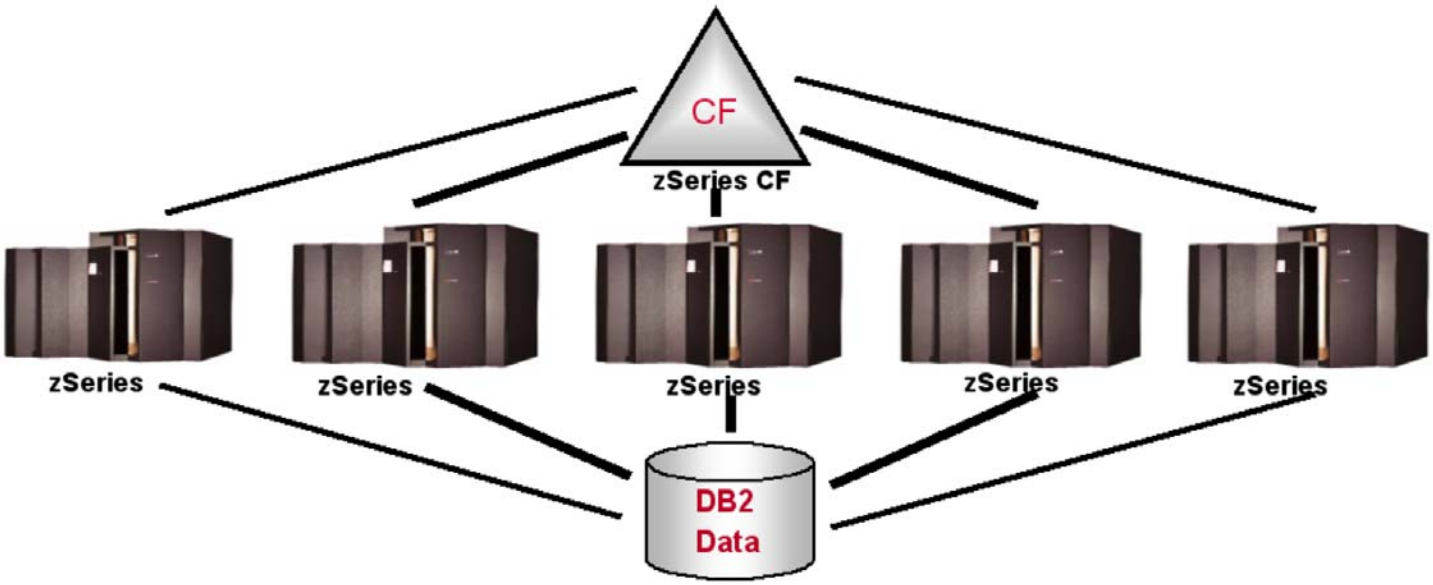


Abb. 17.4.5
Konfiguration für des Skalierbarkeit Test

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

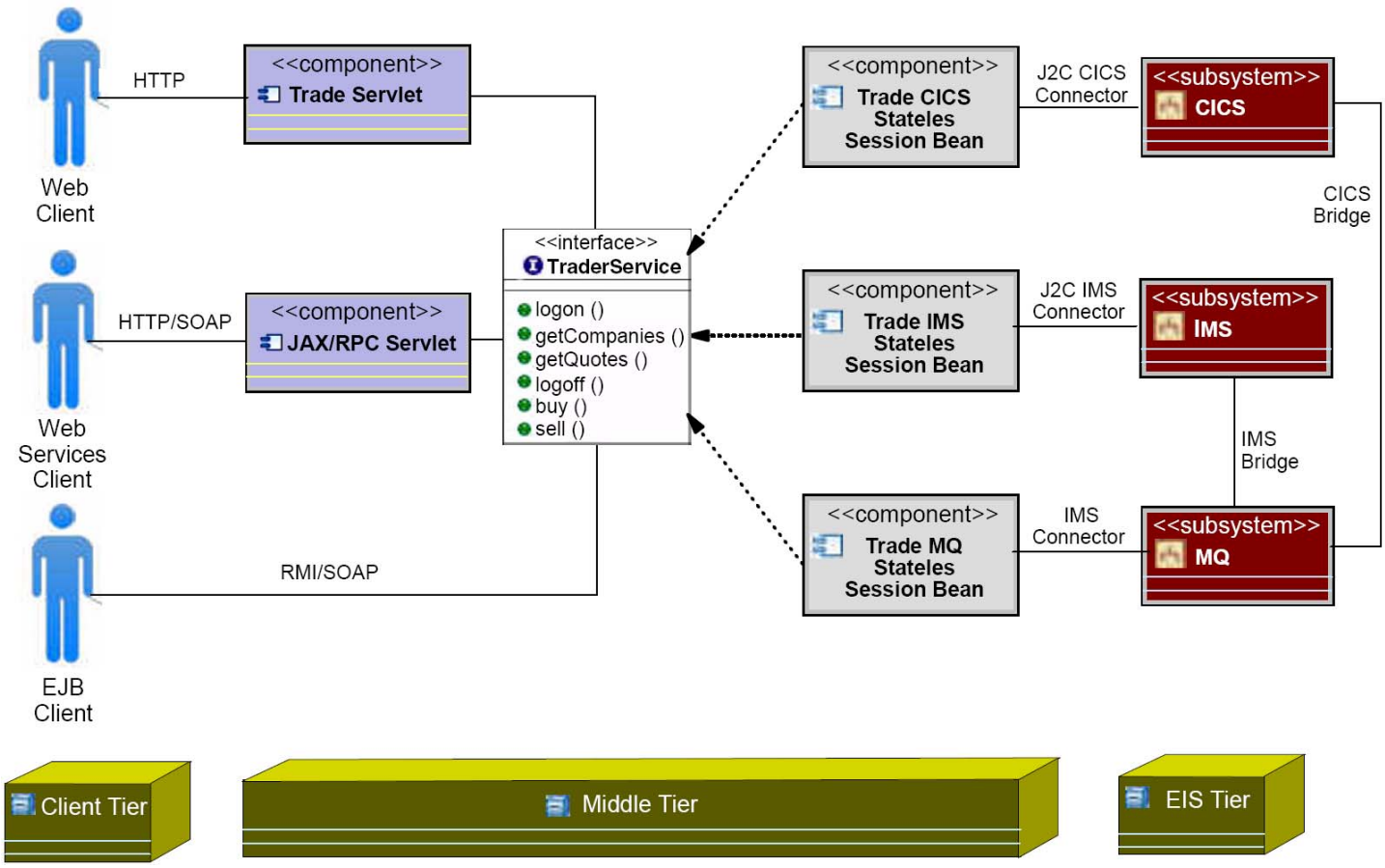


Abb. 17.4.6
Trader Benchmark

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.

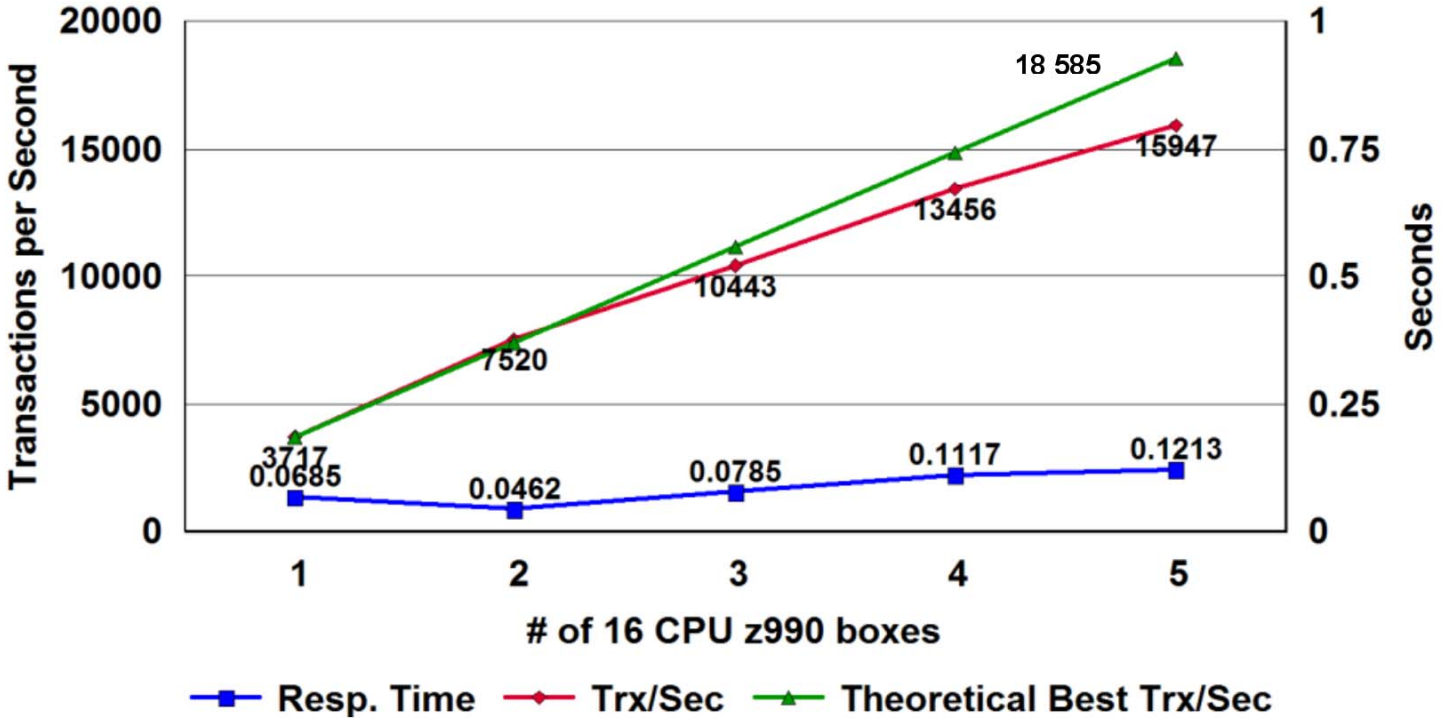


Abb. 17.4.8
Ergebnisse des Skalierbarkeit Tests

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

18. Java Connection Architecture

18.1 SNA Communication over TCP/IP

18.1.1 System Network Architecture

z/OS Networking besteht heute aus zwei getrennten Netzwerk Architekturen: System Network Architecture (SNA) und TCP/IP. SNA entstand in 1974 und hat einige Funktionen, die TCP/IP erst mit Version 6 ebenfalls haben wird. Bis in die 90er Jahre war SNA der de facto Standard in allen größeren Unternehmen. Wegen der Bedeutung des Internets stellten in den letzten 10 Jahren die allermeisten Unternehmen ihre physischen Netze von SNA auf TCPIP um; dieser Prozess ist nahezu abgeschlossen.

IBM unterstützt weiterhin SNA und wird dies auch in Zukunft tun. Viele z/OS Komponenten nutzen nach wie vor SNA. Hierzu werden SNA Nachrichten in TCP/IP Nachrichten gekapselt und vom Empfänger wieder ausgepackt.

Die Kommunikation eines 3270 Emulators mit TSO oder CICS erfolgt über SNA auf diese Art.

Eine weitere Netzwerkarchitektur neben SNA und TCP/IP ist OSI. OSI wurde 1987 auf der CeBIT in Hannover in einem Verbund vorgestellt, in dem 10 ausstellende Unternehmen eine x.400 Intersystem Kommunikation ihrer eigenen OSI Implementierungen vorstellten. x.400 ist das OSI Äquivalent zu dem TCP/IP Simple Mail Transfer Protocol SMTP.

OSI war geplant, SNA und TCP/IP zu ersetzen, was aber nicht passiert ist. Die meisten OSI Produkte sind wieder verschwunden; einige Teilkomponenten existieren noch und haben sich durchgesetzt. Beispiele sind X.400, X.500, x.509 und x.25.

Die OSI Darstellung eines Schichtenmodells hat sich jedoch durchgesetzt und wird auch von TCP/IP benutzt.

18.1.2 OSI - Protokollhierarchie

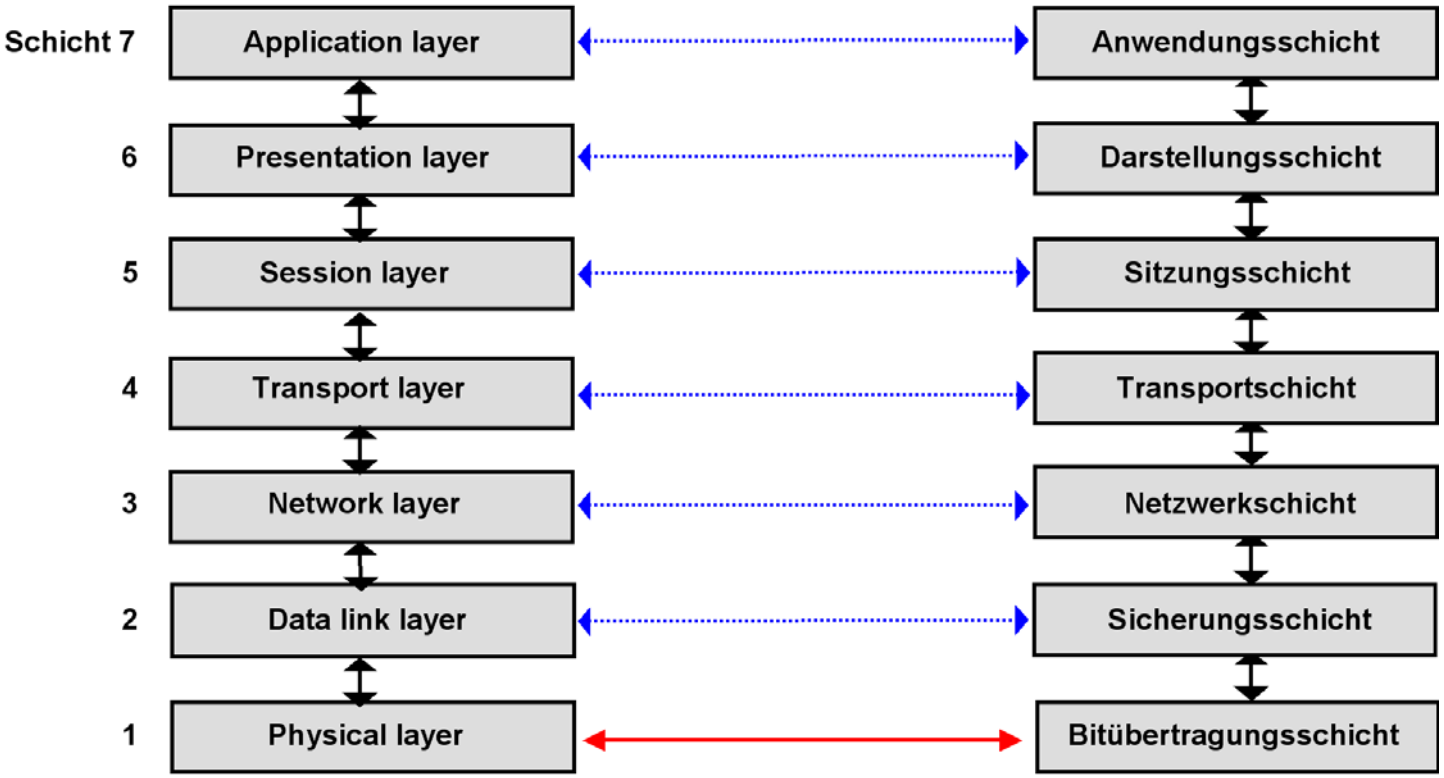


Abb. 18.1.1
Bezeichnung der OSI Schichten

Um die Komplexität einer Netzwerk Implementierung besser verwalten zu können, teilt man die Komponenten in 7 Schichten auf. Jede Schicht glaubt, Nachrichten direkt an die entsprechende Schicht des Partners zu senden. In Wirklichkeit werden Nachrichten an die darunterliegende Schicht weiter gegeben. Nur die beiden Komponenten der untersten Schicht kommunizieren in Wirklichkeit miteinander.

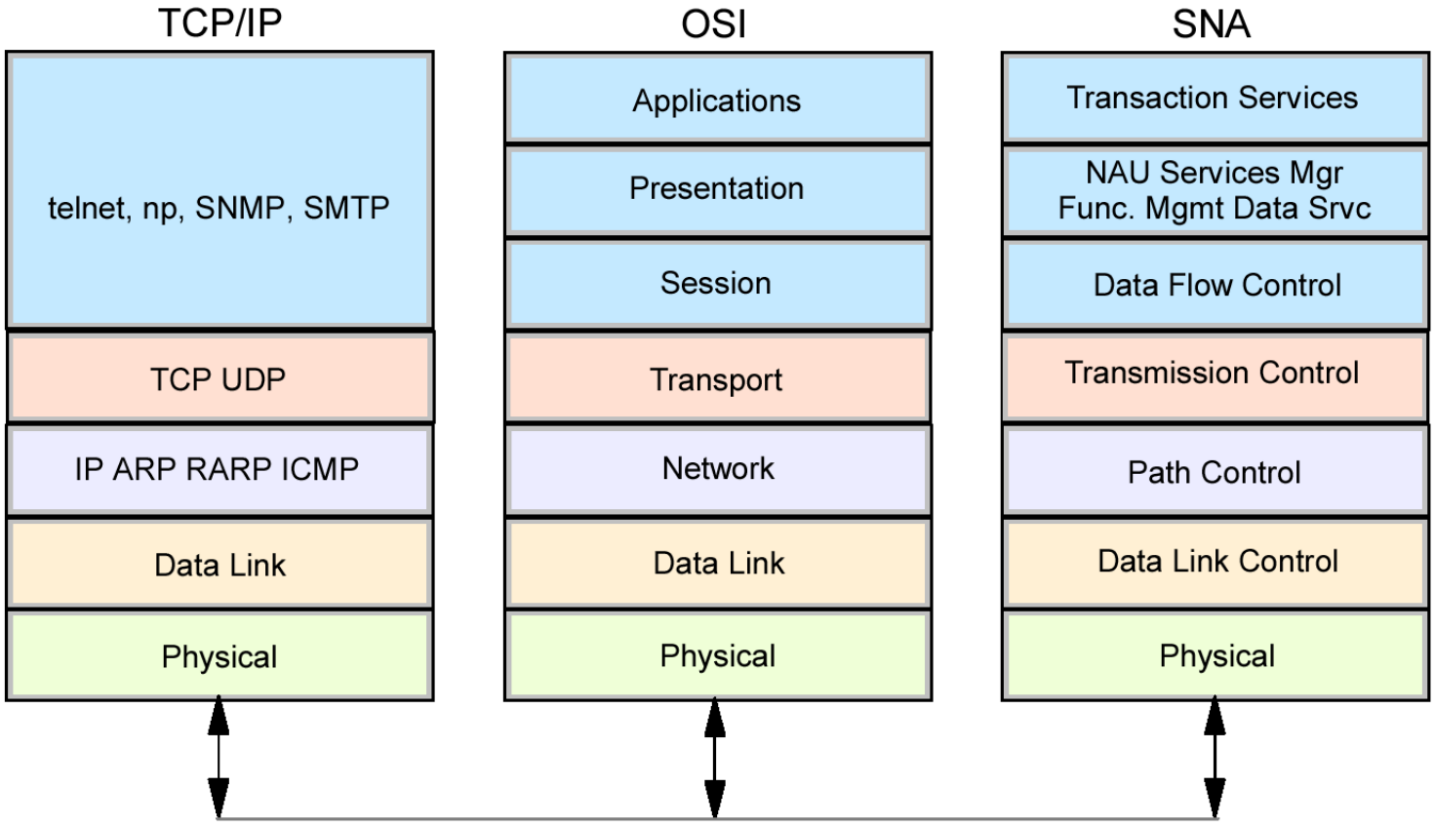


Abb. 18.1.2
SNA und TCP/IP Netzwerk Modell mit 7 Schichten

OSI verlor die Schlacht um Marktanteile an TCP/IP. Mit ganz wenigen Ausnahmen wird kein Geld mehr in weitere OSI Entwicklungen investiert. Das OSI Layer Referenz Modell wurde jedoch fortgeschrieben um auch TCP/IP und SNA zu beschreiben. Abb. 18.1.2 zeigt die Abbildung des OSI Modells auf TCP/IP und SNA.

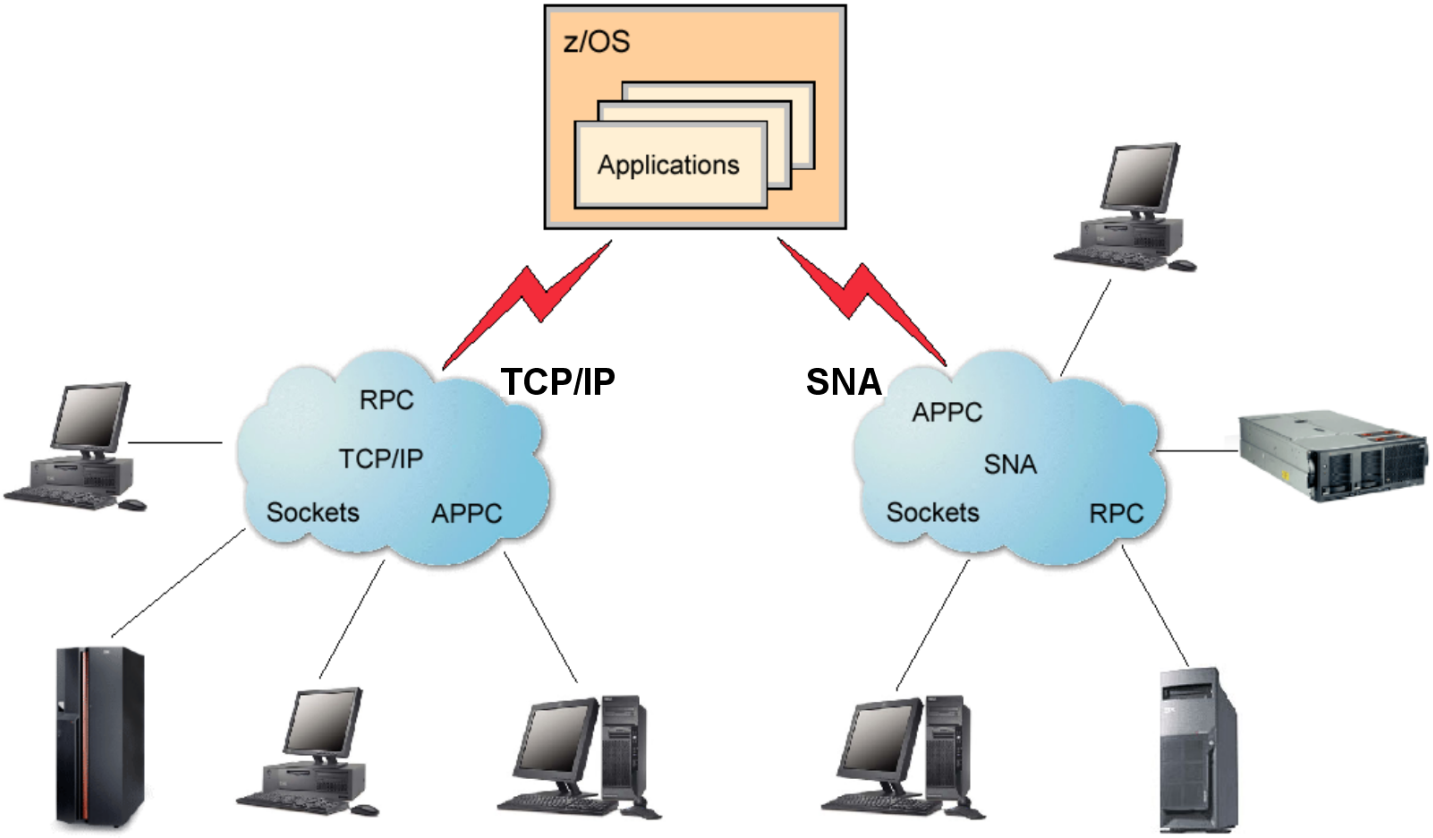


Abb. 18.1.3
Der z/OS Communication Server unterstützt sowohl TCP/IP als auch SNA

Das z/OS „Communication Server Subsystem“ läuft in einem eigenen Address Space und unterstützt neben dem TCP/IP Stack einen vollständigen SNA Stack. Dieser wird heute benutzt, um SNA Nachrichten in TCP/IP Nachrichten zu verpacken und entpacken.

18.1.3 SNA-Vokabular

SNA ist ein Session-orientiertes Protokoll. Eine Session wird zwischen zwei Knotenrechnern aufgebaut (open), Daten werden ausgetauscht, und die Session wird geschlossen (close).

Knotenrechner in einem SNA-Netzwerk werden LU's (Logical Units) genannt. Der ältere Typ LU 2 wurde ursprünglich für nicht-intelligente Terminals entwickelt, und eignet sich nur für die Kommunikation zwischen einem Arbeitsplatzrechner und einem zentralen Server. Es existiert eine RPC ähnliche Funktionalität. Unintelligente 3270 Terminals oder PC´s mit einem 3270 Emulator verwenden LU 2, um mit einer z/OS TSO oder CICS Session zu kommunizieren.

Der derzeitig gängigste Typ wird als LU 6.2 bezeichnet. Er hat volle Peer-to-Peer Funktionalität in beiden Richtungen.

Die Hardware und Firmware (Abschnitt 12.3.1) eines Knotenrechners wird als PU (Physical Unit) bezeichnet. Eine PU steuert die Verbindungsleitungen zu einer anderen PU. Der derzeitig gängige Typ wird als PU 2.1 bezeichnet. Eine PU kann eine oder mehrere LU's unterstützen; die letzteren verwalten vor allem die „Sessions".

(Eine PU 4 ist direkt an einen z/OS Rechner (über einen „Kanal") angeschlossen und hat erweiterte Funktionen. Die auf einem z/OS-Rechner selbst ablaufende Netzsoftware, z.B. „VTAM", wird als PU 5 bezeichnet).

Eine SNA „Session" ist eine zuverlässige virtuelle Verbindung zwischen zwei LU's. Mehrere parallele Sessions zwischen zwei LU's sind möglich

Eine „APPC Conversation" ist eine (von mehreren) Möglichkeiten einer Transaktionen, unter Verwendung des APPC API über eine Session mit einer anderen Transaktion zu kommunizieren. Siehe http://www.cedix.de/VorlesMirror/Band2/APPC.pdf.

CICS „Interprocess Communication (IPC) kann über APPC durchgeführt werden. In der Regel laufen viele Transaktionen der Reihe nach über eine Session.

18.1.4 Transaktionsverarbeitung über ein SNA Netz

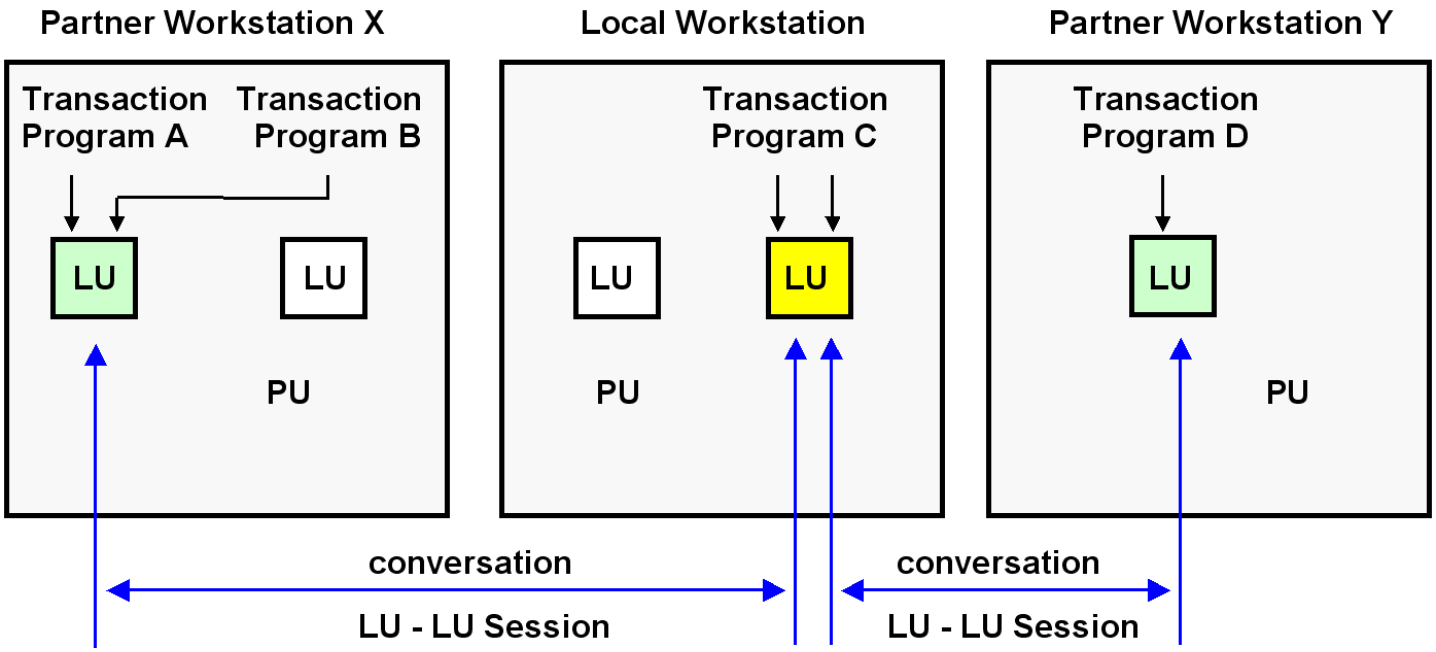


Abb. 18.1.4
LU 6.2 Sessions

Beispiel für eine Transaktionsverarbeitung über ein SNA Netz. Gezeigt ist eine lokale LU, die mit 2 Partner LU s kommuniziert. Der Zugriff zu den LU´s erfolgt über die APPC API.

Das TCP/UDP Protokoll stellt in der Schicht 5 des OSI Modells Sockets zur Verfügung. Mit Hilfe der Socket Schnittstelle können 2 Anwendungen auf 2 Rechnern miteinander kommunizieren.

Das SNA LU 6.2 Protokoll stellt in der Schicht 5 des OSI Modells LUs (Logical Units) vom Typ 6.2 zur Verfügung. Mit Hilfe der APPC-Schnittstelle (Application Program to Program Communication) können 2 Anwendungen auf 2 Rechnern miteinander kommunizieren.

Sockets und LU 6.2 sind in der Schicht 5 des OSI Modells angesiedelt, sehr mächtig, und schwierig zu programmieren. Deswegen werden RPCs benutzt, die unter der Decke Sockets oder LU 6.2 verwenden.

18.1.5 Die Zukunft von SNA

CICS, IMS, and DB2 Anwendungen benutzen auch heute noch unter der Decke SNA, auch wenn relativ erfolgreich versucht wird, dies vor dem Anwender zu verbergen. Dies geschieht mit Hilfe der Enterprise Extender (EE) Komponente.

Enterprise Extender (EE) ermöglicht es Ihnen, ein IP-Netzwerk für den Transport von SNA-Verkehr zu nutzen, einschließlich SNA-Verkehr zwischen verschiedenen Unternehmen.

Siehe http://www.cedix.de/VorlesMirror/Band2/SNA02.pdf

Die existierenden z/OS Installationen haben erhebliche Investitionen in 3270 und SNA Anwendungen getätigt. IBM wird VTAM als Teil des z/OS Communication Servers auch weiterhin zu unterstützen und verbessern, und mit neuen Technologien integrieren. Es existieren keine Pläne, die SNA Unterstützung in dem z/OS Communication Server einzustellen.

Und hier ein Tip für Sie:

Wenn Sie sich um einen Job bewerben, und beim Vorstellungsgespräch mit Vokabeln wie SNA oder LU 6.2 um sich werfen, ist Ihr Interviewer möglicherweise sehr beeindruckt.

Die Wahrscheinlichkeit ist sehr groß, dass der Sie interviewende Abteilungs- oder Hauptabteilungsleiter über SNA auch nicht mehr weiß als das, was auf diesen Folien draufsteht.

Auch in einer großen z/OS Installation gibt es heute in der Regel nur einen (oder wenige) hochbezahlten SNA Spezialisten.

18.1.6 Ursprüngliche 3270 Konfiguration

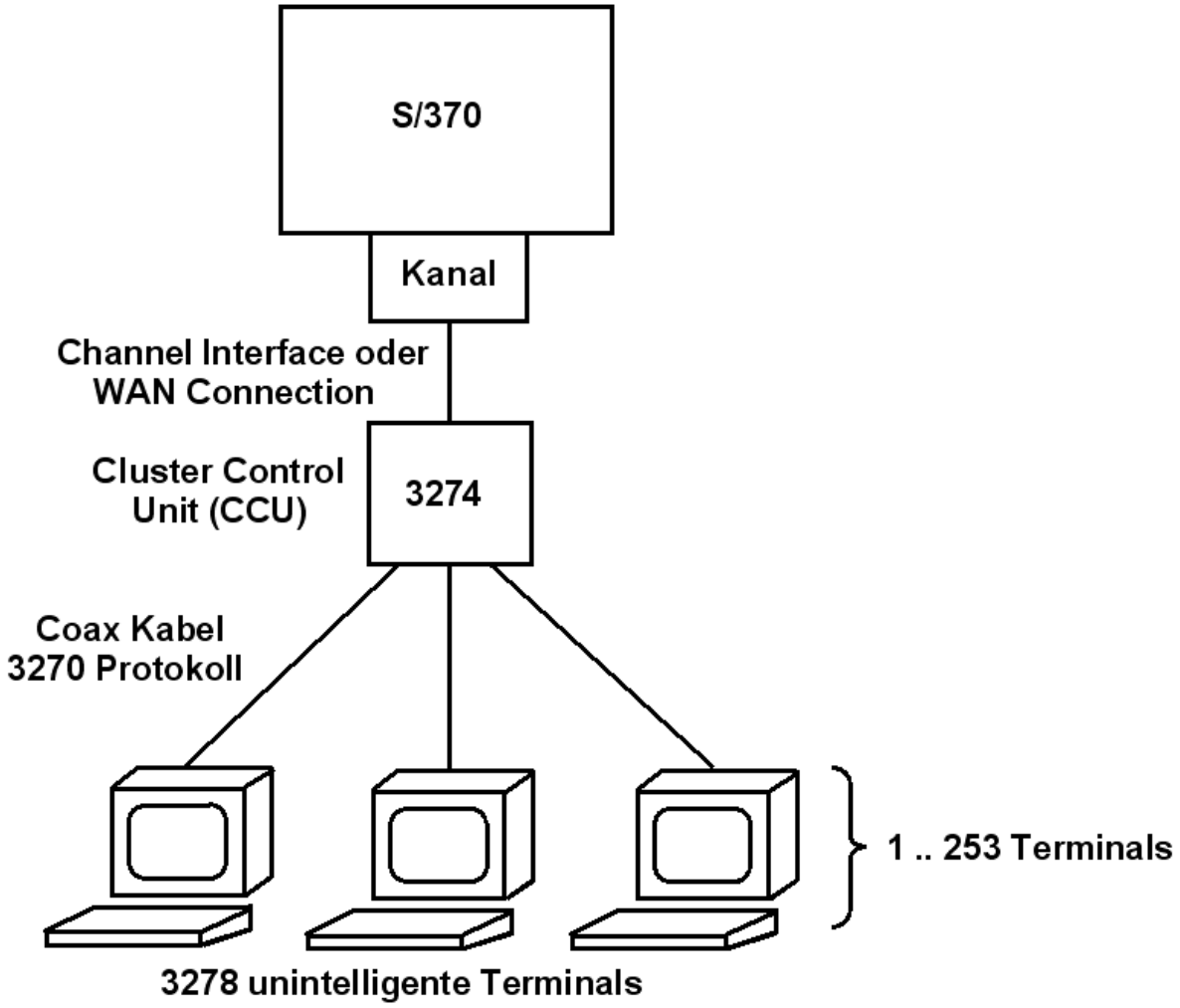


Abb. 18.1.5
Anschluss der 3278 Terminals

Dargestellt ist die ursprüngliche 3270 Konfiguration, wie sie 1972 verfügbar wurde. Ein Terminal bestand aus einem Bildschirm und einer Tastatur, und war mit einem Coax-Kabel mit einer Modell 3274 Steuereinheit verbunden.

Jeder 3278 Terminal war über eine point-to-point Coaxial Kabel Verbindung mit einer IBM 3274 Control Unit verbunden. Die Übertragungsgeschwindigkeit war 2.3 Mbit/s. Ähnlich wie Plattenspeicher werden auch Kommunikationsleitungen über eine Control Unit mit dem Mainframe verbunden

Ein Terminal hatte etwas Logik, aber keinen Processor. Jeder Terminal hatte einen Screen Buffer (memory), der die wiederzugebenden Zeichen speicherte.

Die 3274 Control Unit implementierte eine PU 2 und eine LU 2.1 . Die PU 2 war mit einer VTAM PU 5 verbunden, und die LU 2.1 mit einer TSO oder CICS LU verbunden.

Als IBM 3270 Terminal (siehe Abschnitt 9.1.4) wird eine Klasse von Bildschirmgeräten bezeichnet, die für eine Kommunikation mit einem Mainframe unter Benutzung des 3270 Protokolls entwickelt wurden. Text wurde auf dem ursprünglichen 3278 Terminal augenschonend in grüner Farbe auf einem schwarzen Hintergrund dargestellt; hierher rührt die Bezeichnung „Green Screen“. Anders als serielle ASCII Terminals minimiert der 3270 Terminal die Anzahl der Unterbrechungen auf dem Mainframe.

Später erschienen 3279 Terminals, welche farbige Buchstaben auf einem schwarzen Hintergrund darstellen konnten.

Physische 3270 Terminals werden heute nicht mehr hergestellt; sie wurden fast überall durch 3270 Emulatoren ersetzt, die nach wie vor von zahlreichen Herstellern angeboten werden.

Ein nicht unbeträchtlicher Teil von Benutzern bevorzugt heute nach wie vor einen 3270 Terminal an Stelle eines Bildschirmgerätes mit einer modernen grafischen Oberfläche. Der Grund ist eine höhere Produktivität, vorausgesetzt, man hat sich an die 3270 Darstellung gewöhnt.

Die ursprüngliche Coachs Kabel Verbindung war die erstmalige Implementierung eines Local Area Netzwerkes (LAN). Sie wurde später durch Tosen Ring LAN, und dann durch Ethernet LAN Verbindungen ersetzt.

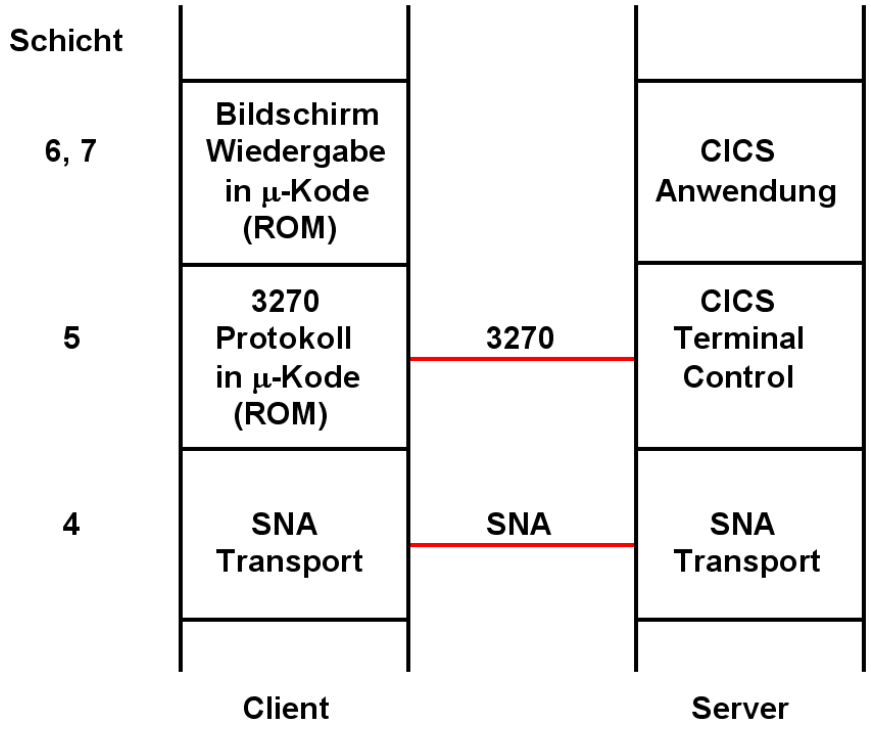


Abb. 18.1.6
Schichtenmodell der 3270 Architektur

Der ursprüngliche 3278 Terminal wurde über SNA mit dem Mainframe-Rechner verbunden.

Das 3270 Protokoll verwendete in der darunterliegenden Schicht eine SNA Session. Die Kombination von 3278 Terminal und 3274 Cluster Control Unit implementierte eine SNA LU 2 und eine PU 2 , wobei auf Grund der geringen logischen Fähigkeiten des 3278 Terminals der Großteil der Arbeit auf die 3274 entfiel.

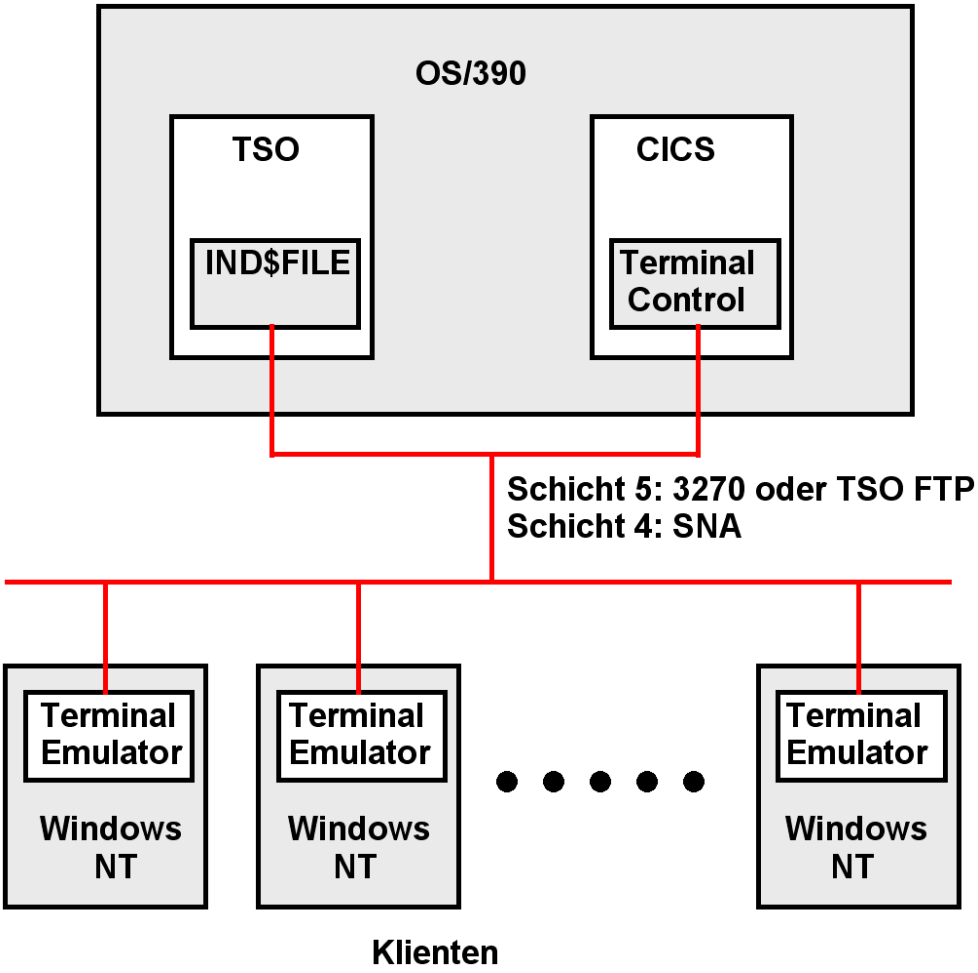


Abb. 18.1.7
Client/Server SNA Kommunikation mit 3270 Terminal Emulatoren

Mit der Verfügbarkeit kostengünstiger Windows (und OS/2) PCs wurden diese an Stelle der ursprünglichen 3278 Terminals und deren Nachfolge-Modellen eingesetzt.

Der Anschluss erfolgt typischerweise über ein LAN, zunächst Token Ring und später Ethernet. Das LAN-Netzwerk selbst benutzte SNA (die Umstellung auf TCP/IP erfolge erst viel später). Jeder PC verfügt über einen Terminal Emulator, welcher eine LU2 und PU2 implementierte. Auf der Mainframe Seite wurde eine PU 5 von VTAM, und eine LU durch die Anwendung implementiert, z.B. durch die CICS Terminal Control Komponente.

Zusätzlich hierzu verfügte diese Konfiguration über eine File Transfer (FTP Äquivalent) Einrichtung mit Hilfe eines TSO Commands namens „IND$FILE“.

18.1.7 Heutiger Zustand

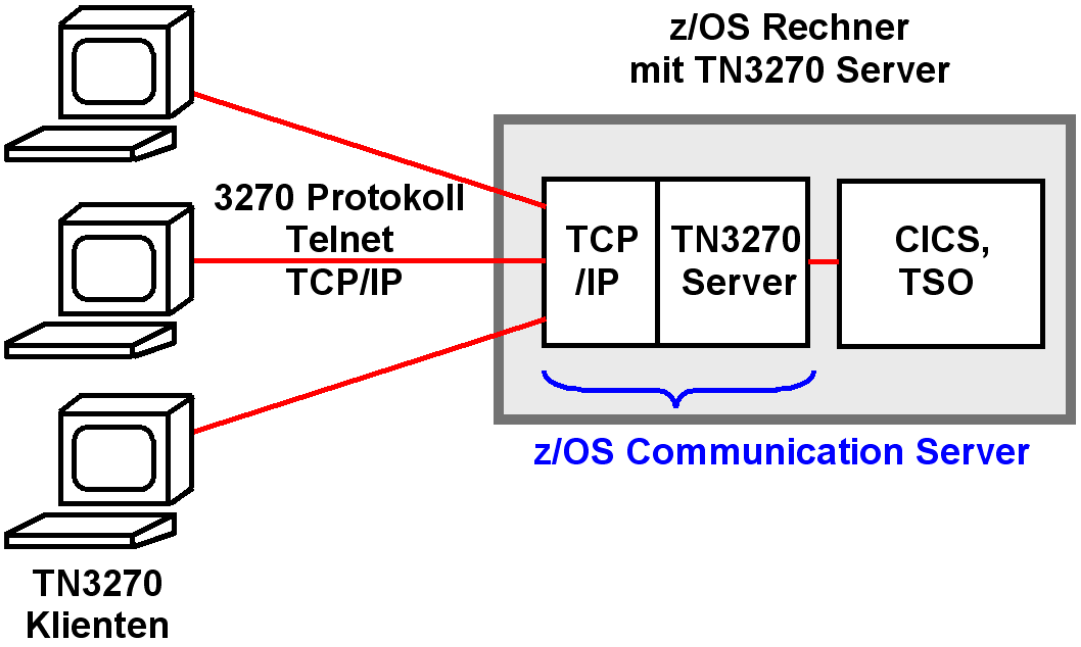


Abb. 18.1.8
TN3270 Protokoll

Heute kommunizieren TSO, CICS und IMS/DC Klienten und Server miteinander nach wie vor über das SNA Netzwerkprotokoll. Es existieren keine Pläne, dies zu ändern.

z/OS Installationen stellen aber ihr Netz von SNA auf TCP/IP um. Für Internet Verbindungen ist TCP/IP erforderlich. Lösung: SNA Protokolle über TCP/IP transportieren.

Die Verbindung eines 3270 Klienten mit dem z/OS Server erfolgt durch das TN3270 Protokoll. TN3270 ist ein erweitertes Telnet Protokoll; aus diesem Grund greifen 3270 Klienten meistens über Port 23 auf einen z/OS Server zu. Dies ist ein hierarchisches Netzwerk.

TN3270 Klienten sind über ein TCP/IP Netzwerk mit dem TN3270 Server verbunden, einer Komponente des z/OS Communication Server Subsystems. Der 3270 Client Klient (z.B. 3270 Emulator) emuliert eine LU2 Session und verpackt jede SNA Nachricht in eine TCP/IP Nachricht.

Der TN3270 Server simuliert jeden angeschlossenen TN3270 Klienten als ein logisches SNA Terminal (LU Typ 2 für Bildschirme, LU Typ 1 und 3 für Drucker).

Zwischen dem TN3270 Server und der z/OS Anwendung wird das SNA Protokoll benutzt. Zwischen dem TN3270 Server und dem TN3270 Klienten wird das 3270 Protokoll benutzt.

TN3270E erweitert das ursprüngliches TN3270 Protokoll und ist für das Internet in RFC 1647 spezifiziert.

18.1.8 TCP62 Protokoll

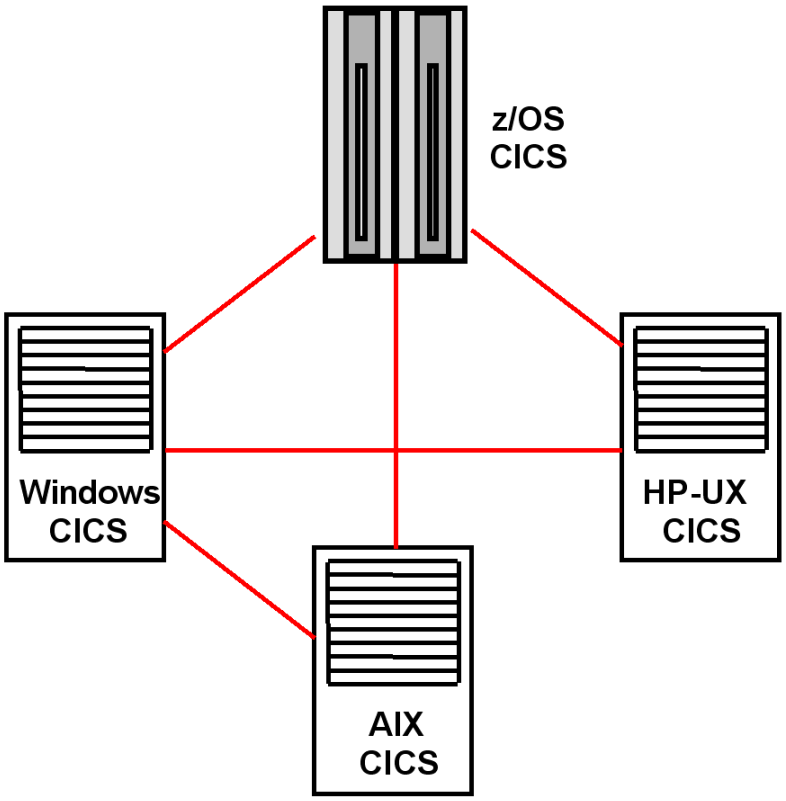


Abb. 18.1.9
Server - Server Konfiguration

Das TN3270 Protokoll ist ein Client/Server Protokoll. Im Gegensatz dazu existiert das Bedürfnis für eine Peer-to-Peer Kommunikation. Ein Beispiel hierfür ist die Kommunikation von zwei CICS Instanzen auf zwei geografisch getrennten z/OS Rechnern.

Die Verbindung erfolgt über das TCP62 Protokoll und eine SNA LU 6.2 –LU 6.2 Session. CICS verwendet hierfür den Distributed Link (DPL) Aufruf.

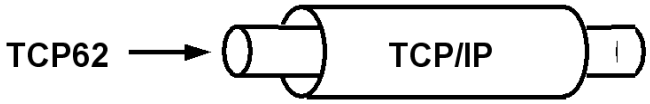


Abb. 18.1.10
Verpackung von TCP62 Nachrichten in TCP/IP Pakete

TCP62 oder TN3270 werden hierzu in TCP/IP Pakete verpackt und über das Internet versendet.

z/OS CICS kommuniziert auf die gleiche Art mit distributed CICS Instanzen auf Platformen wie Windows oder Solaris.

18.1.9 SNA und TCP/IP Protokolle

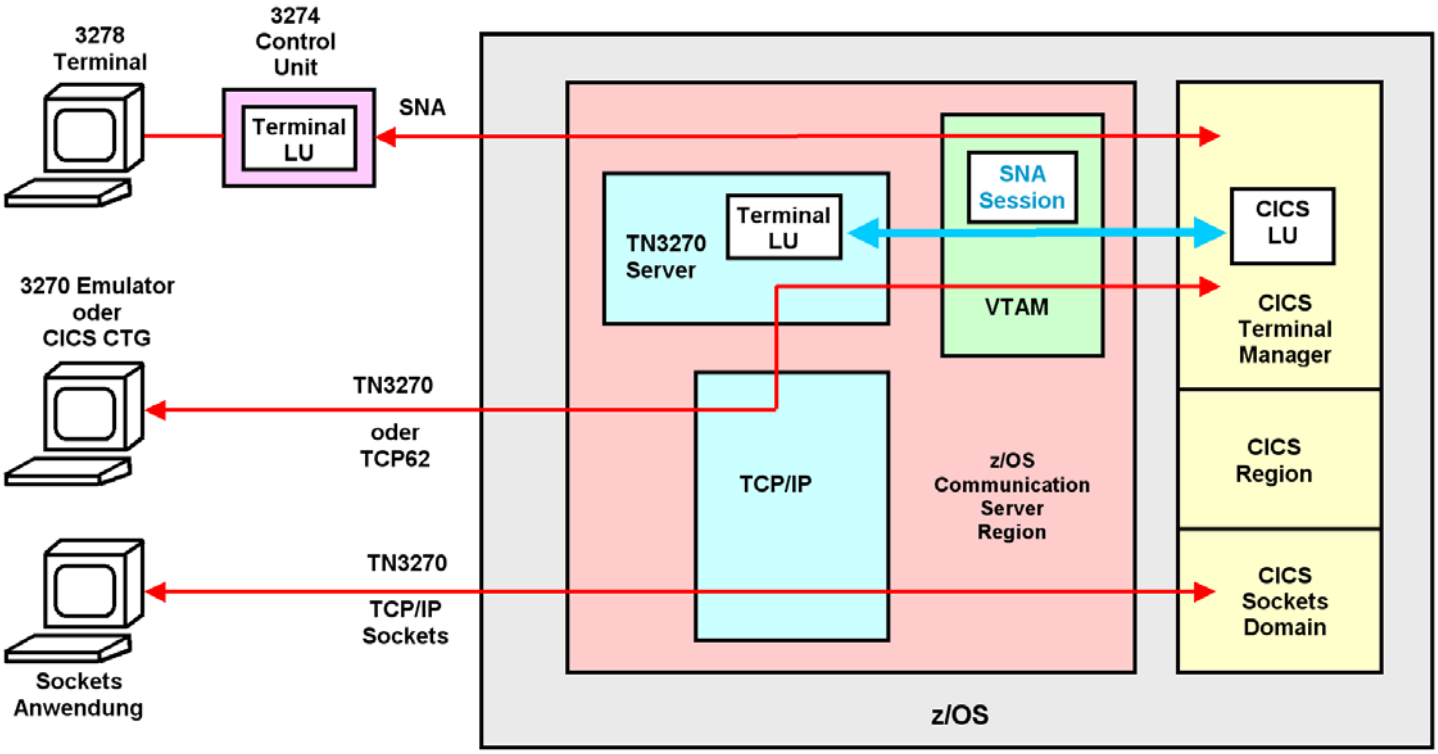


Abb. 18.1.11
CICS verfügt neben SNA Unterstützung auch über eine direkte TCP/IP Unterstützung

Abbildung 18.1.11 zeigt drei Arten der Anbindung eines Terminals an ein z/OS System.

TSO CICS oder IMS/DC Anwendungen glauben, mit einem SNA LU 2 Terminal zu kommunizieren. Der normale 3270 Emulator benutzt TN3270 für eine Verbindung mit z/OS. Der TN3270 Server des z/OS Communication Server Subsystems konvertiert die TN3270 Message in eine SNA Message. Der TN3270 Server kommuniziert mit TSO CICS oder IMS/DC Anwendungen über eine SNA Session über zwei LUs, einer LU2 (Terminal LU) und einer LU5 (z.B. CICS LU).

Heutige PCs haben genügend Leistung, um statt TN3270 das leistungsfähigere TCP62 Protokoll zu benutzen. In diesem Fall, der besonders für CICS interessant ist, wird der 3270 Emulator durch eine TCP62 Komponente, das CICS Transaction Gateway (CTG) ersetzt. Hiermit sind erweiterte Funktionen möglich, die besonders im Zusammenhang mit Java interessant sind.

Sockets sind eine weitere Alternative für Terminal Verbindungen. CICS verfügt über viele Terminal Attachment Alternativen. Eine davon ist die CICS Sockets Schnittstelle.

18.1.10 Zusammenfassung

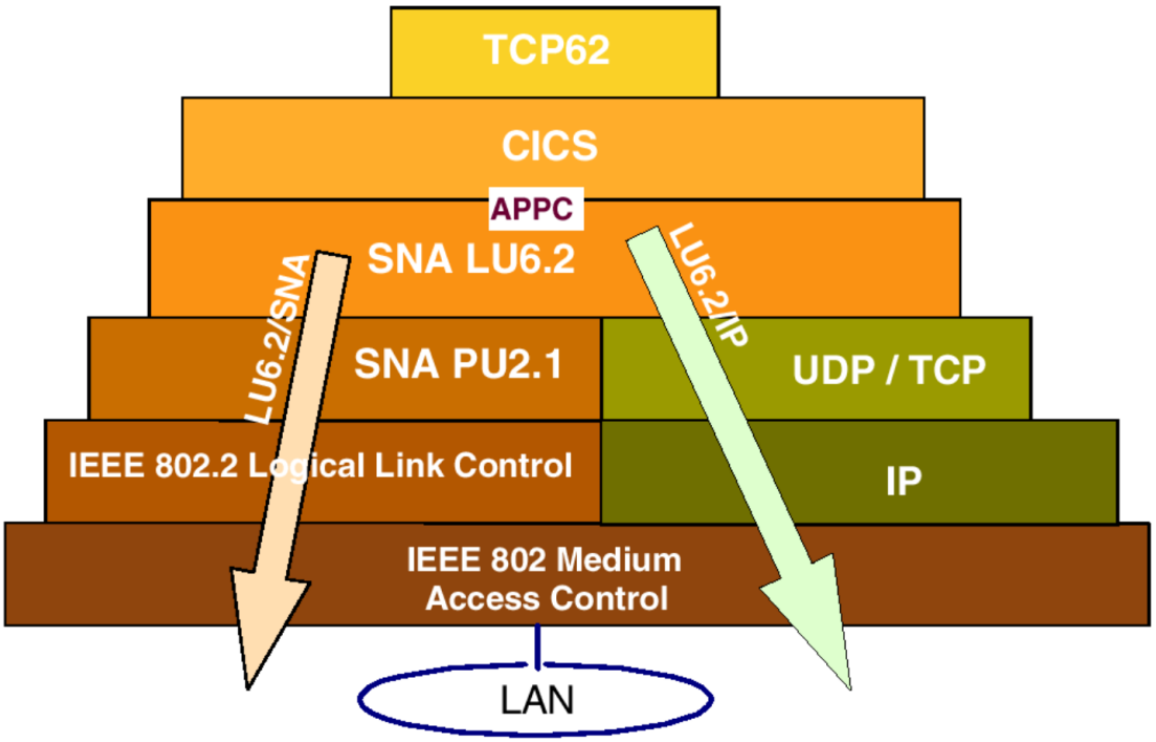


Abb. 18.1.
Koexistenz von SNA und TCP/IP

Ein CICS Subsystem kann mit einem anderen CICS System über SNA und den SNA Protokoll Stack kommunizieren. Die moderne Alternative besteht darin, dass ein CICS System mit einem anderen CICS System mittels des TCP/IP Protokoll Stacks und dem TCP62 Protokoll kommuniziert.

18.2 CICS Interface

18.2.1 Business- und Präsentationslogik

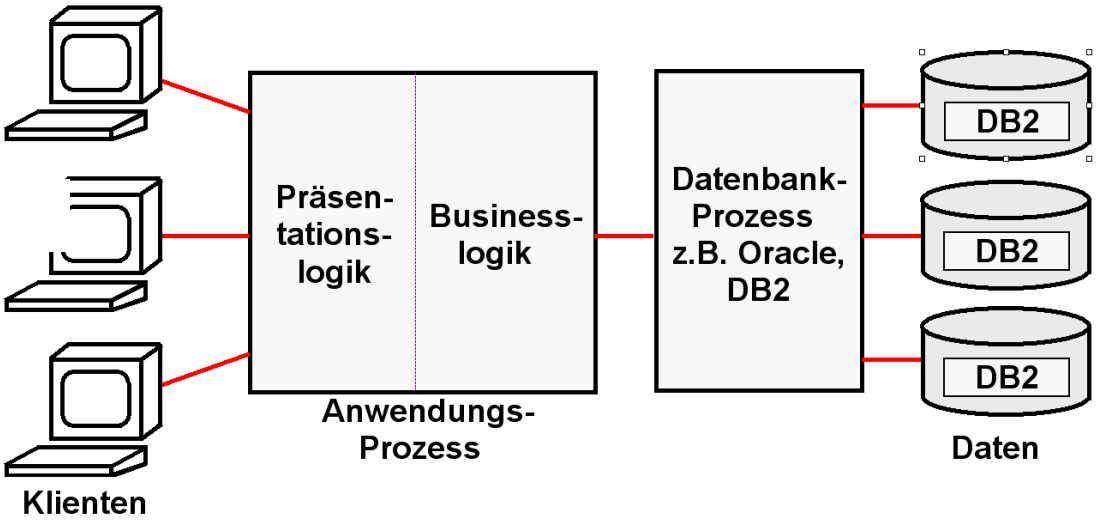


Abb. 18.2.1
Aufteilung in Business Logik und Präsentation Logik

Ein sauber strukturiertes CICS Programm besteht aus zwei Teilen: Business Logik und Präsentations-Logik.

Business Logik ist der Teil, in dem Berechnungen erfolgen und Daten in einer Datenbank gelesen/geschrieben werden.

Präsentations- Logik ist der Teil, in dem die Ergebnisse der Berechnungen so aufgearbeitet werden, dass sie dem Benutzer in einer ansprechenden Art auf dem Bildschirm dargestellt werden können.

Business Logik wird in Sprachen wie C, C++, COBOL, PL/1, Java usw. geschrieben.

Für die Präsentations- - Logik gibt es viele Möglichkeiten. Eine moderne Alternative benutzt Java Server Pages und einen Web Application Server um den Bildschirminhalt innerhalb eines Web Browsers darzustellen.

Die älteste (und einfachste) Alternative verwendet das CICS BMS (Basic Mapping Support) Subsystem. BMS Programme werden in der BMS Sprache geschrieben.

18.2.2 z/OS Internet Integration

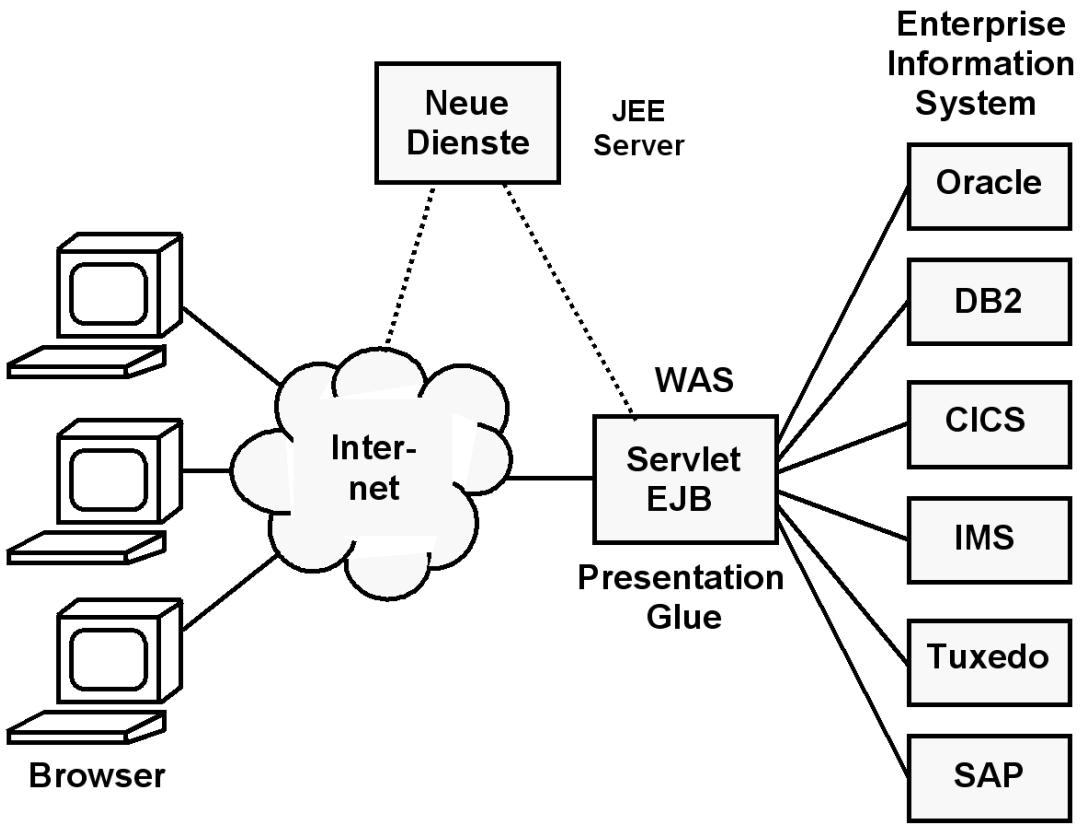


Abb. 18.2.2
z/OS Anbindung an das Internet

Unternehmenskritische Anwendungen und Datenbank-Prozesse laufen in der Regel auf einem zentralen Server. In mittleren und großen Unternehmen und Organisationen ist dies in der Regel ein z/OS Rechner. Aufgabenstellung:

- **Die existierende IT-Infrastruktur mit den Möglichkeiten des Internets integrieren.**
- **Die existierende IT-Infrastruktur so umstrukturieren, so dass sie mit weniger Personal an die sich in immer kürzeren Zeiträumen ändernden Geschäftsbedingungen angepasst werden kann.**

  Hierfür hat sich die Namen Enterprise Application Integration (EAI), sowie Service Oriented Architecture (SOA) eingebürgert.

Die Enterprise Application Integration (EAI) ist ein Ansatz für die Integration von Applikationen und Datenquellen. Dieser soll den Austausch von Daten und die Verbindung von Geschäftsabläufen vereinfachen. Es soll versucht werden, die Zugriffe der unterschiedlichsten Art von (Java) Klienten auf die unterschiedlichsten Arten von Enterprise Information Systems (EIS) zu vereinheitlichen und zu automatisieren. Beispiele für Eins sind CICS, IMS, Oracle, DB2, Tuxedo.

EAI ist im Wesentlichen ein rein technischer Ansatz zur Integration von Anwendungssystemen. Service Oriented Architecture (SOA) betont die Verbindung zu den nicht-technischen Geschäftsprozessen.

18.2.3 Alternativen der CICS Bildschirmausgabe

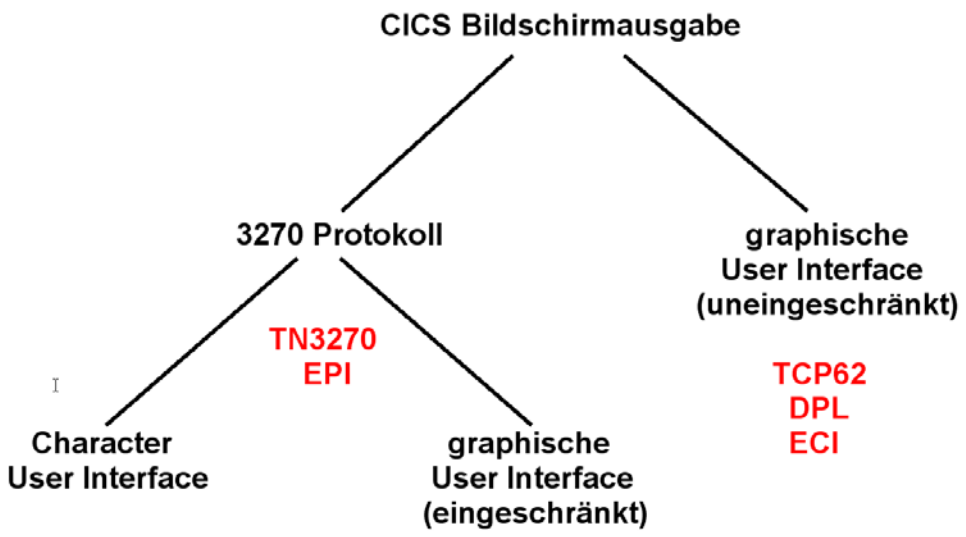


Abb. 18.2.3
Zwei Arten der CICS grafischen Bildschirmausgabe

Die CICS Bildschirmausgabe erfolgt entweder über das 3270 Protokoll unter Nutzung der CICS Terminal Control Komponente und des Basic Mapping Supports (BMS), oder über einen direkten COMMAREA Zugriff.

Im ersten Fall erfolgt die Datenübertragung mit Hilfe des TN3270 Protokolls und der EPI Schnittstelle. Es ist zunächst eine Character User Darstellung möglich (Green Screen). Alternativ kann der 3270 Datenstrom mit Hilfe eines als „Screen Scraping“ bezeichneten Ansatzes grafisch dargestellt werden.

Im zweiten Fall erfolgt die Datenübertragung mit Hilfe des TCP62 Protokolls und der ECI Schnittstelle. Es wird die CICS Distributed Program Link (DPL) Kommunikation eingesetzt. Einschränkungen in den Möglichkeiten der grafischen Darstellung, die im ersten Fall durch das 3270 Protokoll bedingt sind, werden hierbei ausgeschlossen.

18.2.4 Ablauf einer COMMAREA Operation

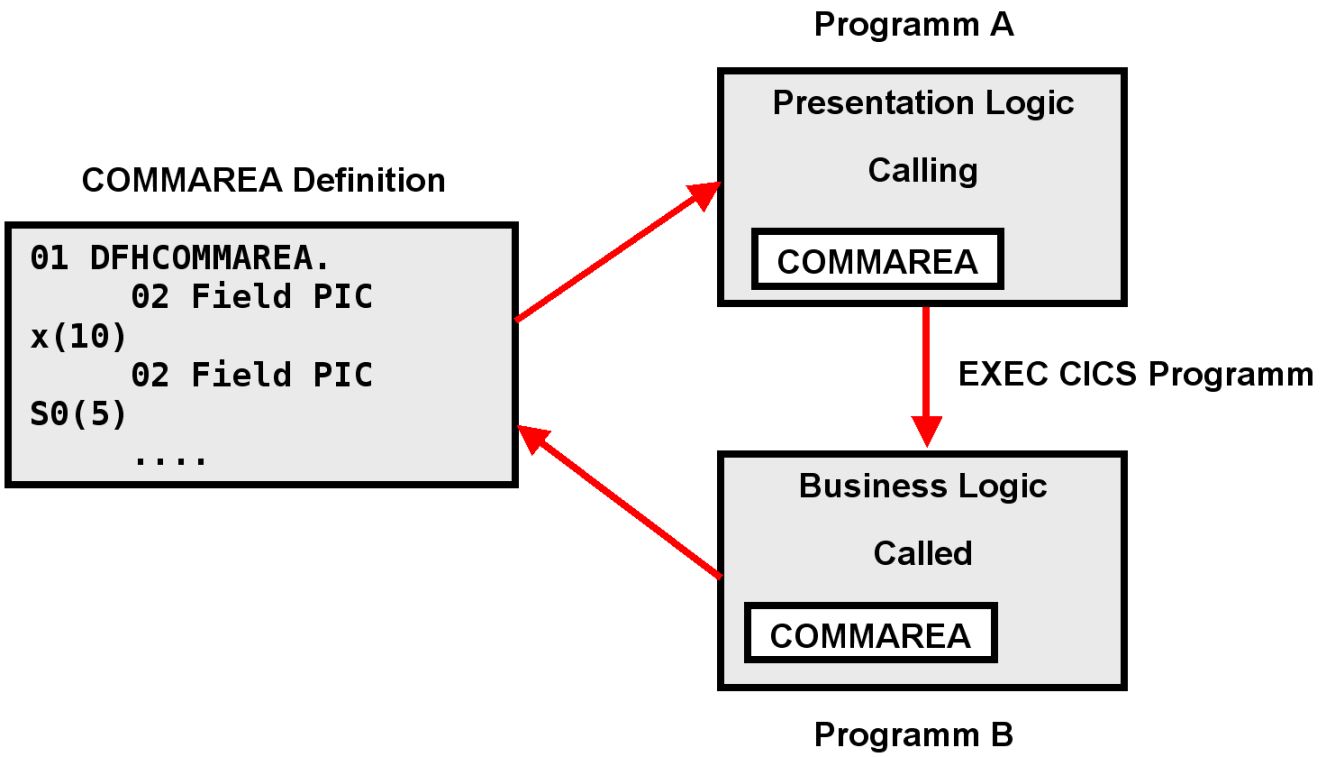


Abb. 18.2.4
Kommunikation mittels COMMAREA

Für die Kommunikation von Eingabe und Ausgabe Informationen extrahiert das aufrufende (calling) Presentation Logik Programm Daten aus der Eingabe Nachricht, und setzt Eingabefelder in der COMMAREA. Es ruft dann ein weiteres (called) CICS-Programm auf. Dieses Programm legt die Ergebnisse der Business Logik Operation wieder in Felder der COMMAREA ab und gibt die Kontrolle an das aufrufende Programm zurück. Letzteres macht das Ergebnis über die COMMAREA verfügbar.

```
01 DFHCOMMAREA.
     02  CA-RETCODE                    PIC 9(8) COMP.
     02  CA-SWSECI1-COMMAREA.
         05  CA-NUMBER-OF-ROWS         PIC 9(4) COMP.
         05  CA-ERROR-MESSAGE          PIC X(10).
         05  CA-CURRENT-DATE           PIC X(8).
         05  CA-CURRENT-TIME           PIC X(8).
         05  CA-CICS-ABSTIME           PIC S9(15) COMP-3.
         05  CA-ROW-DATA  OCCURS 1  TO 1818 TIMES
                          DEPENDING ON CA-NUMBER-OF-ROWS.
             10  CA-ROW-NUMBER         PIC S9(4) COMP.
             10  CA-ROW-NUM-AS-CHAR    PIC X(6).
             10  CA-DATA               PIC X(10).
```

Abb. 18.2.5
Beispiel für den Inhalt einer COMMAREA

https://awebproxyprd.ins.state.ny.us/docs/aiciref/aicirefp10.htm

18.2.5 Die 3270 Schnittstelle

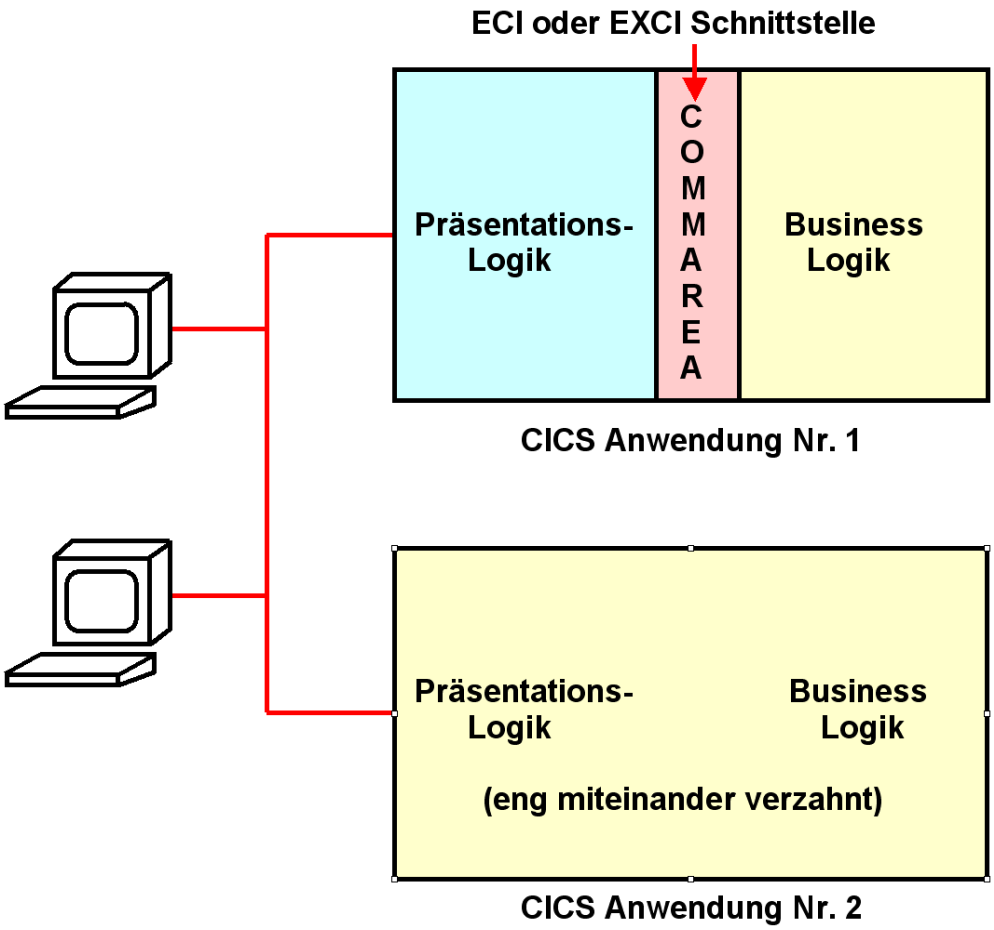


Abb. 18.2.6
COMMAREA als Schnittstelle zwischen Business Logik und Präsentation Logik

Aufgabe: Ersatz der 3270 Schnittstelle durch eine Web Browser Schnittstelle oder eine andere GUI.

Bei modernen CICS Anwendungen (Nr. 1) kommunizieren Business Logik und Präsentationslogik über COMMAREA. Hier ist es relativ einfach, eine Präsentationslogik durch eine andere zu ersetzen, oder für eine gegebene Businesslogik mehrere alternative Präsentationslogiken anzubieten.

Bei vielen älteren CICS Anwendungen (Nr. 2) sind Presentation Logik und Business Logik beide in z.B. Cobol oder PLI geschrieben und eng miteinander verwoben. Es ist vielfach nicht möglich oder sinnvoll, diese Anwendungen umzuschreiben, um eine saubere Trennung zwischen Business Logik und Presentation Logik zu erreichen. Hier ist eine Modernisierung der Oberfläche nur mittels Screen Scraping möglich.

18.2.6 Screen Scraping Implementierung mittels Java

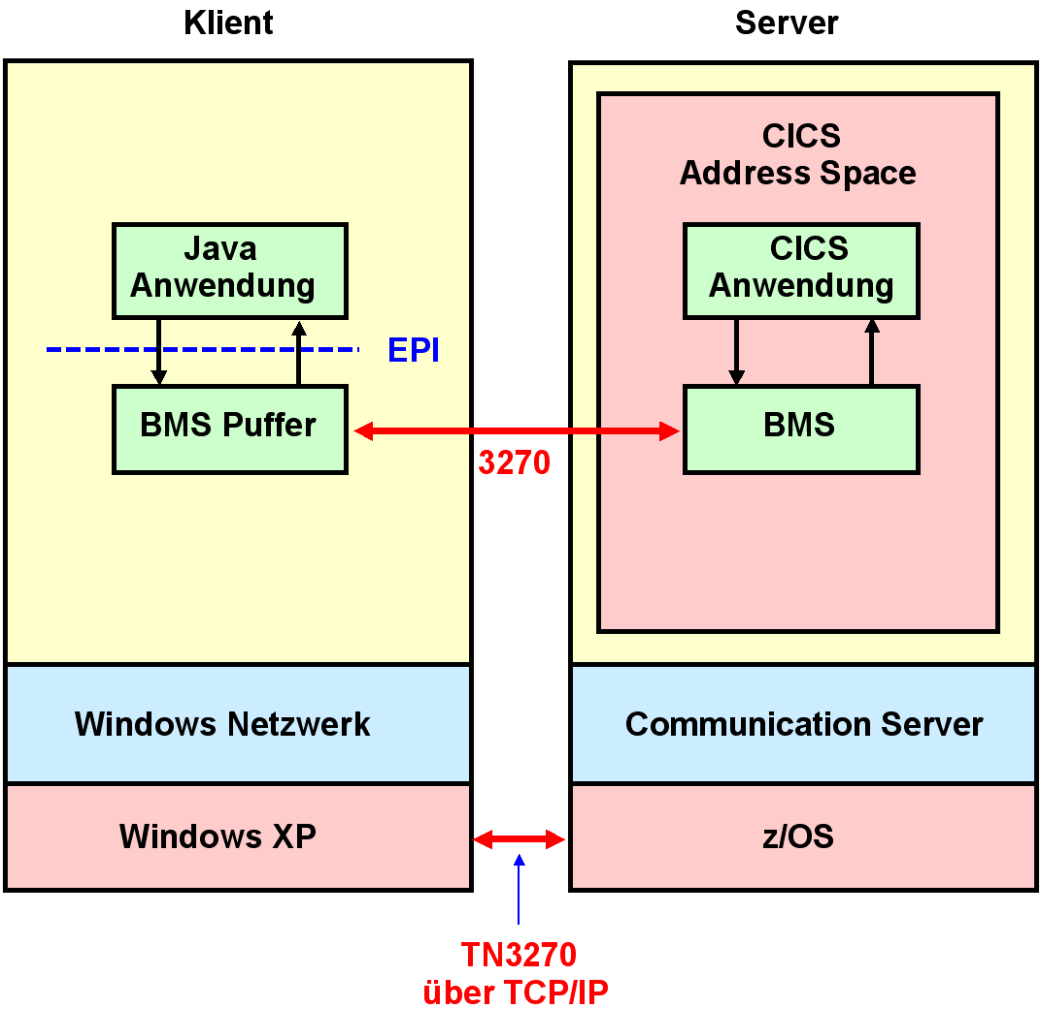


Abb. 18.2.7
Java Screen Scraping Programm auf dem Klienten

CICS Screen Scraping kann mittels eines Java Programms auf dem Klienten Rechner implementiert werden. Hierbei überträgt ein 3270 Datenstrom eine Ausgabenachricht, wie gehabt, in den 24 x 80 Byte großen BMS Puffer des 3270 Emulators. Das Java Programm greift über die EPI Schnittstelle auf den BMS Puffer Inhalt zu.

Eine Java Anwendung auf dem Klienten kann über die EPI Schnittstelle auf den Inhalt des BMS Puffers zugreifen, und mit dessen Daten eine gefälligere graphische Oberfläche (GUI) erzeugen. Die Anwendung kann 3270-basierte CICS Transaktionen starten und Daten senden und empfangen, die mit dieser Transaktion assoziiert sind. Die Anwendung kann alternativ auch in Java, C++, PL/1 oder einer beliebigen anderen Programmiersprache geschrieben sein.

Als Alternative zu dem hier dargestellten Verfahren kann die Screen Scraping Logik auf einem Server laufen.

Der Basic Mapping Support (BMS) ist Bestandteil des CICS Terminal Managers. Er wurde in Band 1, Abschnitt 9.1 erläutert.

18.2.7 Host Access Transformation Services

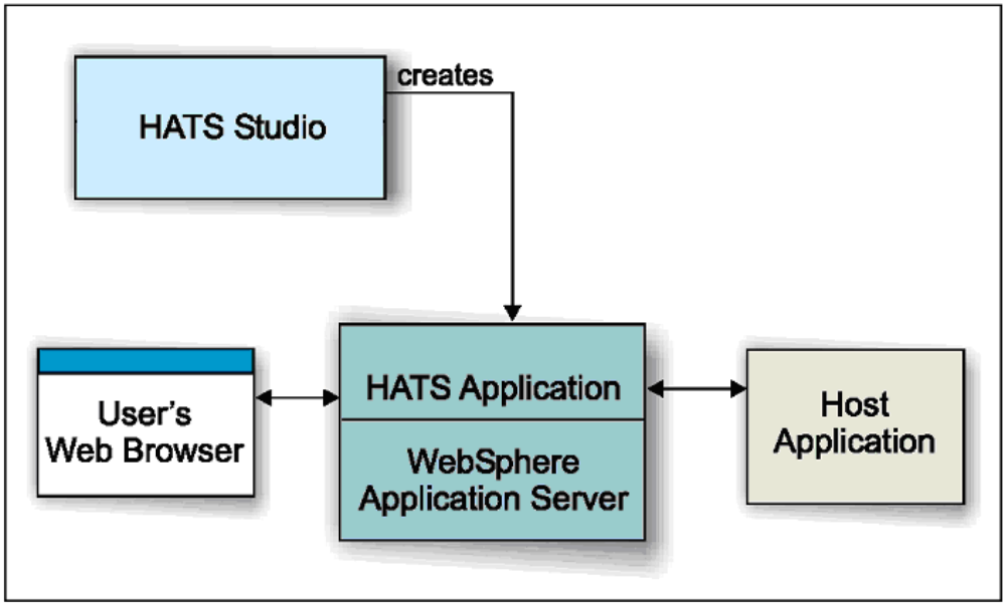


Abb. 18.2.8
HATS Studio erlaubt Erweiterungen der voll automatischen Konvertierung

Host Access Transformation Services (HATS) ermöglicht Screen Scraping auf einem Server. Der HATS Server erstellt für eine Reihe angeschlossener Klienten 3270 Bildschirmdarstellungen als HTML Seiten. HATS erkennt automatisch die Komponenten des 3270 Bildschirm-Inhaltes mit Hilfe eines Satzes vordefinierter Regeln. Es übersetzt die Komponenten des 3270 Bildschirm-Inhaltes in Echtzeit in HTML. Auf dem Klienten ist nur ein Browser erforderlich.

Im einfachsten Fall übersetzt der HATS Server den 3270 Datenstrom automatisch und unmittelbar in eine HTML Seite, die an den Browser des Terminals gesendet wird. HATS Studio ist eine optionale Entwicklungsumgebung, mit deren Hilfe das Erscheinungsbild auf dem Browser verbessert und angepasst werden kann.

Als Beispiel wird die in Abb. 18.2.9 gezeigte Bildschirmdarstellung ...

```
MAIN                          OS/400 Main Menu

Select one of the following:

    1.  User tasks
    2.  Office tasks
    3.  General system tasks
    4   Files, libraries, and folders
    5.  Programming
    6.  Communications
    7.  Define or change the system
    8.  Problem handling
    9.  Display a menu
    10. Information Assistant options
    11. Client Access/400 tasks

    90. Sign off
Selection or command
===>

F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu
```

Abb. 18.2.9
Ursprüngliche Darstellung

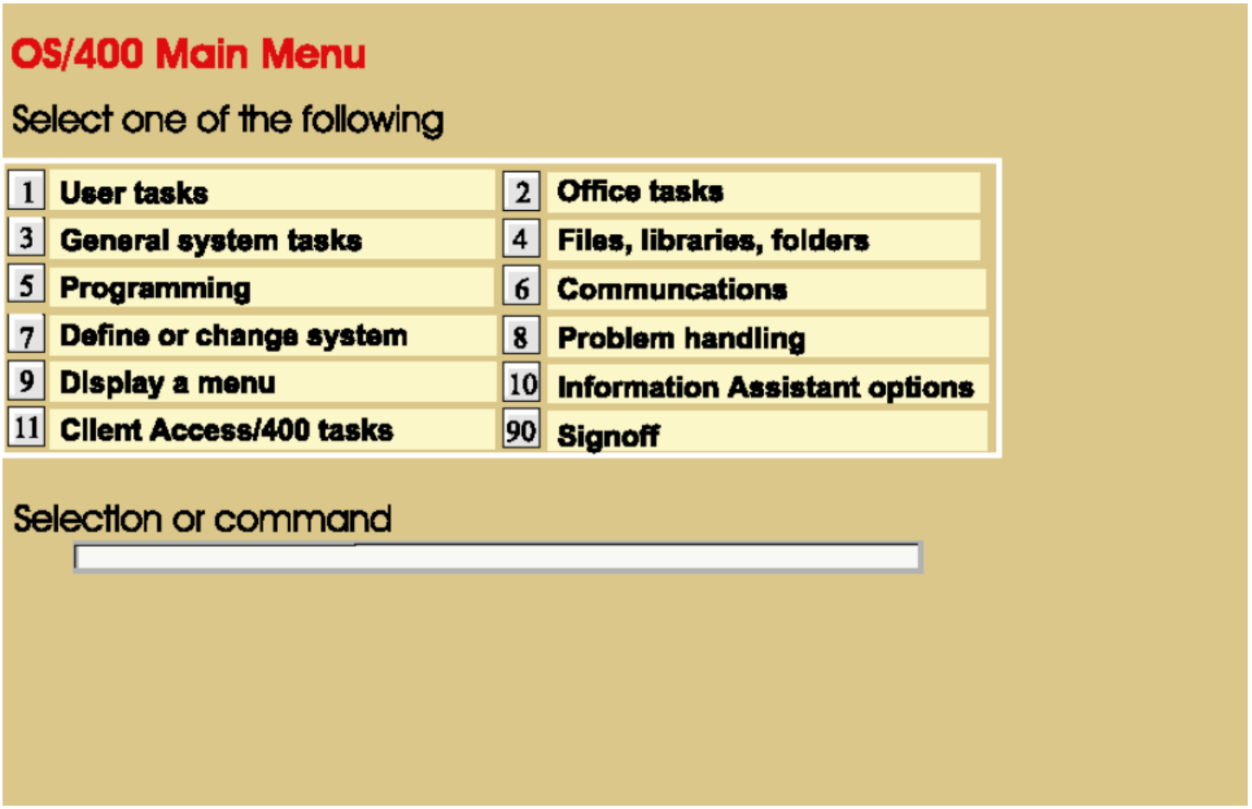


Abb. 18.2.10
Mit HATS automatisch erstellte Darstellung

... in die Darstellung von Abb. 18.2.10 automatisch übersetzt. Das kann für eine große Anzahl von Maps mittels eines automatisch ablaufenden Verarbeitungsvorgang geschehen, ohne dass jede einzelne Map angefasst werden muss.

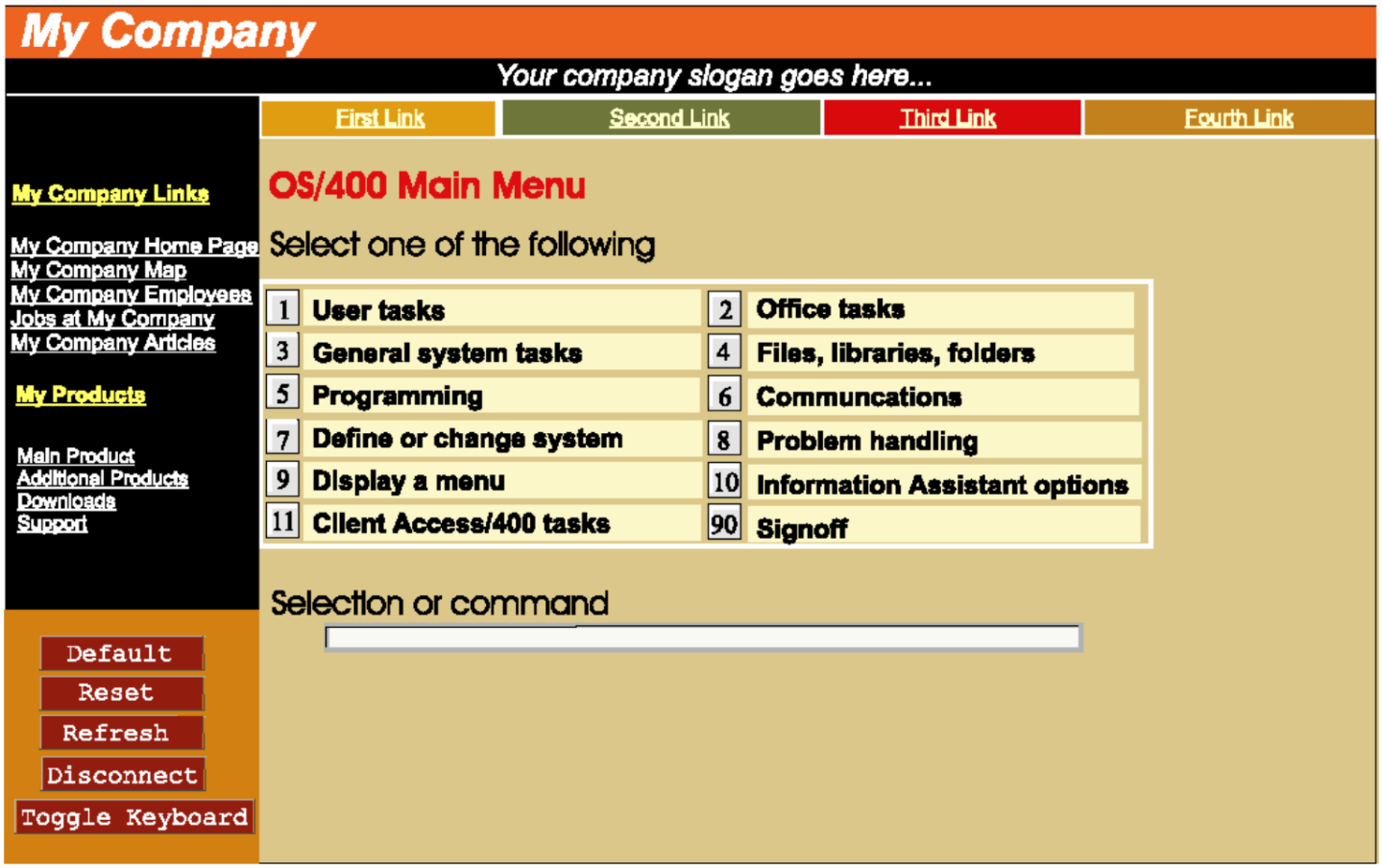


Abb. 18.2.11
Erweiterung mittels HATS Studio

Bei dieser Gelegenheit bietet sich an, die Bildschirmdarstellung um zusätzliche Elemente zu erweitern, die nicht Bestandteil des 3270 Datenstroms sind. Beispiele sind die Einfügung eines Firmenlogos, eine zusätzliche Liste mit Links, usw. Dies erfordert allerdings zusätzlichen Anpassungsaufwand, der mit Hilfe des HATS Studio erbracht werden kann.

18.2.11 HATS Beispiel Universität Tübingen

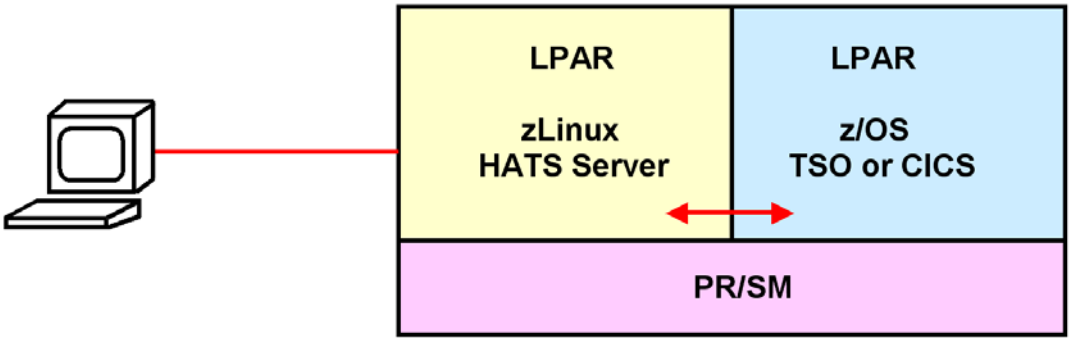


Abb. 18.2.12
HATS installiert auf einer zLinux LPAR

Sie können das vorliegende Beispiel selbst durchführen, indem Sie sich unter

http://galadriel.cs.uni-tuebingen.de:9080/csprak/

einloggen. Die folgenden Abbildungen zeigen einige Beispiele

```
z/OS Z18 Level 0609                                   IP Address = 88.64.140.11
                                                       VTAM Terminal = SC0TCP13

                        Application Developer System

                                 //  OOOOOOO   SSSSS
                                //  OO     OO SS
                        zzzzzz //  OO     OO SS
                          zz  //  OO     OO SSSS
                        zz   //  OO     OO      SS
                      zz    //  OO     OO      SS
                    zzzzzz //   OOOOOOO  SSSS


                    System Customization - ADCD.Z18.*



 ===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
 ===> Enter L followed by the APPLID
 ===> Examples: "L TSO", "L CICS", "L IMS3270

l tso_
```

Aus dem z/OS Welcome Screen wird …

Abb. 18.2.13
Eine mit HATS erstellte Darstellung

```
------------------------------- TSO/E LOGON ----------------------------------

   Enter LOGON parameters below:                    RACF LOGON parameters:

   Userid    ===> PRAK031

   Password  ===>                                   New Password ===>

   Procedure ===> DBSPROC                           Group Ident  ===>

   Acct Nmbr ===> ACCT#

   Size      ===> 5000

   Perform   ===>

   Command   ===> ISPF

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Aus dem Logon Screen wird ....

Abb. 18.2.14
eine mit HATS erstellte Darstellung

```
  Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu                 End of data

 0  Settings      Terminal and user parameters          User ID . : SPRUTH
 1  View          Display source data or listings       Time. . . : 21:42
 2  Edit          Create or change source data          Terminal. : 3278
 3  Utilities     Perform utility functions             Screen. . : 1
 4  Foreground    Interactive language processing       Language. : ENGLISH
 5  Batch         Submit job for language processing    Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands     TSO logon : ISPFPROC
 7  Dialog Test   Perform dialog testing                TSO prefix: SPRUTH
 9  IBM Products  IBM program development products     System ID : ADCD
 10 SCLM          SW Configuration Library Manager      MVS acct. : ACCT#
 11 Workplace     ISPF Object/Action Workplace          Release . : ISPF 6.1
 M  More          Additional IBM Products

      Enter X to Terminate using log/list defaults

 Option ===> _____________________________________________________________
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

und aus dem ISPF Primary Option Menu wird ....

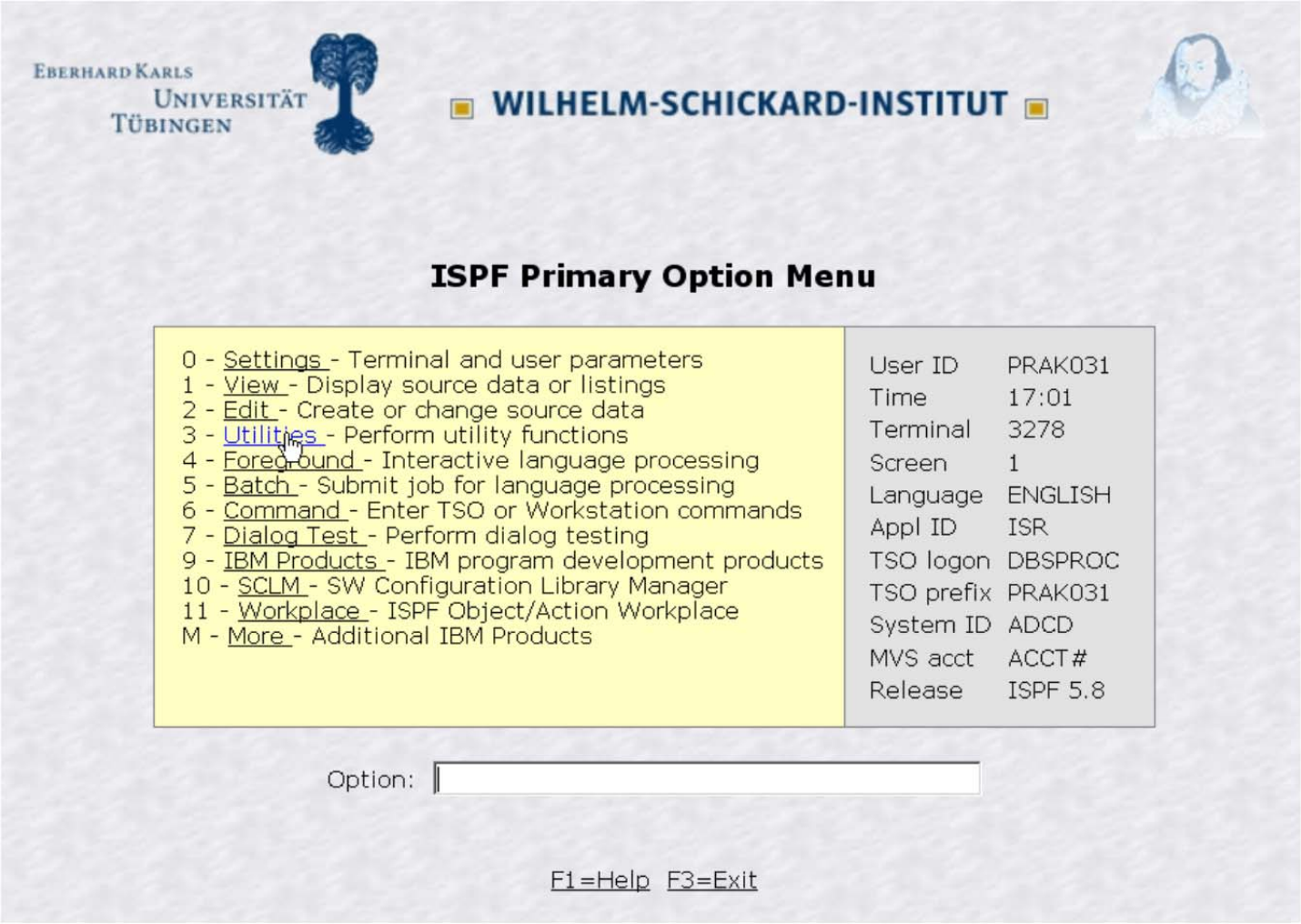


Abb. 18.2.15
diese mit HATS erstellte Darstellung

18.3 CICS Transaction Gateway

18.3.1 Moderne Oberflächen

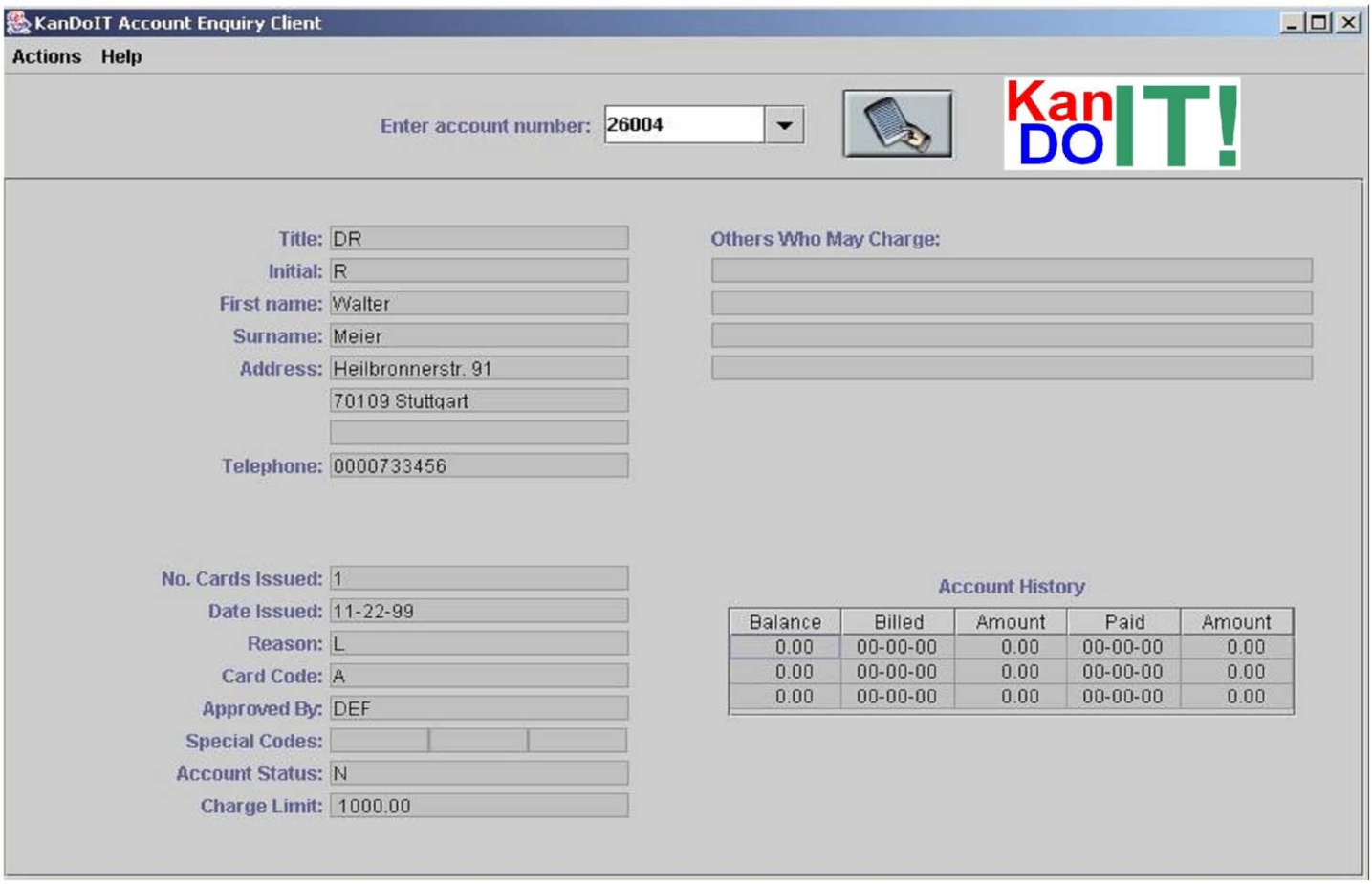


Abb. 18.3.1
Alternative für einen BMS Screen

In traditionellen Fällen benutzen alle Screens das 1971 entstandene „3270 Protokoll“ und die BMS (Basic Mapping Support) Präsentationslogik, welche Bestandteil von jedem CICS TP Monitor ist.

Eine Alternative sind moderne Benutzeroberflächen, die in der großen Mehrzahl der Fälle mit Hilfe von Java programmiert werden. Ein einfaches Beispiel ist in Abb. 18.3.1 wiedergegeben.

In der Wirtschaft und Verwaltung wurden in den letzten Jahren die allermeisten CICS Anwendungen hiermit ausgestattet worden. Dies geschah fast immer als Alternative (und nicht als Ersatz) zu der existierenden BMS Präsentationslogik, die heute immer noch sehr gebräuchlich ist.

18.3.2 COMMAREA als Kommunikationsschnittstelle

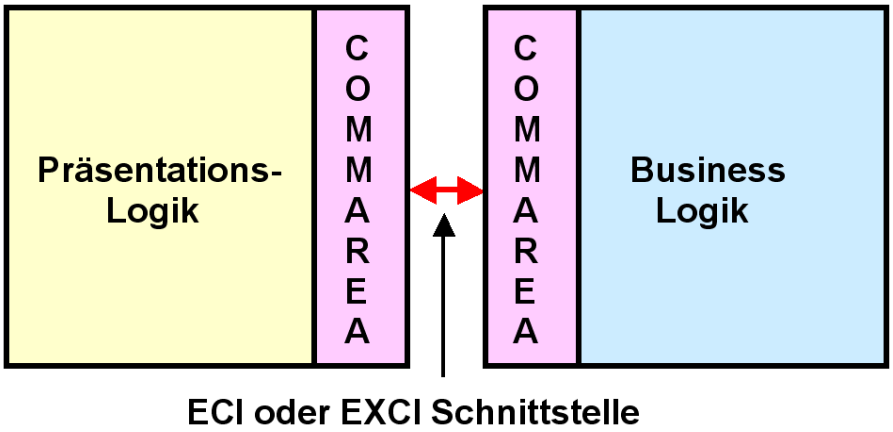


Abb. 18.3.2
Kommunikation über die COMMAREAs von zwei CICS Instanzen

Eine CICS Anwendung besteht aus der Business Logik und der Präsentations- Logik.

Es ist guter Programmierstiel, diese beiden Funktionen voneinander zu trennen und in getrennten Programm Modulen unterzubringen.

Für die Kommunikation zwischen den beiden Modulen wird ein Pufferbereich benötigt. Hierfür bietet sich der COMMAREA Puffer an, dar von der Storage Manager Komponente des CICS Subsystems bereitgestellt wird. COMMAREA ist Bestandteil des Scratchpad - Speicherbereiches der CICS Storage Manager Komponente. Der Scratchpad wird u.a. auch für die Verwaltung von Sessions verwendet, wobei der State einer Transaktion für die Nachfolgetransaktion verfügbar ist.

Für die Kommunikation einer beliebigen Implementierung der Präsentations- Logik mit COMMAREA existiert eine Schnittstelle, die „External Call Interface“ (ECI). Sie verwendet die CICS „Distributed Program Link“ (DPL) Interprocess Communication Einrichtung. DPL ist ein Verfahren ähnlich einem RPC. Ein CICS Programm kann ein anderes CICS Programm mit dem „EXEC CICS LINK (Parameter)“ Befehl aufrufen. Beide Programme können sich auf dem gleichen Rechner befinden, oder über das Netzwerk miteinander kommunizieren.

Befinden sich beide Programme auf dem gleichen z/OS Rechner (oder Sysplex), kann eine als EXCI bezeichnete Variante der ECI Schnittstelle verwendet werden. Diese verwendet Pipes in einem Speicherbereich des z/OS Kernels und vermeidet den Kommunikation Overhead.

18.3.3 CICS Universal Client

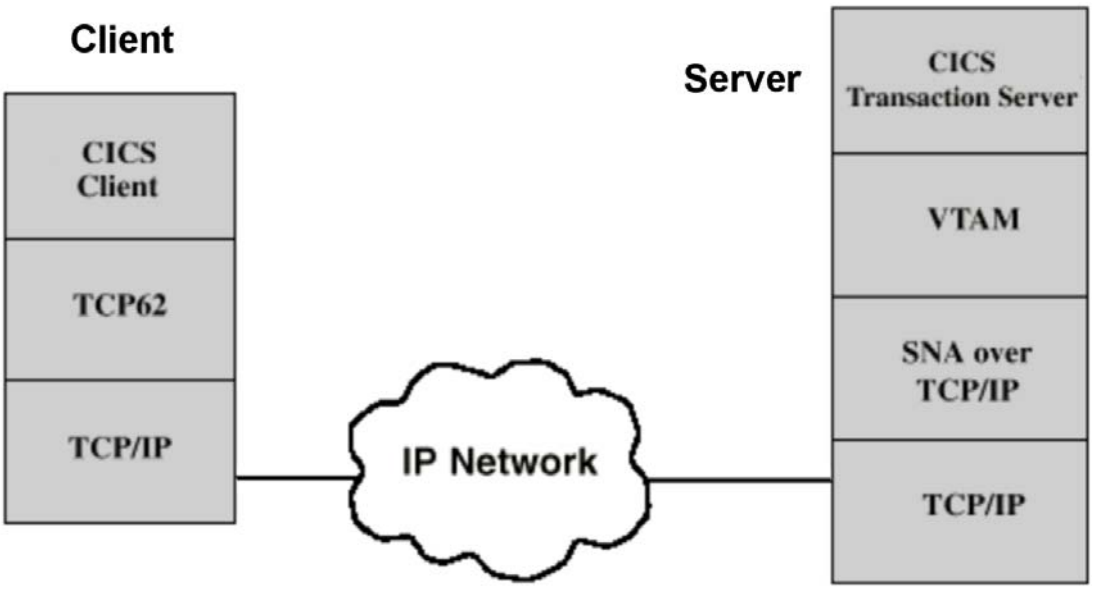


Abb. 18.3.3
Schichtenmodell für die CICS Universal Client Kommunikation

Ein CICS Server kann mit einem anderen CICS Server über eine Peer-to-Peer Verbindung kommunizieren. Dies benutzt die SNA Protokolle APPC, LU 6.2 und TCP62. Die CICS Peer-to-Peer Verbindung erfordert ein CICS System an beiden Communication Endpunkten. Daten werden zwischen den COMMAREAs der beiden CICS Server ausgetauscht.

Nehmen wir an, Sie möchten diese Funktionalität auf Ihrer Workstation haben um mit einem CICS Server zu kommunizieren. Ein Vorteil ist, Sie müssen nicht mit den Einschränkungen des 3270 Protokolls leben. Die Benutzung von DPL und ECI erfordert, dass auf dem Klienten ein funktionsfähiger CICS Transaktionsmonitor installiert wird.

Eine Möglichkeit der Implementierung ist es, einen vollständig en CICS Transaktionsmonitor auf Ihrem Windows oder Linux Arbeitsplatzrechner zu installieren. Dies ist Overkill, wenn Sie lediglich über die TCP62 Peer-to-Peer Funktionalität verfügen wollen. In diesem Fall können Sie eine spezielle Software auf Ihrer Workstation installieren, den "CICS Universal Client", Der CICS Universal Client simuliert ein vollständiges CICS System, verfügt aber lediglich über die Funktionen, die für eine TCP62 Verbindung erforderlich sind.

Der CICS Universal Klient verhält sich bezüglich des DPL Calls wie ein reguläres CICS System, hat sonst aber nur rudimentäre Funktionen.

Spezifisch benutzt der CICS Klient das SNA Protokoll für die Kommunikation mit einem CICS Server (weil auch der heutige CICS Server nur SNA versteht). Die Kommunikation über das Internet und TCP/IP erfolgt über das TCP62 Protokoll.

Heute bezeichnet man den CICS Universal Client oft als „CICS Client".

18.3.4 CICS Universal Client Operation

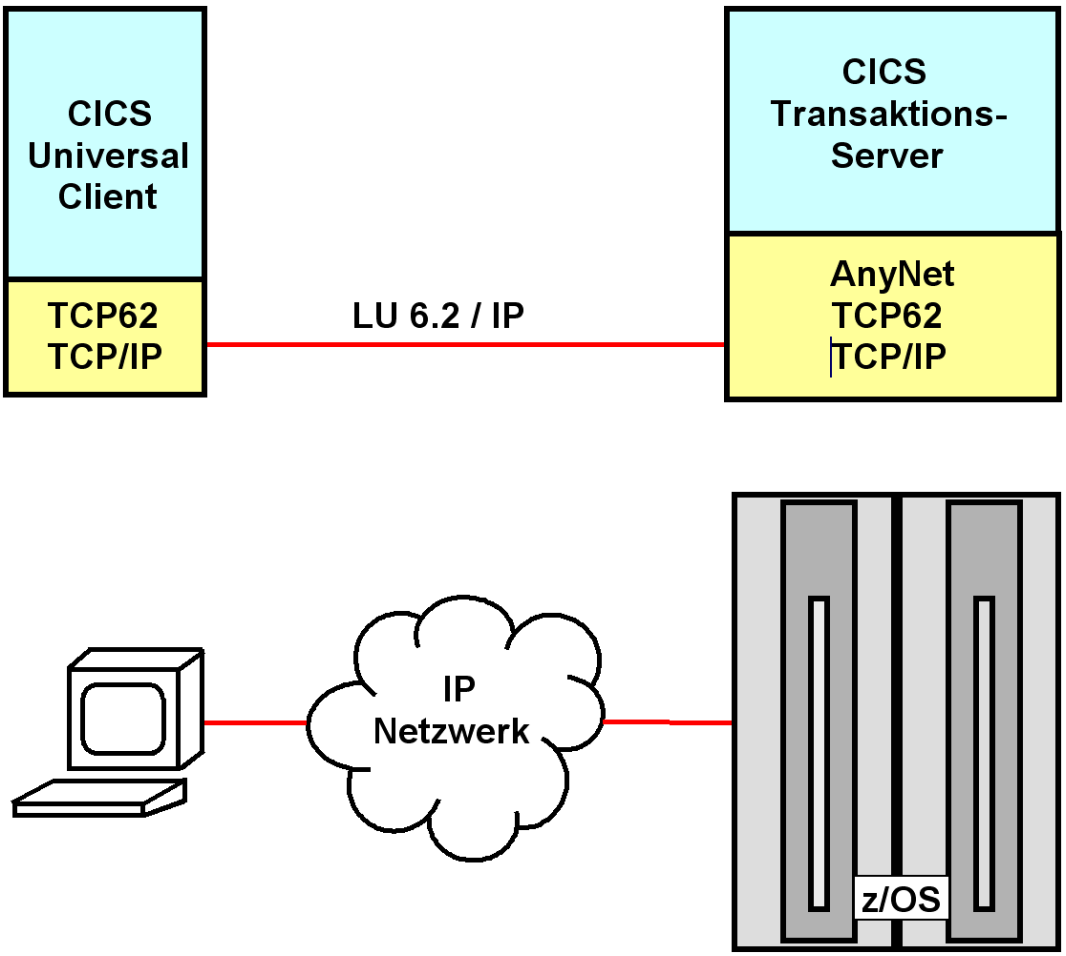


Abb. 18.3.4
Logical Unit Kommunikation

1. Der CICS Universal Client, unter Benutzung der CICSCLI.INI Definitionen, übergibt Daten an die TCP62 Komponente der Workstation.

2. Die TCP62 Komponente auf der Client Workstation verwendet den Domain-Namen Suffix des Partner LU Namens, um einen Internet Protocol (IP) Namen zu generieren. Die IG-Adresse für diesen Namen wird dann entweder von der lokale IP-Hosts-Datei ermittelt, oder von einem Domain Name Server (DNS). Die Daten werden von der TCP/IP-Komponente auf der Workstation an die TCP/IP Komponente des z/OS weiter gereicht

3. TCP/IP auf dem z/OS Host leitet die empfangenen Daten von der Workstation an die SNA über TCP/IP Komponente des z/OS Host weiter.

4. Die SNA über TCP/IP Komponente übersetzt die eingehenden IP-Routing Informationen in SNA Routing-Informationen. Die Daten werden an VTAM übertragen und von dort an den CICS Transaction Server für z/OS weiter geleitet.

18.3.5 CICS Distributed Program Link

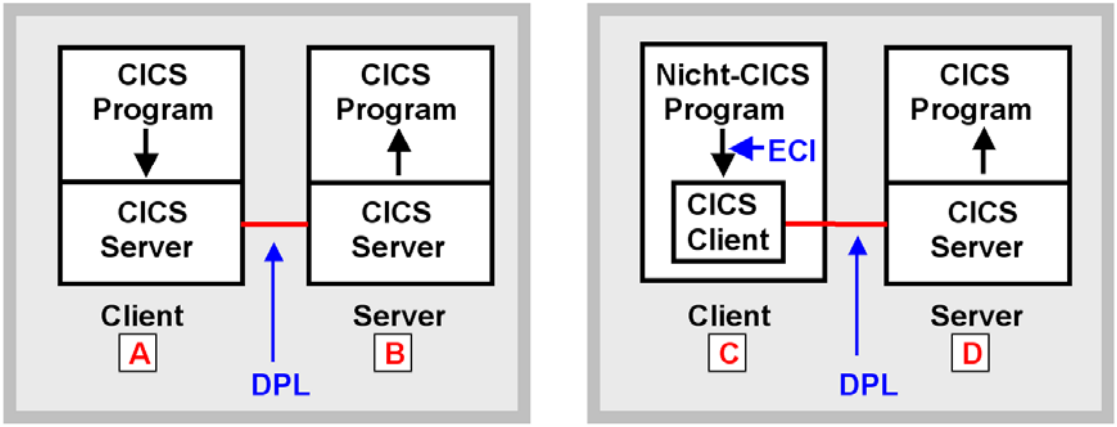


Abb. 18.3.5
CICS Distributed Program Link

Die linke Anordnung zeigt zwei CICS Systeme. Ein Anwendungsprogramm in dem ersten CICS System A ruft ein Anwendungsprogramm in dem zweiten CICS System B mit Hilfe des EXEC CICS LINK Kommandos auf:

```
EXEC CICS LINK PROGRAM(name) COMMAREA(data-area)
```

Das Distributed Program Link (DPL) Protokoll erlaubt es einem Programmierer, das LU 6.2 Link zu benutzen ohne Kenntnis des LU 6.2 Protokolls.

Die rechte Anordnung zeigt einem Client Rechner C, z.B. eine Linux oder Windows Workstation, auf dem der CICS Universal Client installiert ist. Ein nicht-CICS Anwendungsprogramm auf dem Klienten kann sich mit einem CICS Anwendungsprogramm auf dem CICS Server D verbinden, indem es den CICS Client über die ECI Schnittstelle aufruft. Dieser verbindet sich ebenfalls über DPL mit dem CICS Server

18.3. ECI und DPL

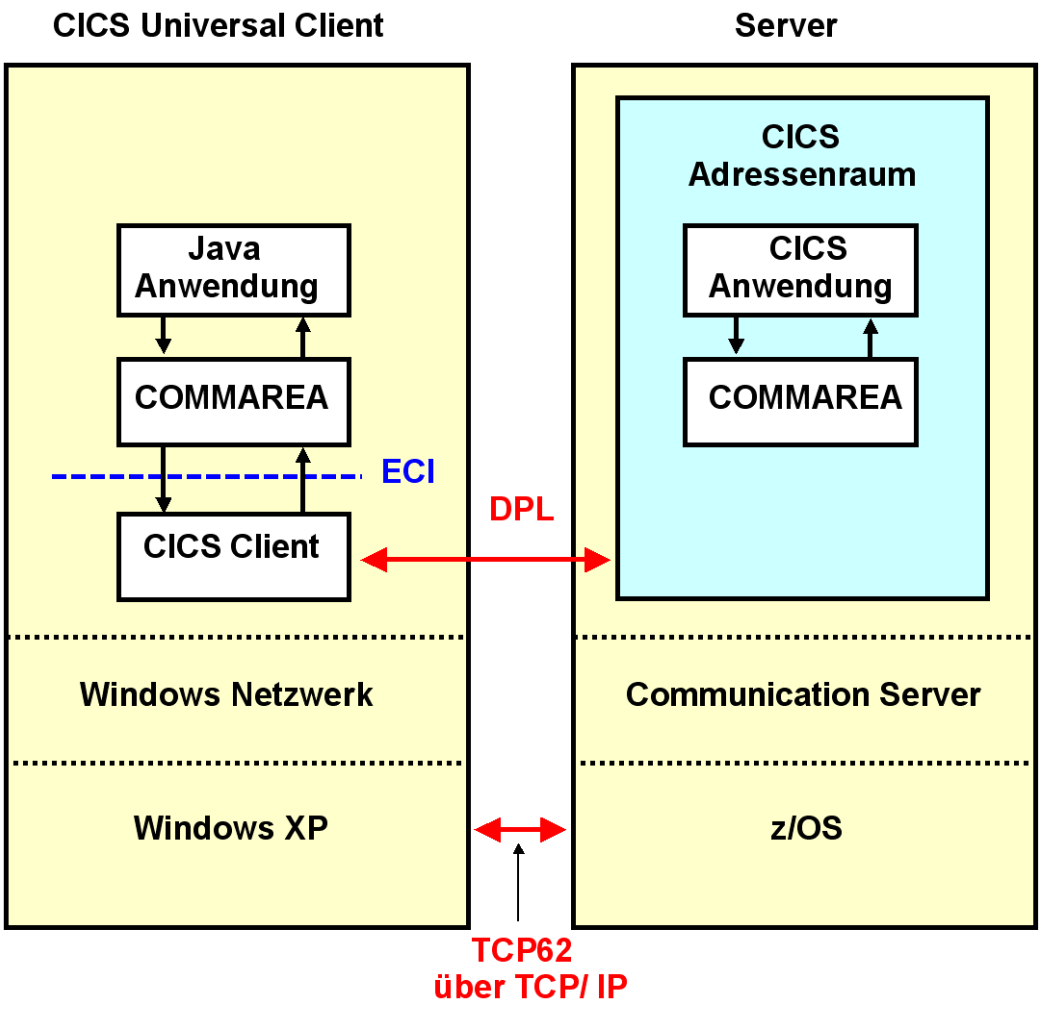


Abb. 18.3.6
CICS Universal Client

Ein CICS Universal Client ist eine echte CICS Anwendung, die mit anderen CICS Anwendungen über Distributed Program Link (DPL, einem RPC ähnlichen Mechanismus des CICS Transaktionsservers) verkehrt.

Eine Java Client Anwendung kann über die ECI Schnittstelle auf den CICS Client zugreifen. Dies ermöglicht einen direkten COMMAREA Datenaustausch zwischen Klienten und Server. Die Beschränkungen des BMS/3270 Datenprotokolls (z.B. keine Scroll Bar) werden damit umgangen.

18.3.6 CICS Transaction Gateway

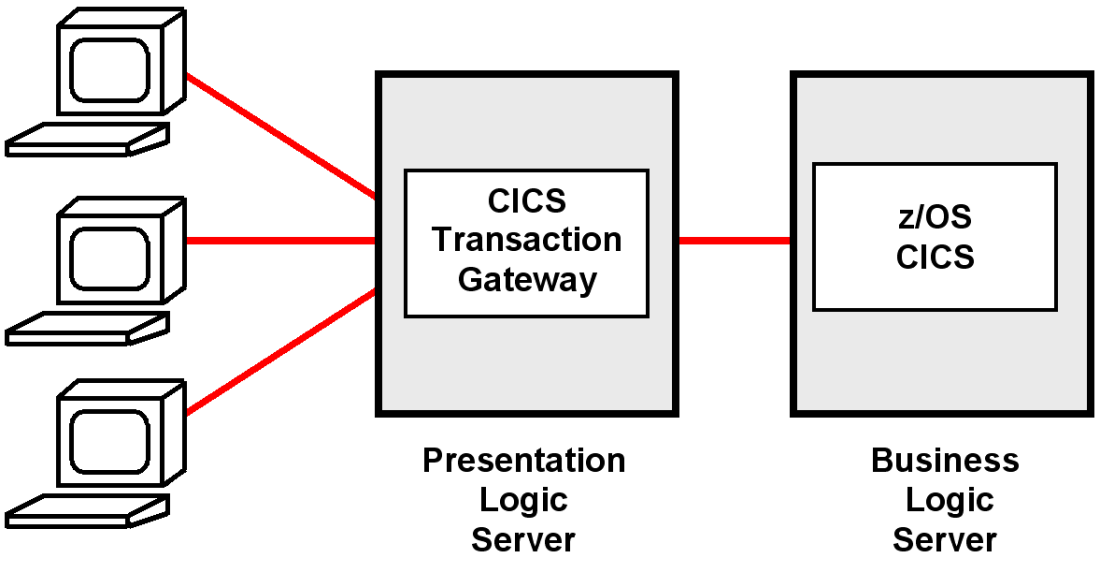


Abb. 18.3.7
Server seitige Präsentationslogik

Der CICS Client hat den Nachteil, dass bei einer Installation mit 10 000 Klienten der CICS Universal Client und die entsprechende ECI Anwendung 10 000 mal administriert und gewartet werden muss.

Deswegen wird man in vielen Fällen statt dessen einen zentralen Presentation Logic Server einsetzen. Hierfür existiert ein entsprechendes Software Product: das CICS Transaction Gateway (CTG).

Das CICS Transaction Gateway ist als EJB implementiert, und läuft normalerweise auf einem WebSphere Application Server. Dieser wiederum läuft entweder auf dem z/OS Rechner unter Unix System Services, oder alternativ auf einem getrennten (distributed) Windows oder Linux Server.

18.3.7 CICS Klienten Anbindung

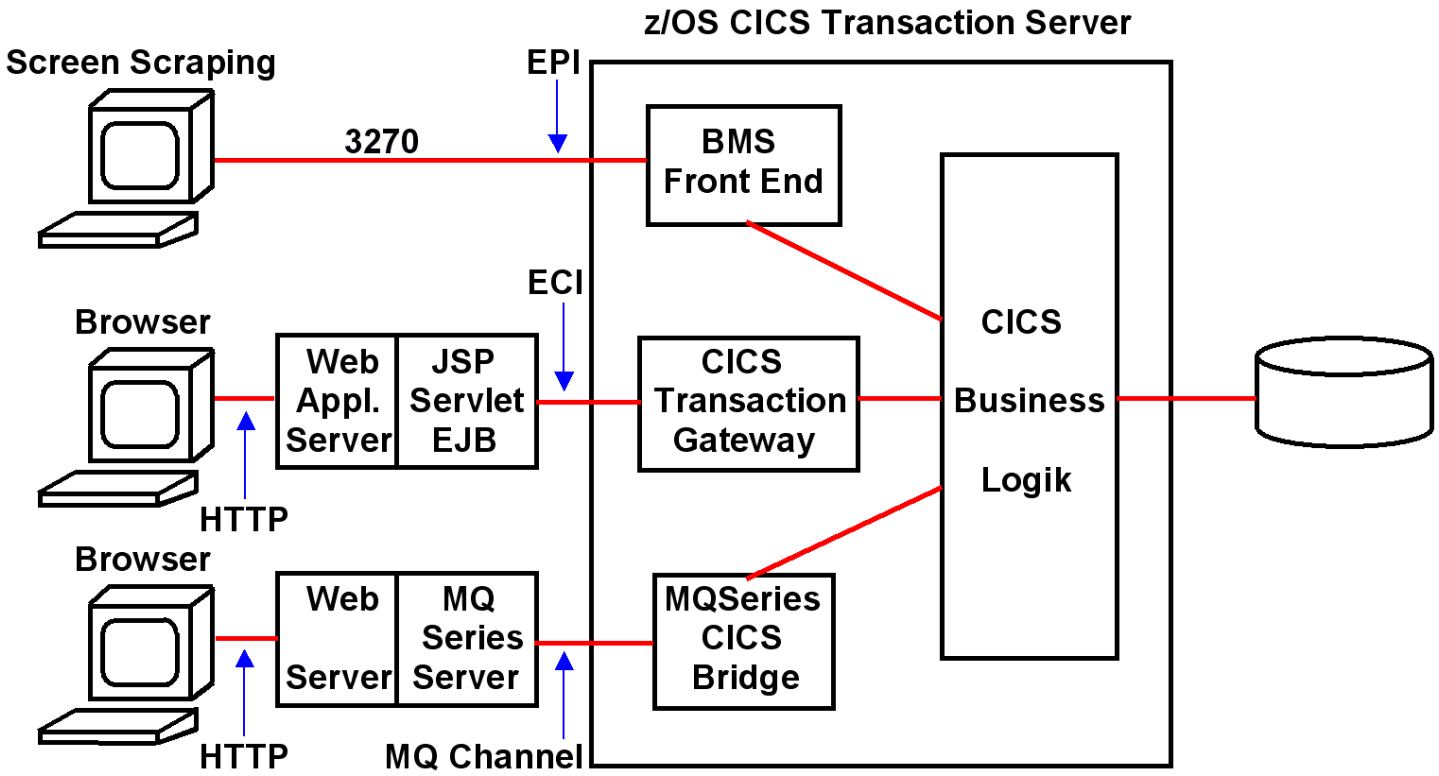


Abb. 18.3.8
Drei populäre Anbindung von CICS Klienten

Das 3270 Protokoll mit der EPI Schnittstelle gestattet nur eine Zeilen-orientierte Ausgabe oder Screen Scraping. Die BMS Maps werden weiter verwendet. Keine Änderung der Information, die auf dem Bildschirm wiedergegeben wird. Die Darstellung der Information kann geändert werden.

Bei einer uneingeschränkten grafischen Ausgabe wird die Presentation Service Komponente von CICS (BMS) wird nicht genutzt. In dem hier gezeigten Beispiel erfolgt ein direkter Zugriff auf COMMAREA. Die Presentation Service Komponente von CICS (BMS) wird nicht genutzt.

18.3.8 Was wird heute eingesetzt ?

Es existieren Dutzende von Alternativen, um CICS Anwendungen in das Internet zu integrieren, für die CICS über die entsprechende Unterstützung verfügt. Eine ganze Reihe von Lösungen werden von IBM geliefert, viele andere von unabhängigen Herstellern (third party vendors). Ein Beispiel eines unabhängigen Herstellers ist die Firma Attachmate.

Drei Alternativen sind besonders bedeutend; sie sind in Abb. 18.3.8 gezeigt.

Besucht man ein heutiges Unternehmen, so stellt man fest, dass häufig für die gleiche CICS Anwendung mehrere unterschiedliche Präsentations-Logiken verfügbar sind. Eine populäre Möglichkeit ist die asynchrone pseudoconversationale Verarbeitung unter Benutzung von MQSeries und der CICS MQSeries Bridge.

Grafische Oberflächen – teilweise Browser basiert – sind sehr populär. Viele – nicht alle – werden in Java mit Servlets und JSPs implementiert. Hier spielt das CICS Transaction Gateway (CTG) eine bedeutende Rolle.

Ein überraschend großer Anteil der CICS Anwender benutzen nach wie vor die 3270 Oberfläche. Der Grund ist, die zeilenorientierte 3270 Oberfläche ist produktiver als eine grafische Oberfläche. Sie braucht allerdings mehr Einarbeitungszeit.

18.4 JCA

18.4.1 Enterprise Information System

„Enterprise Information System“ (EIS) ist ein Begriff, der vor allem von JEE Standards verwendet wird. Ein EIS ist existierende Middleware, auf die mit Hilfe von Java Komponenten, besonders EJBs, zugegriffen wird. EIS Beispiele sind Transaktionsmonitore wie SAP R/3, CICS, IMS/DC, Tuxedo, Datenbanksysteme wie DB2, IMS/DB, Oracle, Adabas oder Message based Queuing Systeme wie MQSeries.

Die meisten EIS sind reinrassige Middleware; der Hersteller eines EIS nimmt an, dass ein Benutzer seine eigenen Anwendungen für das EIS schreibt. Manche Hersteller von EIS liefern umfangreiche betriebswirtschaftliche Anwendungen mit ihrem EIS aus. Diese EIS werden als ERP (Enterprise Resource Planning) Systeme bezeichnet. Ein Beispiel ist SAP R/3. Oracle vertreibt neben seiner relationalen Datenbank ebenfalls mehrere ERP Systeme, darunter Oracle Enterprise One, oder als direkten SAP Konkurrenten die Oracle E-Business Suite.

Kommentar zum Namen SAP R/3: Auch die Firma SAP liebt es, die Namen zu ändern. Bis Dezember 2003 wurde das Produkt unter dem Namen SAP R/3 geführt, bis 2007 unter dem Namen mySAP ERP. Die derzeitige Version heißt R/3 Enterprise oder SAP ERP.

Es existieren viele Anwendungen auf unabhängigen Anwendungsservern und viele unterschiedliche Enterprise Information Systems (z.B. CICS, SAP, IMS, MQSeries, DB2, ...). Aufgrund der großen Anzahl von unterschiedlichen Enterprise Information Systemen (EIS) ist die Lösung der Integrationsfrage sehr komplex. Um aus einer Java Anwendung heraus auf Informationen eines EIS zuzugreifen, war bisher notwendig, eine anwendungsspezifische Verbindung zu programmieren. Die Folge war ein hoher Entwicklungsaufwand, welcher mit der Anzahl unterschiedlicher EIS-Systeme und Anwendungsserver (z.B. WebLogic oder WebSphere) wuchs, da für jeden Anwendungsserver eine Verbindung zu jedem EIS hergestellt werden muss. Bei m Anwendungsservern und n EIS-Systemen bedeutet dies einen Aufwand von "m * n".

Durch die Java Connector Architektur (JCA) soll dieser Aufwand reduziert werden.

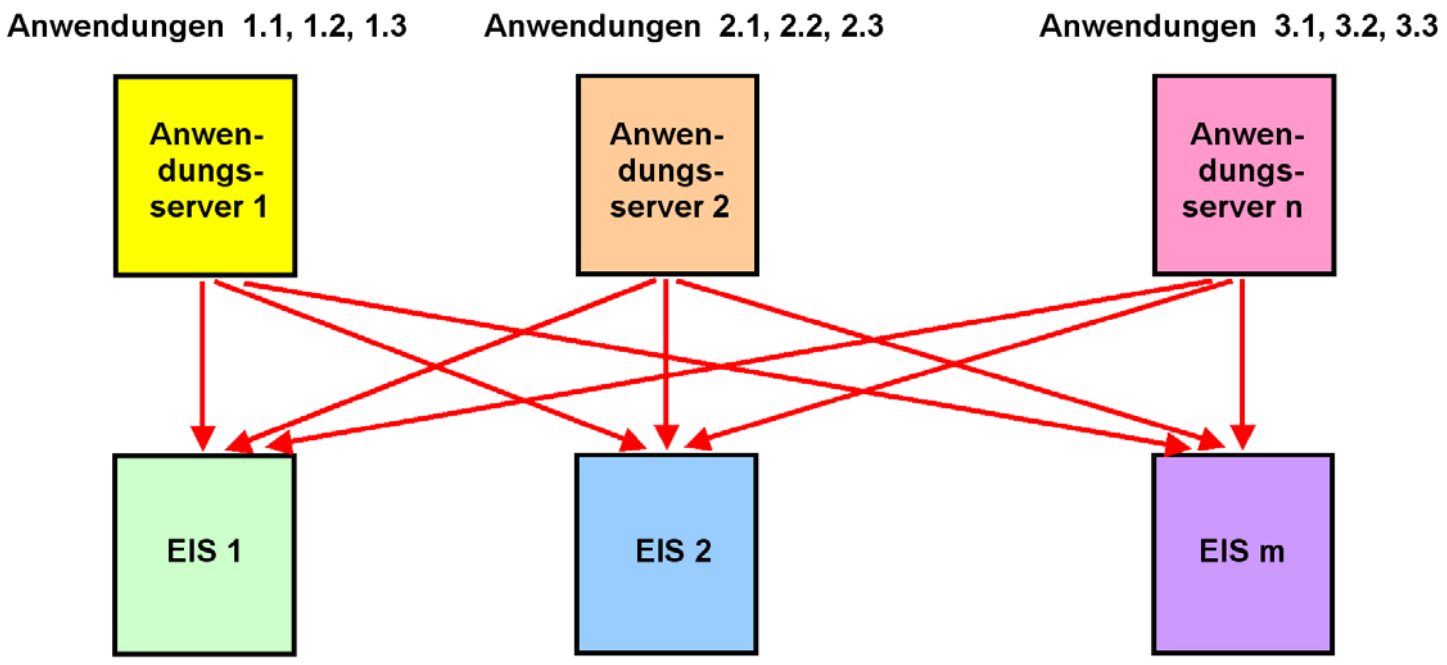


Abb. 18.4.1
Verbindungsvielfalt zwischen Anwendungsservern und EIS Komponenten

Jeder Pfeil stellt eine personalisierte Verbindung zwischen einem Anwendungsserver und einem Enterprise Information System (EIS) dar.

18.4.2 Konnektoren

Bei den Implementierungen von JEE Anwendungen spricht man von der Präsentationslogik (Frontend), welches typischerweise mit einem Web Application Server realisiert wird, und der Businesslogik (Backend, Beispiele: Auftragseingang, Finanzbuchhaltung), wofür häufig vorhandene Systeme wie CICS oder DB2 eingesetzt werden. Zusätzlich werden Erweiterungen als EJBs implementiert.

In vielen Fällen werden 20 % des Projektaufwandes für die Neuentwicklung des Frontends und 80% für dessen Integration in das vorhandene Backend aufgewendet.

Konnektoren sind in beliebigen Sprachen geschriebene Software Komponenten, welche eine Schnittstelle zu existierenden Enterprise Information (Legacy) Systemen bilden. Die meisten Konnektoren werden heute als EJBs erstellt und eingesetzt. Beispielsweise sind folgende EJB Konnektoren für den IBM WebSphere Application Server für die folgende Middleware Software verfügbar:

- **JDBC, SQLJ**
- **DB2**
- **Oracle**
- **Adabas**
- **CICS**
- **IMS**
- **MQSeries**
- **SAP R/3**
- **Lotus Domino**

} **Enterprise Information Systeme**

18.4.3 JEE Connector Architecture

Die Java EE Connector Architecture (JCA) ist eine Software-Architektur und Programmier-Schnittstelle (API) zur Integration von heterogenen Anwendungen und EIS in die JEE Platform.

JCA ist eine Standard Architektur für die Integration von existierenden Business Logik Komponenten .

- **ERP Systeme, z.B. SAP R/3**
- **Mainframe Transaktions- Monitore, z.B. CICS, IMS**
- **Non- Java Legacy Anwendungen**
- **Datenbank Systeme**

} **vom JCA Standard als Enterprise Information Systeme (EIS) bezeichnet**

Wichtigste Bestandteile der JEE Connector Architecture (JCA) sind:

- **JCA Konnektoren, als Resource Adapter (RA) bezeichnet**
- **Common Client Interface (CCI)**

A Resource Adapter is a system level software library that is used by an application server or client to connect to a Resource Manager. A Resource Adapter is typically specific to a Resource Manager. It is used within the address space of the client using it. An example of a resource adapter is the JDBC driver to connect to relational databases.

Die Common Client Interface (CCI) ist eine universelle API für die Interaktion mit beliebigen Resource Adaptoren. Sie ist weitgehend unabhängig von den spezifischen Eigenschaften des über einen JCA Connector verbundenen EIS.

Weiterhin spezifiziert der JCA Standard eine System Level Programming Interface (Service Provider Interfaces, SPI)). Hiermit werden Dienste wie connection pooling, Resource Adapter Deployment und Packaging definiert. Darüber hinaus enthält die JCA noch eine API für lokale Transaktions- Demarkationen. Für CICS existieren:

- **JCA ECI Resource Adapter (für einen COMMAREA Zugriff),**
- **JCA EPI resource adapter.**

Das CICS Transaction Gateway (CTG) ist ein JCA Standard konformer JCA Adapter. Er enthält ECI Resource Adapter, EPI Resource Adapter und CCI, sowie weitere Zugriffsmechanismen außerhalb der JCA.

18.4.4 Unterschiedliche Connector Arten

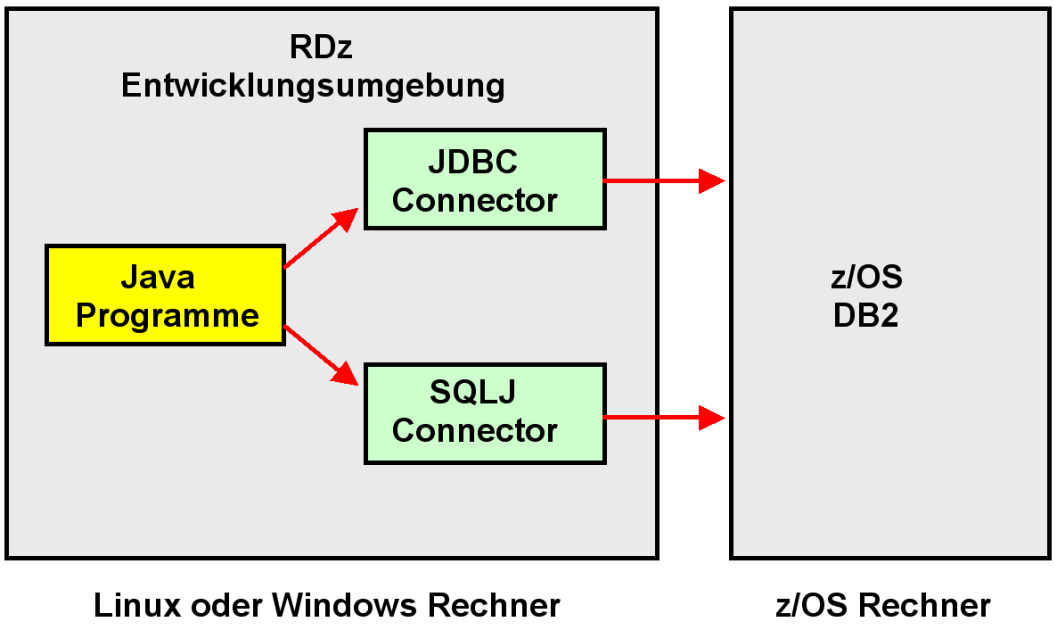


Abb. 18.4.2
JDBC und SQLJ Konnektoren

Es existieren viele unterschiedliche Konnektor Arten. Beispielsweise sind für den DB2 Zugriff JDBC und SQLJ Konnektoren vorhanden. JDBC ist von ODBC abgeleitet, und implementiert einen dynamischen Datenbankzugriff. SQLJ implementiert statische Datenbankzugriffe.

DB2Connect ist ein für die DB2 API optimierter Connector für in beliebige Programmiersprachen implementierte Klienten. Die Java Version implementiert den JCA Standard. Das derzeitige DB2Connect Software Paket beinhaltet mehrere DB2 Konnektoren, spezifisch auch JDBC und SQLJ.

18.4.5 JCA Struktur

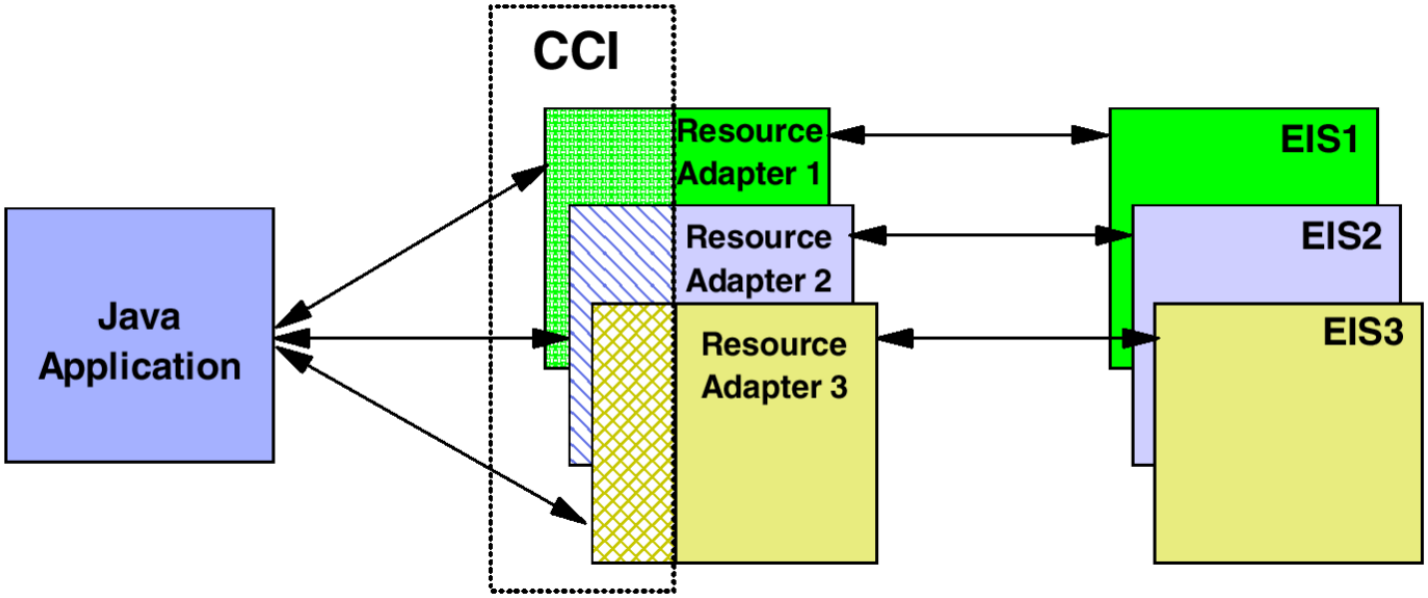


Abb. 18.4.3
Komponenten der JEE Connector Architecture

Der JCA Standard benutzt die folgenden Begriffe:

- **CCI Common Client Interface**
- **EIS Enterprise Information System**

CCI und Resource Adapter sind als Java Klassen implementiert. Sie können von einer Java Anwendung alleinstehend (ohne Benutzung eines JEE Servers) benutzt werden (non-managed environment).

Üblich ist es, CCI und Resource Adapter als Elemente eines JEE Web Application Servers einzusetzen (managed environment). Hierbei kann der Web Application Server das Management von Verbindungen, Transaktionen und Sicherheit direkt übernehmen. Die CCI Entwicklung kann in ein Entwicklungswerkzeug wie Eclipse integriert werden.

18.4.6 Common Client Interface

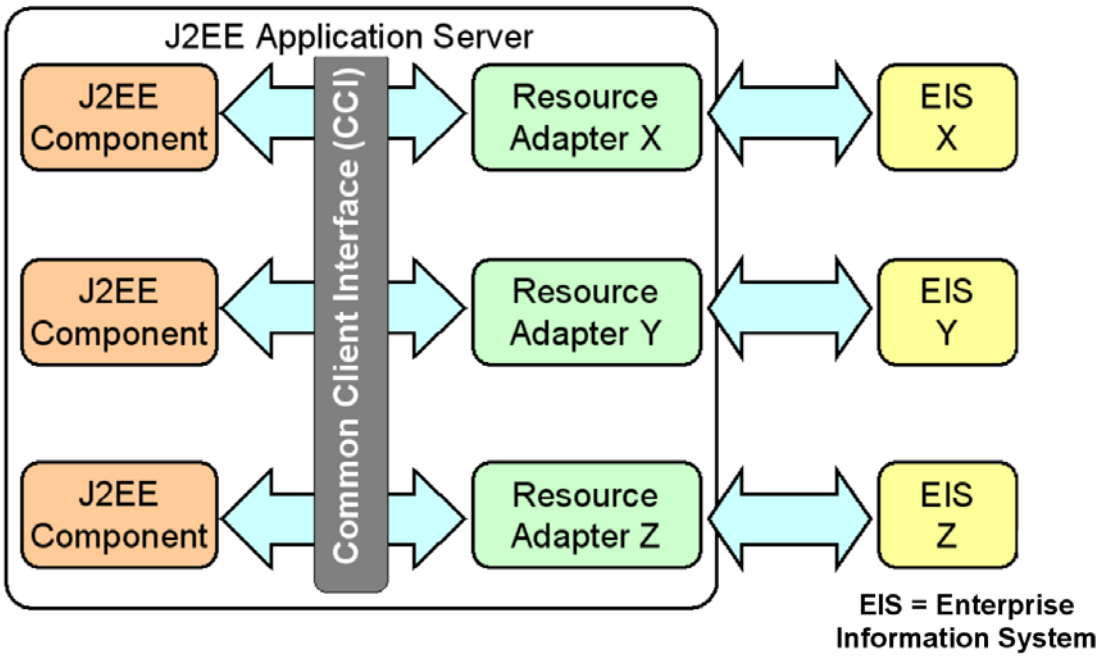


Abb. 18.4.4
Identische Common Client Interface für unterschiedliche Java Anwendungen

Die identische JCA Common Client Interface (CCI) wird von vielen verschiedenen Java Anwendungen verwendet. Die CCI bietet Schnittstellen zu vielen verschiedenen Resource Adapter (RA) an. Ein RA ist eine Schnittstelle zu einer bestimmten Business Logic Middleware. In der JCA Standard Dokumentation wird die Business Logik Middleware als Enterprise Information System (EIS) bezeichnet.

Resource Adapter werden von den Herstellern von Standard Middleware Komponenten zur Verfügung gestellt, z. B. von Oracle, IBM, SAP, und vielen anderen.

18.4.7 CICS als Enterprise Information System

CICS hat eine wachsende Bedeutung. Es werden jedes Jahr mehr CICS Transaktionen als im Vorjahr ausgeführt, und dieser Trend existiert seit 35 Jahren. Die Benutzer tätigen jedes Jahr bedeutende Investitionen in CICS, und auch in MQSeries.

Der WebSphere Application Server hat einen offensichtlichen Wert als Platform für neue JEE-Anwendungen. Die Kombination von CICS, WebSphere MQ und WebSphere Application Server bildet ein sehr leistungsfähiges Ökosystem für die Ausführung von unternehmenskritischen Anwendungen.

Heute ist es sehr üblich, WAS (WebSphere Application Server) Anwendungen zu schreiben, die CICS-Transaktionen auslösen. Das ist ein Umfeld, in dem CICS als Partner mit WAS agiert, wobei die Transaktion von WAS ausgelöst wird.

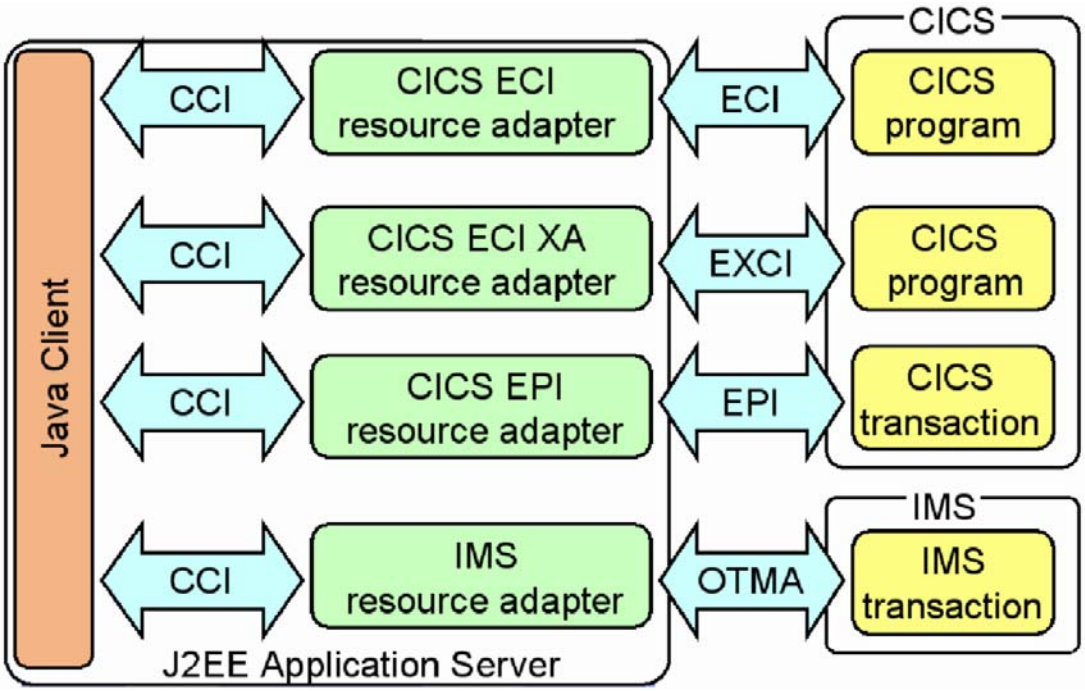


Abb. 18.4.5
Identische CCI für unterschiedliche Resource Adapter

Als Beispiel sind 4 verschiedene Resource Adaptoren (RA) dargestellt, die für IBM Mainframes verfügbar sind. Der CICS ECI und EPI Resource Adapter wurde bereits diskutiert.

Der EXCI Resource Adapter ist eine spezielle Version des ECI. Der ECI Adapter wird verwendet, wenn der JCA Resource Adapter auf einem entfernten System installiert ist. Der EXCI Adapter wird verwendet, wenn der JCA Resource Adapter und CICS auf dem gleichen Mainframe installiert sind. In der Regel läuft der EXCI Resource Adapter unter USS.

Die OTMA Adapter hat die gleiche Funktion für IMS/DC, welche die ECI und EXCI Adapter für CICS haben.

Beim Wechsel zu einer anderen Ressource-Adapter sind aufgrund der CCI keine Änderungen im Java-Client-Code erforderlich.

18.4.8 JCA Benutzung des CICS Transaction Gateways

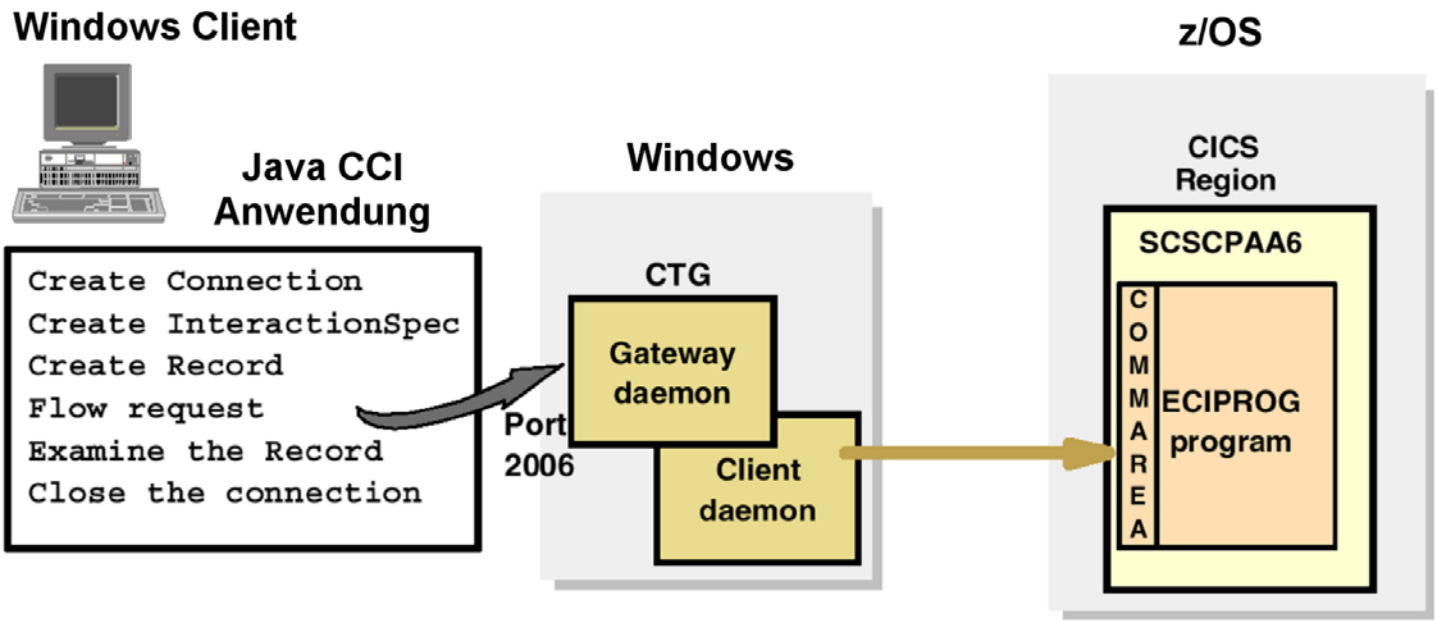


Abb. 18.4.6
CCI Skelett für das CICS Transaction Gateway

Abb. 18.4.6 zeigt CCI Aufrufe einer Java Client Anwendung und ihre CCI Connection zum CICS Transaction Gateway.

18.4.9 CCI Beispiel für die CICS ECI

```
import javax.resource.cci.*;
import com.ibm.connector2.cics.*;

//create a connection
ECIConnectionSpec connSpec = new ECIConnectionSpec();
connSpec.setUserName("CICSUSER");
Connection conn = connfac.getConnection(connSpec);

//Create an interaction
Interaction inter = conn.createInteraction();
ECIInteractionSpec iSpec = new ECIInteractionSpec();
iSpec.setFunctionName("CICSPROG");

//Create a record
MyRecord record = new MyRecord("Data to send to CICS");

//Perform the interaction with CICS
inter.execute(iSpec, record, record);

//Display contents of output record
System.out.println(record.getContents());
```

Abb. 18.4.7
CCI Code für eine Verbindung zu CICS

Gezeigt ist das Skelett einer Java Client Anwendung, welche die CCI verwendet.

18.4.10 CICS Transaction Gateway

Abb. 18.4.8 zeigt einen z/OS-Server, auf dem ein WebSphere Web Application Server mit einer Servlet-Engine und einer EJB-Engine für ein CTG, sowie ein CICS Server für die Geschäftslogik installiert ist.

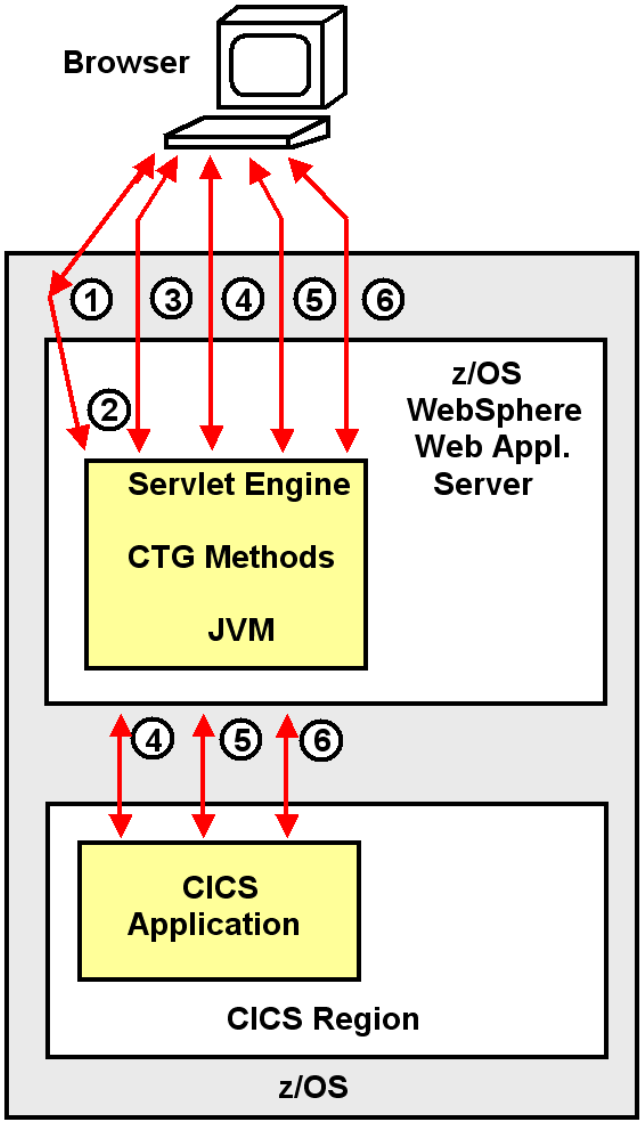


Abb. 18.4.8
Schritte für die Benutzung des CICS Transaction Gateway

Schritte 1 und2: **Der Browser stellt eine Verbindung zu dem Websphere Web Application Server her.**

Schritte 3 bis 6 **Jeder weitere Schritt benutzt Servlets für die Präsentations-Logik und das CICS Transaction Gateway für den Aufruf der CICS Business Logik.**

18.4.11 Versionen des CTG

Das CICS Transaction Gateway (CTG) wird häufig als der Java Enterprise Edition (JEE)-Konnektor für CICS bezeichnet, da es den Zugang zu CICS innerhalb einer JEE Umgebung wie dem IBM WebSphere Application Server (WAS) unterstützt. Allerdings unterstützt das CTG Application Programming Interfaces (API), Platformen und Topologien, die über JEE hinausgehen. Das CTG offeriert CICS-Konnektivität zu C, C++, COBOL und Microsoft DotNet-Anwendungen sowie zu Java Standard Edition (JSE)-Anwendungen.

CTG ist in zwei Varianten erhältlich, um verschiedenen Anforderungen und Einsatzszenarien unterzubringen.

- **Als eine Variante bietet die Multiplatform (distributed) CTG Version einen Multi-User-Zugriff auf CICS aus einer UNIX-, Linux- oder Windows-Umgebung.**
- **Die CTG for z/OS Version offeriert CICS-Konnektivität mit der Skalierbarkeit und den Performance-Vorteilen beim Betrieb in enger Proximity mit dem CICS Transaction Server.**

Für die distributed und die z/OS Version des CTG existieren drei unterschiedliche Topologien, mit denen ein Zugriff auf einen z/OS CICS Transaction Server erfolgen kann.

- **Topologie 1: WebSphere Application Server und CICS Transaction Gateway sind beide auf einer distributed (non-zSeries) Plattform installiert.**
- **Topologie 2: WebSphere Application Server und CICS Transaction Gateway sind beide auf einem z/OS Server installiert.**
- **Topologie 3: WebSphere Application Server und CICS Transaction Gateway sind beide unter zLinux auf dem gleichen physischen Rechner wie z/OS und CICS installiert.**

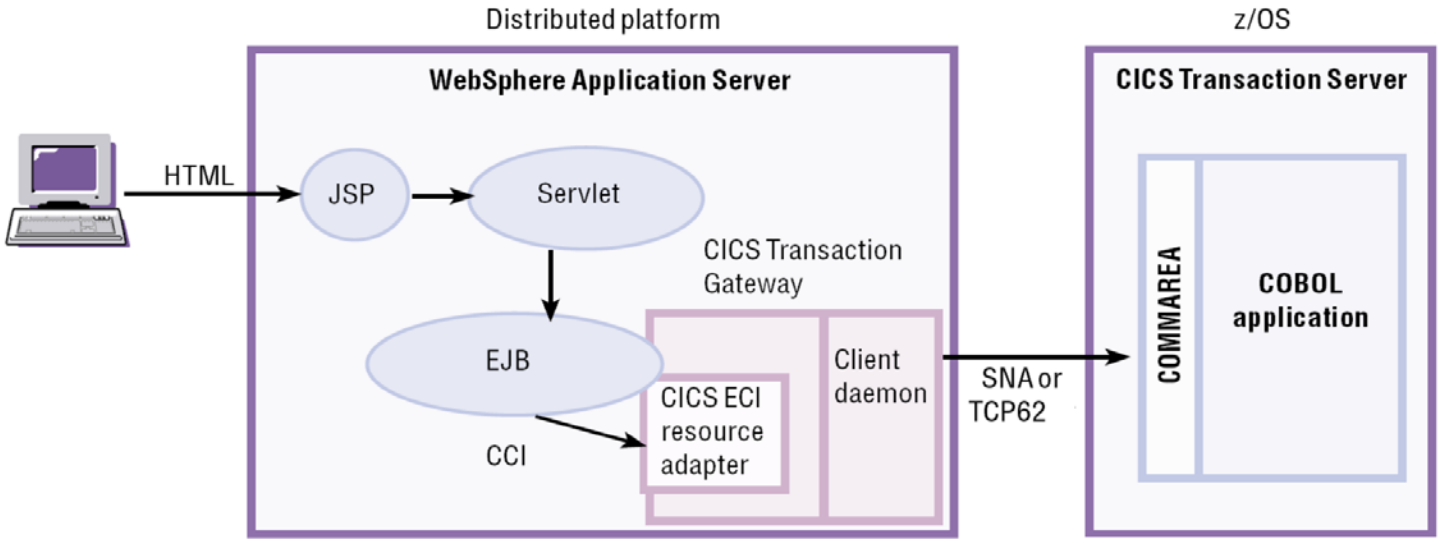


Abb. 18.4.9
CICS Transaction Gateway Topologie 1

In dieser Topologie (3-Tier Konfiguation) sind der WebSphere Application Server und das CICS Transaction Gateway auf einer distributed Plattform installiert, z.B. Windows oderr UNIX. Sowohl ECI als auch EPI (nicht gezeigt) Resource Adapter können benutzt werden. Der CICS Universal Client ist Bestandteil des CTG.

Abb. 18.4.9 zeigt eine Enterprise Java Beans (EJB) Anwendung, die den ECI Resource Adapter benutzt um auf eine COMMAREA-basierte CICS COBOL Anwendung zuzugreifen.

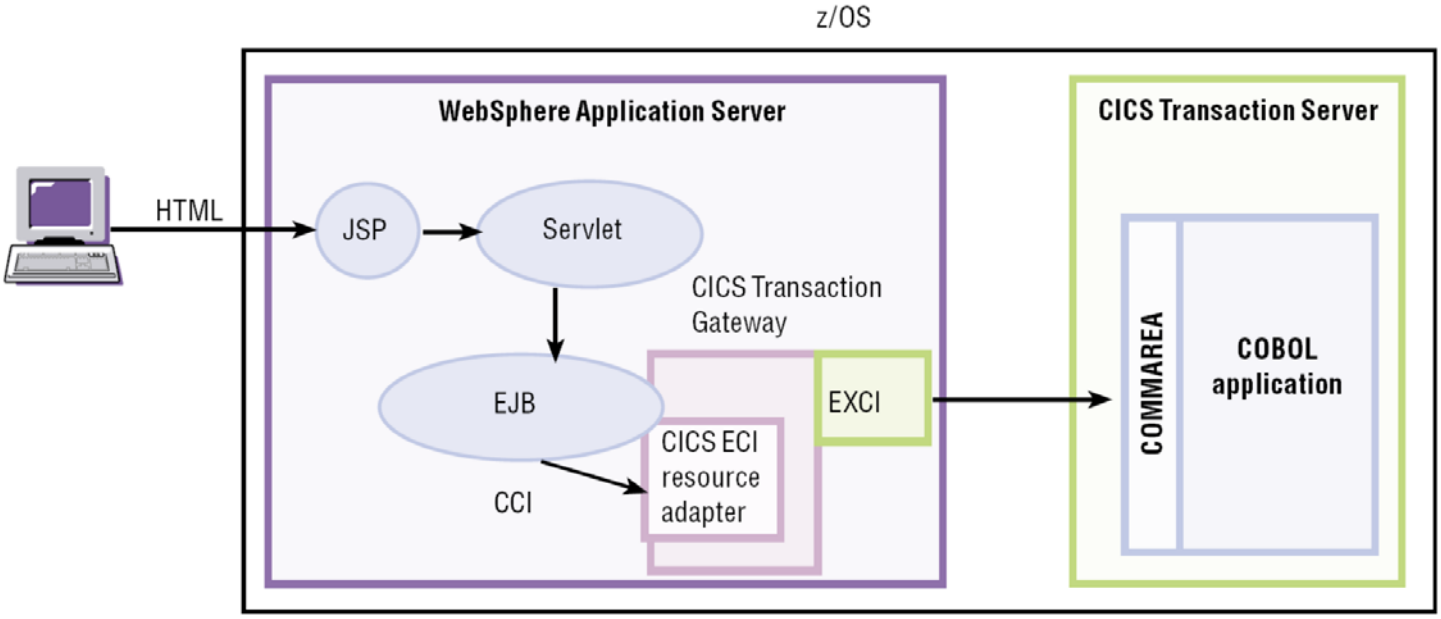


Abb. 18.4.10
CICS Transaction Gateway Topologie 2

In der in Abb. 18.4.10 dargestellten 2-Tier Topologie ist der WebSphere Application Server unter dem gleichen z/OS wie CICS installiert, allerdings in getrennten Address Spaces. Das CTG ist Teil des WAS. Es werden nur CICS ECI Resource Adapters unterstützt. Unter z/OS ermöglicht dies eine direkte Cross-Memory EXCI Verbindung zwischen dem WebSphere Application Server und CICS.

EXCI ist eine Komponente des CICS Transaction Server für z/OS. EXCI ermöglicht es einem z/OS non-CICS Region (Address Space) auf Programme innerhalb einer CICS Transaction Server Region zuzugreifen.

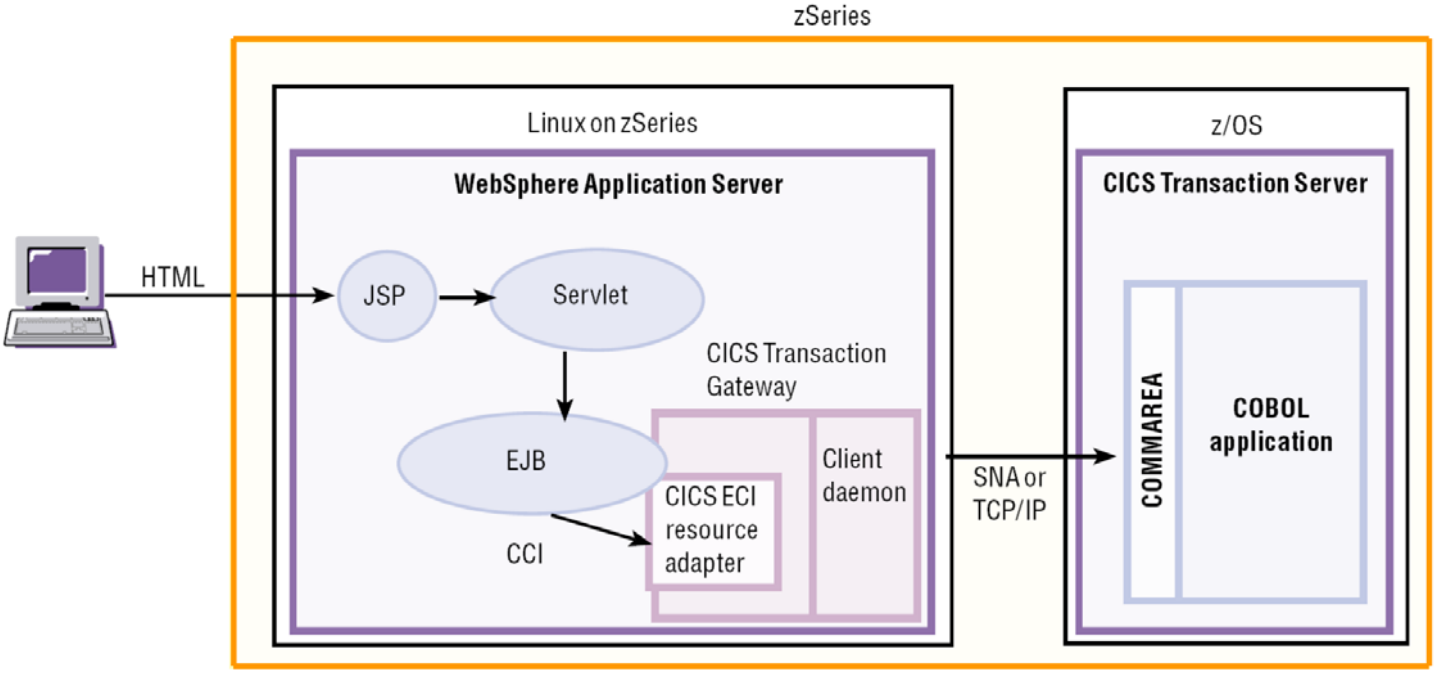


Abb. 18.4.11
CICS Transaction Gateway Topologie 3

Eine weitere Alternative ist es, den WebSphere Application Server auf zLinux zu installieren, und zwar auf dem gleichen System z Rechner auf dem auch z/OS CICS installiert ist, siehe Abb. 18.4.11. Im Vergleich zu Topologie 1 (Abb. 18.4.9) erlauben Hipersockets die Nutzung einer Reihe weiterer Funktionen.

18.4.12 Java Kryptografie Architektur

Beim Öffnen einer z/OS Installation für das Internet hat Sicherheit eine stark wachsende Bedeutung. Kryptografie ist ein leistungsfähiges Werkzeug um die Internet-Sicherheit zu verbessern.

Die Java Kryptografie Architektur definiert die Abstimmung zwischen der Anwendung, dem Connector und dem Anwendungsserver, auf dem die Anwendung bereitgestellt wird.

Die System z Hardware verfügt über Kryptographie Prozessoren und Beschleuniger, welche die Kryptographie Verarbeitungszeit deutlich verbessern.

Security ist ein Problem in allen großen Installationen. System z und z/OS haben deutlich weitergehende Sicherheitseigenschaften als alle anderen Platformen. Die Topologien 2 und 3 in Abb. 18.4.10 und 18.4.11 sind deshalb unter Sicherheitsgesichtspunkten vorteilhaft gegenüber Topologie 1 in Abb. 18.4.9 .

Anmerkung: JCA bedeutet alternativ JEE Connector Architecture und Java Cryptography Architecture.

18.5 Weiterführende Information

Vielleicht interessiert Sie hier ein Video über das CICS Transaction Gateway. Es wurde im IBM Entwicklungslabor in Hursley, Südengland aufgenommen. Das Hursley Labor ist spektakulär in einem früheren englischen Adelssitz untergebracht:

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

Eine Art der Anwendungsentwicklung:

http://www.youtube.com/watch?v=Wc39pSHHKIg

(1) Eine kurze Einführung in SNA und APPC finden Sie unter

http://www.cedix.de/VorlesMirror/Band2/SNA.pdf

http://www.cedix.de/VorlesMirror/Band2/APPC.pdf

Eine Beschreibungen der SNA Enterprise Extender finden Sie unter

http://www.cedix.de/VorlesMirror/Band2/SNA02.pdf

Ein guter Überblick ist enthalten in:
IBM Redbook: Introduction to the New Mainframe: Networking, August 2006, SG24-6772-00,
http://www.redbooks.ibm.com/abstracts/sg246772.html

und
IBM Redbook: System z Connectivity Handbook, Februar 2013, SG24-544-13,
http://www.redbooks.ibm.com/redbooks/pdfs/sg245444.pdf

Eine ausführliche Beschreibung von SNA ist enthalten in dem Lehrbuch:
R.C. Cypser: Communications for Cooperating Systems, OSI, SNA, and TCP/IP. Addison Wesley, 1991. ISBN 0-201-50775-7

Eine ausführliche HATS Demo finden Sie unter

http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/galadriel.pdf

19. Java Transaction Processing

19.1 EJB Transaktionseigenschaften

19.1.1 Locking Funktion für eine Transaktion

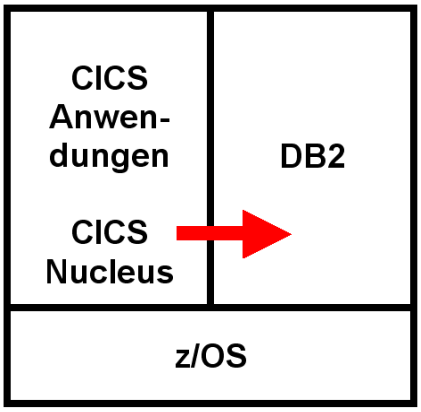


Abb. 19.1.1
Wird nur eine einzige Datenbank benutzt, übernimmt diese das Locking

In einer einfachen Konfiguration greift eine transaktionale Anwendung auf eine einzige Data Bank (zum Beispiel DB2) zu, die auf dem gleichen Rechner untergebracht ist. Mehrere unabhängige Anwendungen können auf diese Datenbank zur gleichen Zeit zugreifen. DB2 enthält hierfür eine eigene Transaktionsverarbeitungsfunktion, die z.B. auch bei Stored Procedures eingesetzt wird. Diese ordnet den einzelnen Transaktionen je eine Signatur zu, überwacht die transaktionalen Zugriffe und ist für den Transaktionsabschluss und die Fehlerbehandlung zuständig.

Bei der Benutzung von VSAM zur Datenspeicherung übernimmt der CICS File Manager die Locking Funktion.

19.1.2 Transaction Manager und Transaction Service

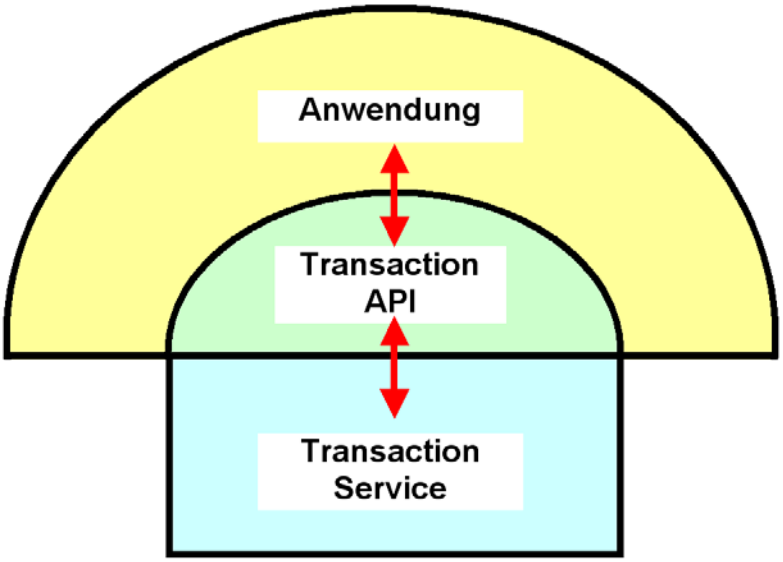


Abb. 19.1.2
Erläuterung der Begriffe

Formal unterscheidet man bei dem JEE Standard zwischen der Java Transaction API und einer Transaction Service Implementation. Die Java Transaction API ist Teil der JEE Spezifikation und ist die Schnittstelle zum Transaction Service.

Der Transaction Service implementiert einen Transaktionsmonitor vergleichbar zu CICS oder Tuxedo und ist die mit Abstand umfangreichste Komponente eines Web Application Servers. Der Transaction Service ist nicht Teil der JEE Spezifikation und ist eine proprietäre Implementierung des Herstellers des Web Application Servers. So verwendete früher der WebLogic Application Server den hauseigenen Tuxedo Transaktionsmonitor. Distributed WebSphere verwendete hier die distributed CICS Version.

19.1.3 Distributed Transaction System

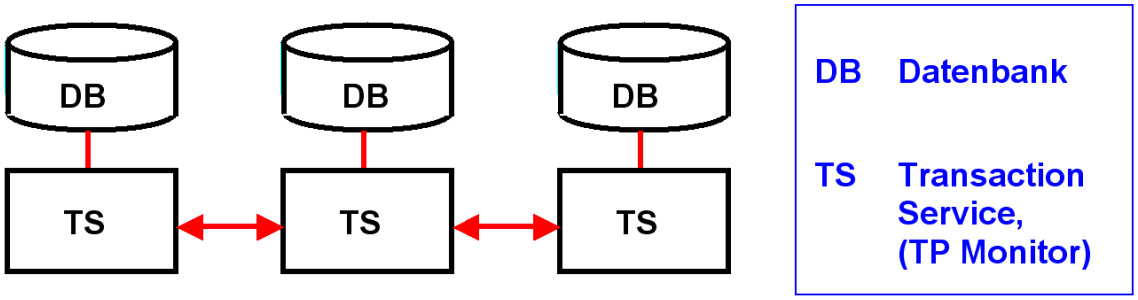


Abb. 19.1.3
Mehrere Transaktionsmanager implementieren eine globale Transaktion

In einem distributed Transaction System wird eine globale Transaktion auf mehreren Rechnern ausgeführt. Dabei können Datenbanken (DB) auf mehreren Rechnern mit ihrem eigenen lokalen Transaktionsmonitoren (hier als Transaction Service, TS, bezeichnet) an der Ausführung der Transaktion beteiligt sein. Eine globale Transaktion wird meistens mittels des 2-Phase-Commit Protokolls implementiert, siehe Band 1, Abschnitt 7.4.14.

Enterprise Java Beans und JEE Server unterstützen neben lokalen Transaktionen, bei denen nur auf eine lokale Datenbank zugegriffen wird, auch globale Transaktionen in verteilten Umgebungen mit mehreren Datenbanken.

19.1.4 Distributed Transaction Processing (DTP)

Wenn Information in einer Datenbank geändert wird, muss sichergestellt sein, dass die Änderung korrekt erfolgt. Nehmen wir an, Sie unterhalten bei Ihrer Bank zwei Konten, ein laufendes Konto (Checking Account) und ein Anlagekonto (Savings Account), und Sie wollen Geld von Ihrem Anlagekonto auf Ihr laufendes Konto überweisen. Wird die Transaktion nur teilweise ausgeführt, indem das Geld von Ihrem Anlagekonto abgebucht, aber nicht Ihrem laufenden Konto gutgeschrieben wird, wird Sie das nicht zufrieden stellen.

Weiteres Beispiel: Sie wollen einen Flug von San Francisco nach Paris buchen, aber ein direkter Flug ist nicht verfügbar. Wenn Sie keine Buchungsbestätigung sowohl für die Verbindung San Francisco – Chicago, als auch eine weitere Buchungsbestätigung Chicago – Paris erhalten, werden Sie diesen Flug nach Paris nicht buchen (commit). Sie werden ein "roll back" für diesen Flug nach Paris durchführen, weil eine Buchungsbestätigung nur für einen Teil des Fluges für Sie nicht annehmbar ist.

In beiden Beispielen sind mehrere kleinere Teiltransaktionen erforderlich um eine gesamte (globale) Transaktion durchzuführen. Wenn ein Problem mit einer Teiltransaktion besteht, wollen Sie ein Commit der gesamten Transaktion (Geldtransfer, oder diesen spezifischen Flug nach Paris buchen) nicht durchführen. Statt dessen wollen Sie ein Rollback für jede Teiltransaktion durchführen. Für eine erfolgreiche Geldüberweisung oder Parisreise-Buchung wollen Sie sicherstellen, dass alle Teiltransaktionen koordiniert und gesteuert werden um die globale Transaktion erfolgreich durchzuführen.

Für eine derartige koordinierte globale Transaktionsverarbeitung enthält die JEE Platform ein "distributed Transaction Processing Environment". Dieses besteht aus einem JEE Application Server, JEE Application Komponenten sowie einem JEE Connector Architecture (JCA) Resource Adapter. Hiermit können Resourcen über physische Rechnergrenzen hinweg transparent und koordiniert abgeändert werden.

In einer einfachen Konfiguration greift eine EJB Anwendung auf eine einzelne Data Bank (zum Beispiel DB2) zu. In einer komplexeren Situation greift eine EJB Transaktion auf mehrere "Resources" (Datenbanken, Files, Queues) zu, die von mehreren Transaction Processing Monitoren verwaltet werden, die evtl. wiederum auf getrennten Rechnern laufen und von unabhängigen Herstellern stammen. Eine derartige globale Transaktion erfordert ein "Distributed Transaction Processing" (DTP) Environment. Ein Beispiel ist die verteilte Flat Transaction und das 2-Phase Commit Protocol.

19.1.5 Two Phase Commit

Angenommen, ein CICS Programm will parallel eine Änderung einer CICS eigenen VSAM Datei und einer externen DB2 Datenbank durchführen. Das CICS Programm wird hierzu ein EXEC CICS Syncpoint Kommando ausführen. Dies löst eine 2-Phase Commit Operation aus. Siehe hierzu Band 1, Abschnitt 7.4.14.

Existierende Transaction Monitore wie CICS, IMS und DB2 verfügen hierzu über eigene Commit Coordinator Komponentenen. Das Update der VSAM Datei und der DB2 Datenbank wird erst entgültig, wenn der CICS Commit Coordinator Phase 2 des 2-Phase Commit Protokolls erfolgreich abgeschlossen hat.

Neu entwickelte Resource Manager Produkte unter z/OS (z.B. WebSphere Enterprise Java Beans) verwenden z/OS Resource Recovery Services (RRS) an Stelle einer nicht vorhandenen eigenen Two-phase Commit Protocol Komponente.

19.1.6 Resource Recovery Services

In der Vergangenheit haben z/OS Resource Manager wie IMS, CICS und DB2 ihre eigenen Commit Coordinator Funktion für die Two-Phase Commit Verarbeitung entwickelt. Hierbei übernimmt einer der beteiligten Resource Manager die Rolle des Commit Coordinators. Dies erforderte die Entwicklung von getrenntem Commit Coordinator Code jeweils für IMS, CICS und DB2.

Mit der wachsenden Anzahl verfügbarer z/OS Resource Manager entstand das Bedürfnis für einen generischen Commit Coordinator (von IBM als Sync Point Manager bezeichnet). Diese Rolle übernimmt eine neue z/OS Komponente, die Resource Recovery Services (RRS). RRS ist ein 2-Phase Commit Coordinator Service, der beliebigen z/OS Subsystemen zur Verfügung steht, so auch für WebSphere unter z/OS.

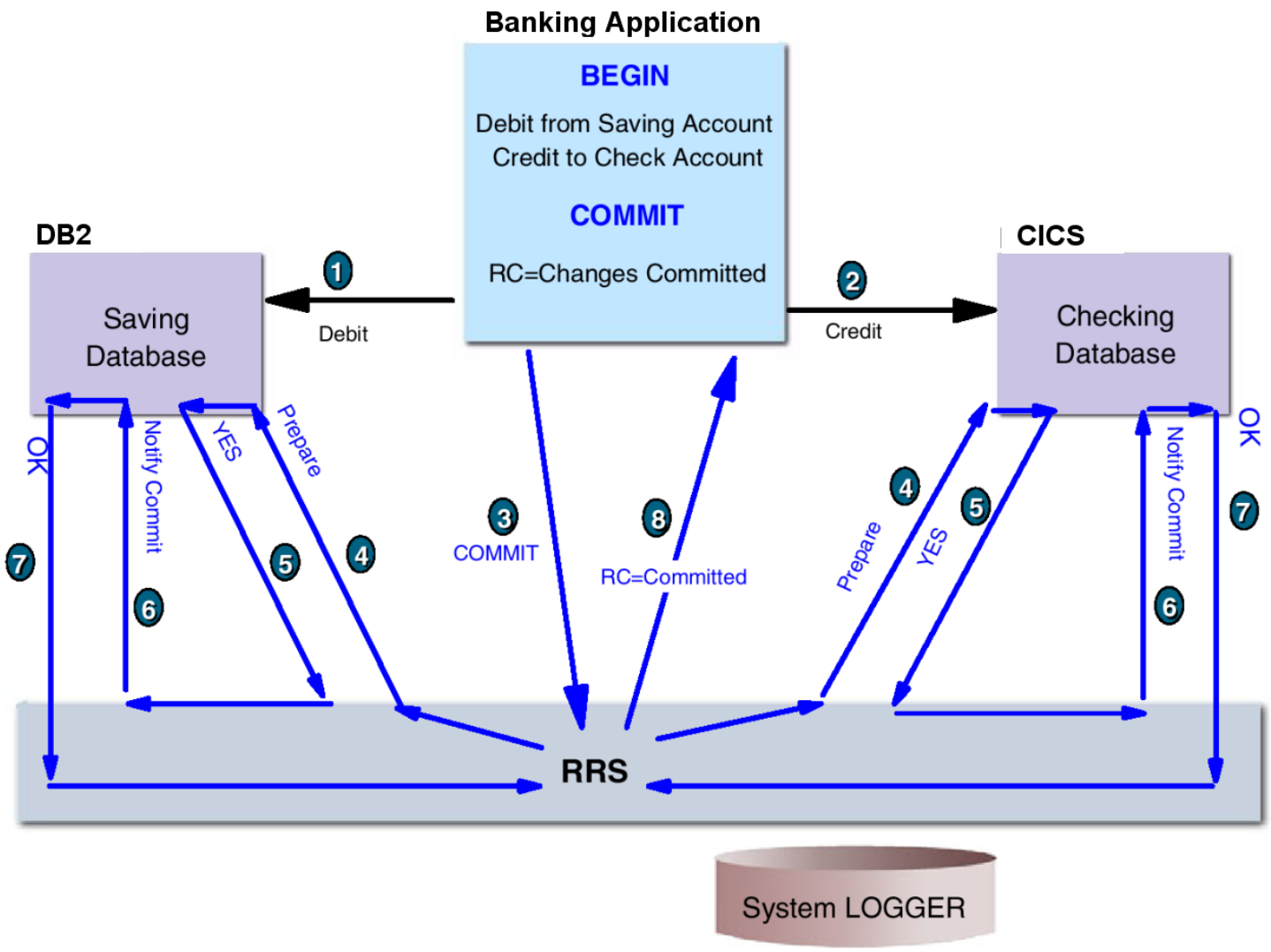


Abb. 19.1.4
RRS Funktionsablauf

Heute können Resource Manager wie IMS oder CICS die RRS Komponente an Stelle ihrer eigen Commit Coordinator Funktionen verwenden. Neue Entwicklungen wie WebSphere unter z/OS müssen RRS verwenden.

Das CICS Transaction Gateway kooperiert mit dem

- **Java Transaction Manager einer beliebigen JEE Platform (u.a. WebSphere), den**
- **Resource Recovery Services (RRS) unter z/OS, und**
- **CICS,**

um konsistente Änderungen für CICS und andere geschützte Resourcen durchzuführen.

Vom Standpunkt von CICS aus gesehen, agiert RRS als ein "external coordinator" für die Änderung oder Recovery von Resourcen. Vermutlich wird sich RRS in der Zukunft zu einer zentralen z/OS Komponente für die Transaktionssteuerung entwickeln.

Literature: http://www.redbooks.ibm.com/redbooks/pdfs/sg246980.pdf

19.1.7 X/Open

Das X/Open Konsortium wurde 1984 von mehreren UNIX Systems Herstellung mit der Zielsetzung gegründet, offene Standards auf dem Gebiet der Informationstechnologie zu etablieren. Spezifisch wurde eine einheitlich Unix Spezifikation für die Interoperability von Anwendungen und die Portierung von Software angestrebt.

Es existieren eine ganze Reihe von unterschiedlichen X/Open Standards. Sie gelten für alle Sprachen, einschließlich Java.

X/Open XA (kurz XA) ist ein von X/Open spezifizierter Informatik-Standard für Distributed Transaction Processing (die Abarbeitung von Transaktionen, die über mehrere Systeme verteilt sind). Mithilfe des XA-Standards können verschiedene „Ressource Manager“ (wie Datenbanken, Transaktionsmonitore, Applikationsserver, Messaging Systeme) innerhalb einer Transaktion angesprochen werden, unter Bewahrung der ACID-Eigenschaften von Transaktionen.

Die XA Spezifikation beschreibt, was ein Ressource Manager leisten muss um gemäß XA verteilte Transaktionen zu ermöglichen. Damit wird die Vorgangsweise und Schnittstelle zwischen einem globalen „Transaktion Manager“ und den lokalen Resource Managern festgelegt. Ein Transaktionsmanager ist somit etwas anderes als ein Transaktionsmonitor (Transaktionsserver). Ressource Manager, die den XA Standard erfüllen nennt man „XA compliant“. Der XA-Standard basiert auf dem 2-Phasen-Commit-Protokoll.

Die Java Implementierung des X/Open XA definiert die Schnittstelle (Interfaces und Exception-Classes), über die Java-Programme mit Transaktionsmanagern kommunizieren können. Transaktionsmanager ihrerseits implementieren üblicherweise die Java Transaction API (JTA) Programmierschnittstelle, Teil des EJB Standards. JTA stellt die Standardschnittstelle für XA fähige Transaktionsserver dar.

19.1.8 Globale Transaktion und der 2-Phase Commit Process

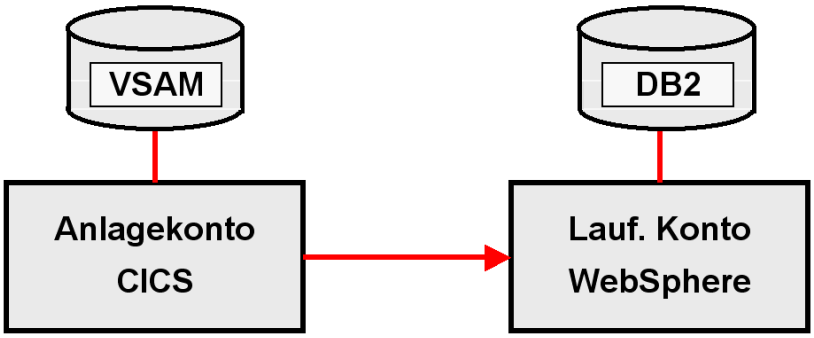


Abb. 19.1.5
Beispiel für eine globale Transaktion

Ein JEE-konformer Anwendungsserver (wie z.B. der WebSphere Application Server) benutzt den XA Transaction Manager für die Kommunikation zwischen den Anwendungs-Komponenten (z.B. Java Servlets oder Enterprise Java Beans) und externen Resource Managern (z.B. CICS oder DB2). Die Koordination einer globalen Transaktion erfolgt typischerweise über Java Connector Architecture (JCA) konforme Resource Adapter, wie z.B. dem CICS Transaction Gateway.

Wenn ein Transaction Manager (TM) eine globale Transaction mit mehr als einem Resource Manager koordiniert, verwendet die Commit Coordinator Funktion des Transaction Managers das Two-Phase Commit Protokoll.

In dem vorher erwähnten Bank-Beispiel unterhielten Sie bei Ihrer Bank zwei Konten, ein laufendes Konto (Checking Account) und ein Anlagekonto (Savings Account). Sie wollten Geld von Ihrem Anlagekonto (Savings Account) auf Ihr laufendes Konto (Checking Account) überweisen. Angenommen, Ihr Anlagekonto wird von einer CICS Anwendung bearbeitet, deren Daten in einem VSAM Data Set gespeichert sind, und Ihr laufendes Konto wir von einer z/OS WebSphere EJB Anwendung bearbeitet, deren Daten in einer DB2 Datenbank gespeichert sind. In diesem Fall würde die Commit Coordinator Funktion des WebSphere Transaction Managers die Änderungen zwischen VSAM und DB2 koordinieren.

19.1.9 Transaktionale Konnektoren

Konnektoren sind spezielle EJBs, welche eine Verbindung zu einem Backend System wie CICS oder DB2 herstellen. WebSphere für z/OS unterstützt zwei Arten von Konnektoren: non-transactional und transaktional. Die Connector Verarbeitung variiert je nach Konfiguration für jede installierte Instanz eines Connectors.

Die Transaktion Policy der EJB bestimmt, ob die Verarbeitung im Rahmen einer globalen Transaktion ausgeführt wird, oder nicht. Wenn ja, ist der Connector für die Steuerung der globalen Transaktionsverarbeitung zuständig. Für transaktionale Connectoren wird die Art der Transaktionsverarbeitung zu dem Zeitpunkt festgelegt, wenn eine Anforderung an das Ziel Enterprise Information System (EIS) gesendet wird.

z/OS WAS ist in der Lage, in Kooperation mit z.B. CICS eine distributed Transaction durchzuführen, die das 2-Phase Commit Protokoll erfordert. Diese benutzt den transaktionalen Typ von Connector. Letzterer ist konfiguriert, mit der Ressource Recovery-Service (RRS) Komponente von z/OS eine Two-Phase Commit Verarbeitung durchzuführen. RRS beinhaltet die Commit Coordinator Funktion für die 2-Phase Commit Verarbeitung; z/OS WAS beinhaltet keine eigene Commit Coordinator Funktion.

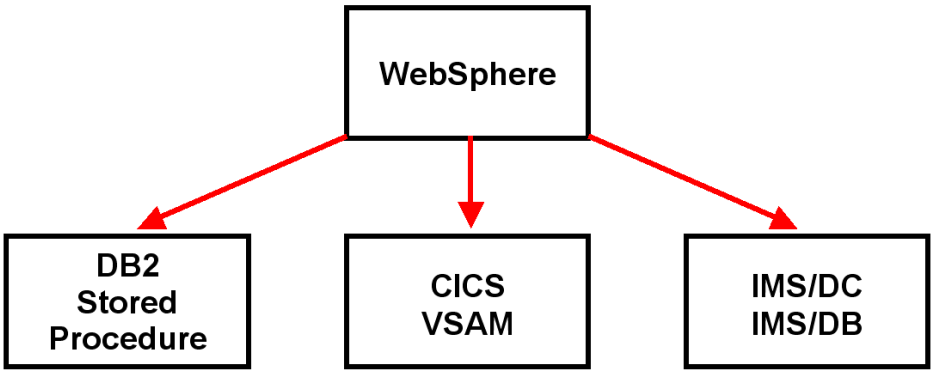


Abb. 19.1.6
WebSphere als Commit Koordinator

Beispiel: Ein WebSphere Anwendungsprogramm führt eine globale Transaktion durch, an der DB2, CICS und IMS/DC als Resource Manager beteiligt sind.

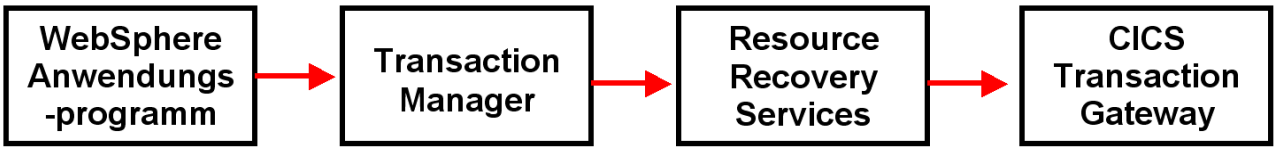


Abb. 19.1.7
Ablauf der Commit Koordination

Dies sind die Merkmale:

- **WebSphere übernimmt die Rolle des Transaktionsmanagers**
- **Die 2-Pase Commit Funktion wird durch Resource Recovery Services implementiert**
- **Das CICS Transaction Gateway ist ein JCA konformer Konnektor, dessen Konfiguration für eine globale Transaktion enabled wurde.**
- **Die JCA Konnektoren für DB2 und IMS/DC wurden ebenso enabled.**

19.1.10 X/Open Distributed Transaction Processing Model

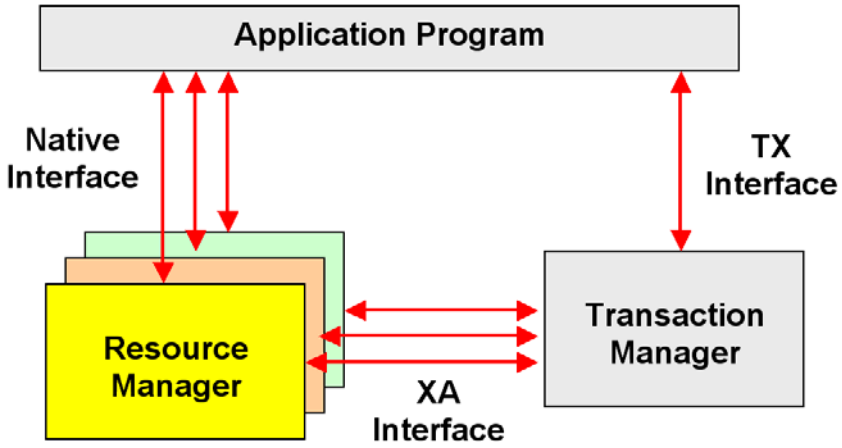


Abb. 19.1.8
X/Open Interfaces

Vorsicht: Als Transaktion Manager wird manchmal ein Transaktionsserver (Transaktionsmonitor) wie CICS bezeichnet. X/Open DTP bezeichnet statt dessen als Transaktion Manager eine Funktion, die einen 2-Phase Commit Coordinator enthält. Das X/Open Distributed Transaction Processing (DTP) Modell ist eine Spezifikation für das Management von Transaktionen, deren Operationen auf unterschiedlichen Rechnern oder auf Datenbanken unterschiedlicher Hersteller (z.B. Oracle, DB2) erfolgen.

DTP besteht aus 3 Kernkomponenten:

- **Einem Application Program. Dieses definiert Transaktionsgrenzen und spezifiziert die Aktionen, die eine Transaktion darstellen.**
- **Resource Managers (RM) sind Subsysteme oder Komponenten wie CICS, IMS, oder DB2. Diese verwalten Resourcen, die von den Transaktionen in Anspruch genommen werden, wie z.B. Datenbanken, Files oder Queues. Alternativ kann ein RM auch ein Data Access Komponente wie ODBC oder JDBC sein.**
- **Ein Transaction Manager (TM). Dieser markiert Transaktionen mit Idendifiern, überwacht den Fortschritt und ist zuständig für den erfolgreichen Abschluss (commit), oder alternativ für Fault Recovery (Rollback). Ein TM beinhaltet vor allem einen 2-Phase Commit Coordinator.**

RRS (Resource Recovery Services, Abschnitt 11.ist ein Beispiel einer XA Transaction Manager Implementierung.

19.1.11 X/Open Interfaces

Der X/Open Standard ist für beliebige Sprachen anwendbar, einschließlich Java.

Das Application Program, der Resource Manager (RM) und der Transaction Manager (TM) kommunizieren über drei spezifische Interfaces: Native, TX, and XA.

- **Über die Native interface sendet das Application Program Requests direkt an die Resource Managers. Diese Schnittstelle ist Resource Manager spezifisch; embedded SQL oder EXEC CICS sind Beispiele.**

- **Die TX Interface ist das Medium zwischen dem Application Program und dem Transaction Manager, zum Beispiel der Commit Coordinator Komponente von CICS. Die API verwendet TX Calls (über dieTX Interface), um den Transaction Manager zum Start, Commit oder Roll Back einer globalen Transaktion aufzufordern. Diese Schnittstelle ist Transaction Manager spezifisch.**

- **Die XA Interface ist das Medium zwischen den Resource Managern und dem Transaction Manager. Mit Hilfe von XA Calls fordert der Transaction Manager den Resource Manager zum Transaction Start, Commit, oder Roll Back auf. Der Transaction Manager ist auch für eine Recovery zuständig.**

Die X/Open XA Interface ist ein offener Standard um Änderungen für multiple Resources zu koordinieren, wobei die Integrität dieser Änderungen gewährleistet wird. Beispiele sind Resourcen, die eine persistente Speicherung beinhalten wie Datenbanken, Queues und File Systeme. Mehrere unterschiedliche Transaction Processing Monitore benutzen typischerweise die XA Interface für eine gegenseitige Kommunikation. Ein gleichzeitiger Zugriff auf mehrere Datenbanken ist damit möglich.

19.2 Enclaves

19.2.1 Language Environment

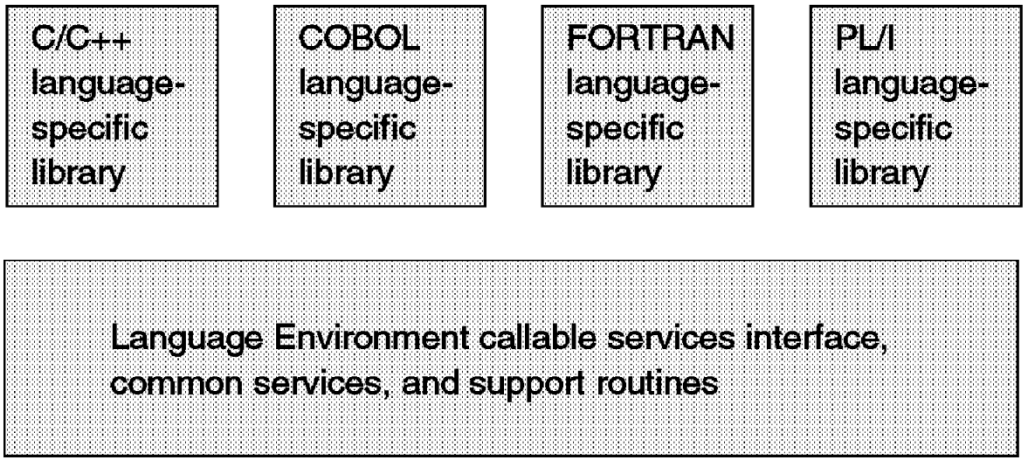


Abb. 19.2.1
LE Übersicht

Das z/OS „Language Environment“ (LE) Programm Management Modell bietet einen Rahmen, in dem der Code von Anwendungsprogrammen ausgeführt werden kann, die in verschiedenen Sprachen geschrieben wurden.

LE bietet eine gemeinsame Laufzeitumgebung für die z/OS Versionen bestimmter Hochsprachen (High Level Language) an, nämlich C/C++, COBOL, Fortran, PLI und Java. LE kann als eine Sammlung von Ressourcen, Bibliotheken und Betriebssystem-spezifischen Diensten und Schnittstellen betrachtet werden.

LE kombiniert häufig verwendete Laufzeit-Dienste, wie Routinen für

- **Mathematische Funktionen (z.B. transzendentale Funktionen wie Wurzel, arctan, x^{π}, ...)**
- **Laufzeit Message-Handling,**
- **Condition Handling,**
- **Storage-Management, und**
- **Datum und Uhrzeit.**

Das bedeutet z.B., dass es nur eine arctan Object Code Routine gibt, die von unterschiedlichen Programmiersprachen genutzt wird. LE stellt dafür eine Reihe von Schnittstellen zur Verfügung, die konsistent für alle unterstützten Programmiersprachen sind.

Language Environment besteht aus:

- **Basic-Routinen, die den Start und Stop von Programmen, Zuweisung von Speicherkapazitäten, sowie Handhabungsbedingungen unterstützen.**
- **Gemeinsame bibliothekarischen Dienstleistungen, wie z. B. mathematische Funktionen sowie Datum und Uhrzeit Dienste.**
- **Sprach-spezifische Teilen der Laufzeitbibliothek, da viele Sprach-spezifische Aufrufe von Language Environment Routinen unterstützt werden müssen. Dabei ist das Verhalten konsistent für alle unterstützten Sprachen.**
- **Einrichtungen für die Kommunikation zwischen Programmen, die in verschiedenen Sprachen geschrieben sind.**

POSIX Unterstützung ist in der Language Environment Grundausstattung sowie in der C/C++ spezifischen Library vorhanden.

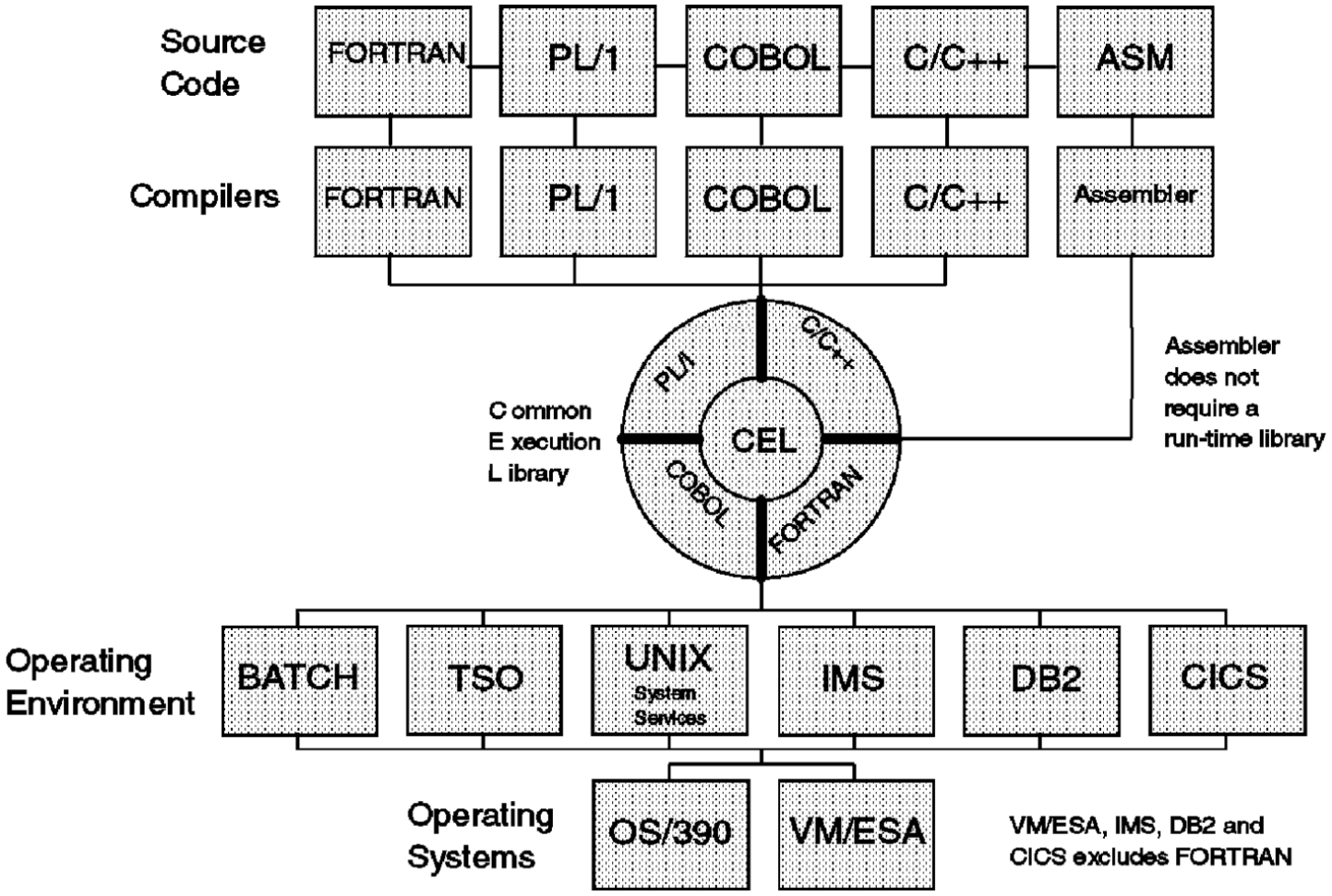


Abb. 19.2.2
Language Environment unter z/OS

Jede High Level Language hat ihre spezifischen Laufzeit und SYSLIB Bibliotheken. Zusätzlich benutzt sie zusammen mit anderen Sprachen eine gemeinsame Common Execution Library (CEL).

Die auf diese Weise hergestellten Load Modules können in verschiedenen Ausführungsumgebungen (Batch, CICS, DB2 usw.) unter z/OS oder z/VM ausgeführt werden.

19.2.2 Übersetzung eines neu entwickelten Programms

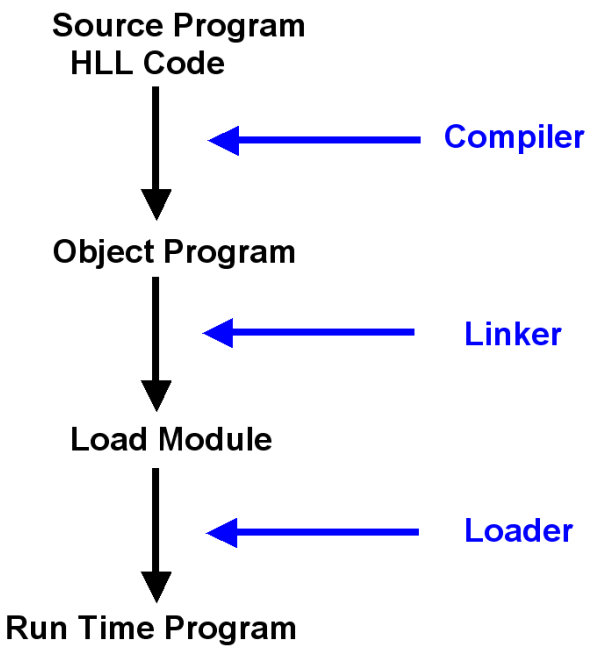


Abb. 19.2.3
Übersetzung eines Programms

Der Compiler übersetzt ein Source Programm (in Cobol, Java, C++, ...) in ein Object Programm (Binaries). Das Object Programm besteht aus Maschinenbefehlen.

Der Linker verkettet das Objekt Programm, gemeinsam mit einem oder mehreren zu einem anderen Zeitpunkt entstandenen Objekt Programm(en), sowie mit Routinen aus einer Programmbibliothek und aus der Common Execution Library zu einem ausführbaren Gesamtprogramm, dem Load Module. Die Common Execution Library besteht aus Object Modulen.

Alle Komponenten des Load Modules haben eindeutige Adressen; der zusammenhängende Adressenbereich beginnt typischerweise mit der Adresse 000...000 .

Der Loader lädt das Load Module vom Plattenspeicher in den Hauptspeicher, typischerweise auf eine Adresse, die nicht mit 000...000 beginnt. Hierzu werden alle Adressen durch den Loader verschoben (relocated).

19.2.3 Language Environment Definition

```
 DEFINE PROGRAM(PROG020) GROUP(PRAKT20)
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
 CEDA  DEFine PROGram( PROG020   )
   PROGram        : PROG020
   Group          : PRAKT20
   DEscription  ==>
   Language     ==> Le390              CObol | Assembler | Le370 | C | Pli
   RELoad       ==> No                 No | Yes
   RESident     ==> No                 No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : 00                 0-24 | Public
   CEdf         ==> Yes                Yes | No
   DAtalocation ==> Below              Below | Any
   EXECKey      ==> User               User | Cics
   COncurrency  ==> Quasirent          Quasirent | Threadsafe
  REMOTE ATTRIBUTES
   DYnamic      ==> No                 No | Yes
+  REMOTESystem ==>

                                                   SYSID=C001 APPLID=A06C001
  DEFINE SUCCESSFUL                           TIME:  00.00.00  DATE: 01.037
PF 1 HELP 2 COM 3 END             6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abb. 19.2.4
LE Definition für ein CICS Programm

Bei der Bearbeitung einer CICS Übungsaufgabe sieht man den in Abb. 19.2.4 dargestellten Bildschirm. Der Benutzer wird gebeten, eine Sprache für das CICS-Programm angeben. Die richtige Antwort ist Le390 (oder Le370). Le390 ist jedoch keine Sprache, es spezifiziert das z/OS Language Environment als die Ausführungsumgebung für eine CICS-Anwendung.

19.2.4 Parallele Verarbeitung von Transaktionen

Ein Hochleistungs-Transaktionssystem verarbeitet in jedem Augenblick gleichzeitig Hunderte oder Tausende von Transaktionen. Für die Verarbeitung gibt es zwei Alternativen:

- **1 Prozess pro Transaktion**
- **1 Thread pro Transaktion)**

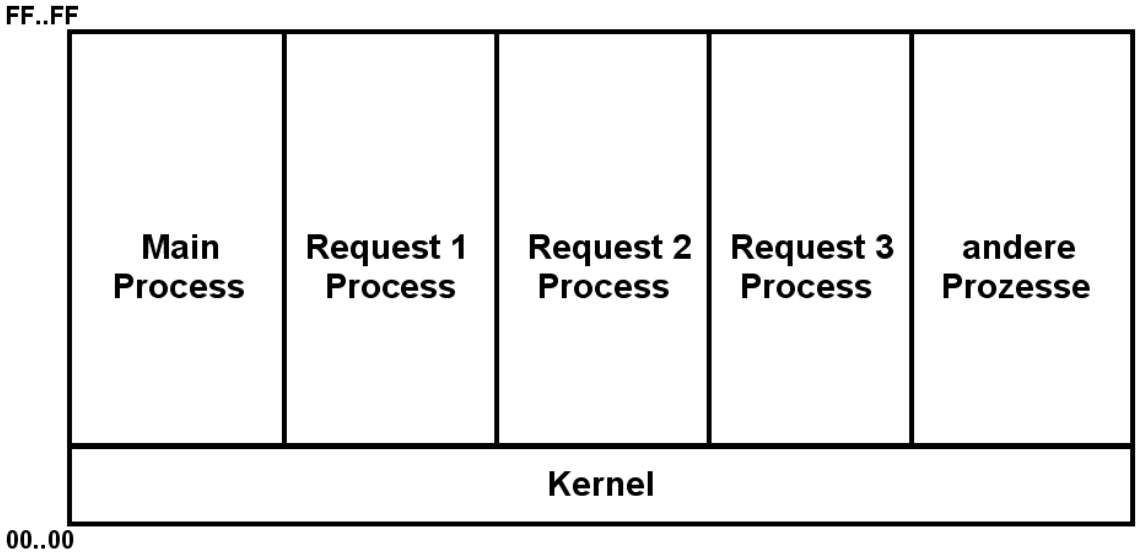


Abb. 19.2.5
Ansatz mit mehreren Prozessen

1 Prozess pro Transaktion bedeutet, dass jede Transaktion in einem eigenen virtuellen Adressenraum (z/OS Region) läuft. Vorteilhaft ist, dass die Isolation der Transaktionen untereinander (das i in ACID) optimal gewährleistet ist. Nachteilig ist der höhere Verarbeitungsaufwand im Vergleich zum Thread Ansatz.

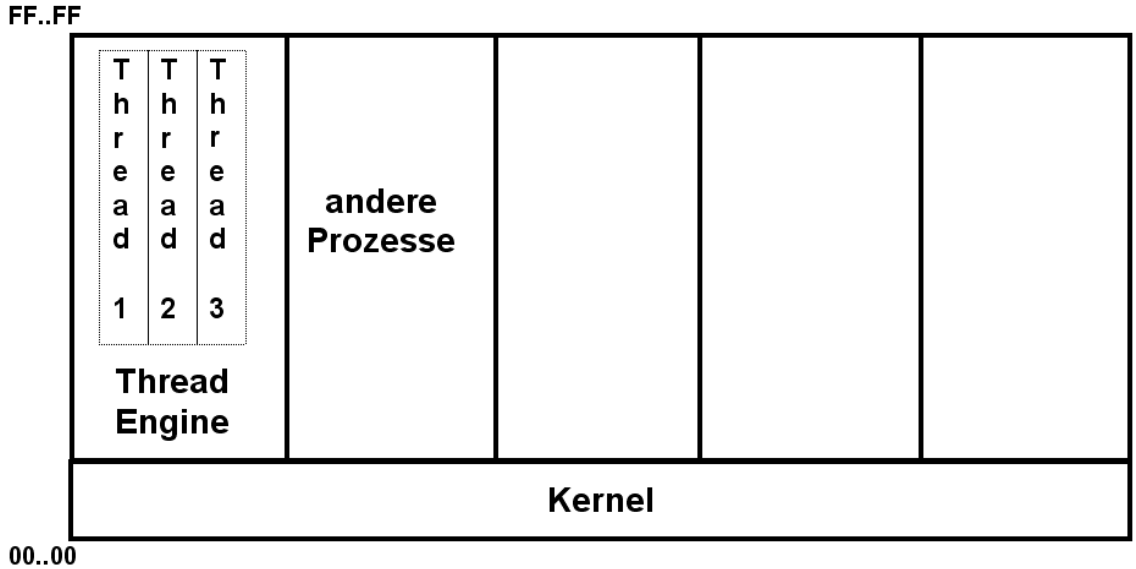


Abb. 19.2.6
Thread Ansatz

Threads haben den Nachteil, dass die Isolation der Threads gegeneinander gewährleistet werden muss.

Bei der z/OS Version von CICS laufen alle Anwendungen unter einem Thread ähnlichen Mechanismus in einem einzigen Adressenraum.

19.2.5 CICS Enclaves

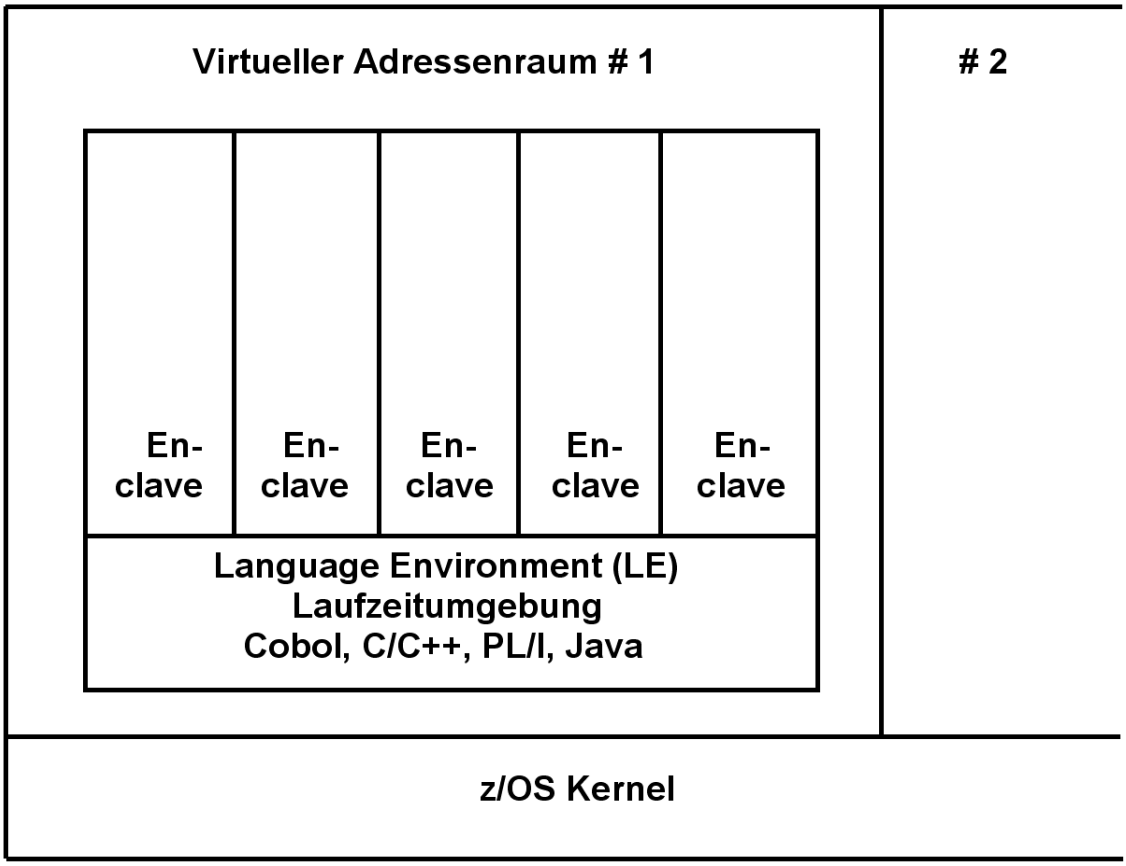


Abb. 19.2.7
Enclaves in der CICS Region

Die grundlegende CICS Konfiguration besteht aus einer einzigen z/OS Region, die den CICS Nucleus beinhaltet, sowie bis zu mehr als 1 000 gleichzeitig ausgeführten Transaktionen beherbergt.

Transaktionen müssen ACID Anforderungen erfüllen: Atomicity, Consistency, Isolation, Durability.

Die Isolation ist oft schwierig zu implementieren. CICS verwendet etwas ähnliches wie Threads, den „Enclave“ Mechanismus.

Enclaves sind ein Teil des z/OS Language Environment (LE). Enclaves sind Thread-ähnliche Einheiten. Sie sind Laufzeit-Einheiten, die das Äquivalent einer multithreaded Verarbeitung durch den CICS Nucleus ermöglichen. Enclaves stellen sicher, dass parallel ausgeführte Transaktionen innerhalb eines einzigen z/OS Adressraums (Region) voneinander isoliert sind.

Language Environment Enclaves können eingesetzt werden, um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressenraums voneinander zu isolieren. Sie werden u.a. in transaktionalen Subsystemen wie CICS, IMS und DB2 benutzt. Zahlreiche Enclaves laufen in einem einzigen CICS Virtuellen Adressenraum. CICS Anwendungen (Transaktionen) werden in getrennten Enclaves ausgeführt.

Im Falle von CICS ist die LE Laufzeitumgebung Bestandteil des CICS Nucleus.

Enclaves benutzen hierfür einen Speicherschutzschlüssel (Storage Protection Key). Der Hardware Speicherschutz arbeitet unabhängig und zusätzlich zu dem üblichen Speicherschutz über Segment- und Seitentafeln und ist eine Einrichtung der z/Series Hardware Architektur, die auf anderen Plattformen nicht verfügbar ist. Der CICS Nucleus verwendet typischerweise Speicherschutzschlüssel Nr. 8, die Anwendungen in ihren Enclaves Speicherschutzschlüssel Nr. 9. Der CICS Nucleus ist somit vor falschen Zugriffen durch fehlerhafte Anwendungen in den Enclaves geschützt. Siehe Band 1, Abschnitt 1.3.10.

Im Falle von Java läuft in jeder Enclave eine JVM, und in dieser eine Java Transaktion unter einem eigenen Open TCB (JTCB). Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines Adressraums laufen zu lassen, siehe Band 1, Abschnitt 9.3.3.

19.2.6 LE Program Management

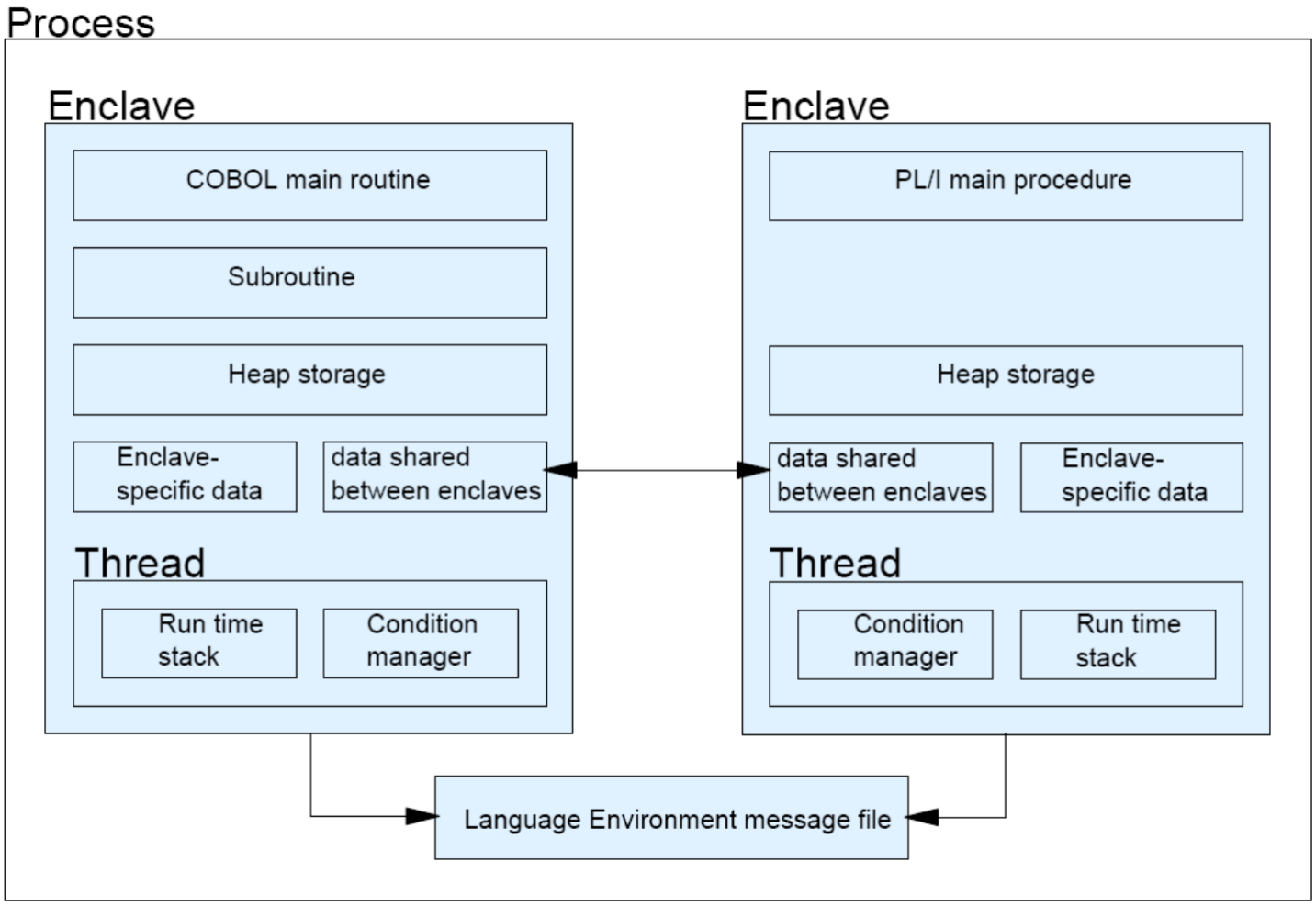


Abb. 19.2.8
Struktur der Enclave

Drei Elemente:

- **Prozess,**
- **Enclave und**
- **Thread**

sind der Kern des Language Environment Programm-Management-Modells.

Ein Prozess kann mehrere Enclaves enthalten. Jede Enclave enthält ihren eigenen Programm-Code. Der Code kann in verschiedenen Sprachen geschrieben werden, darunter auch Cobol und Java.

Die Threads innerhalb der gleichen Enclave benutzen gemeinsam den gleiche Code, und ebenso den Heap-Storage. Jeder Thread hat seinen eigenen Stack.

Die höchste Ebene des Language Environment Programm-Modells ist der Prozess. Jeder Prozess hat seinen eigenen Adressenraum. Ein Prozess ist eine Sammlung von Ressourcen, sowohl Programmcode als auch Daten.

Ein Prozess besteht in der Regel aus einer Enclave und ist logisch getrennt von anderen Prozessen. Jeder Prozess hat seinen eigenen Speicherplatz; Prozesse haben keinen gemeinsamen Speicherplatz. Prozesse sind gleichberechtigt und unabhängig voneinander. Es besteht keine hierarchische Gliederung.

Prozesse können neue Prozesse erstellen. Sie kommunizieren miteinander mit Hilfe einer Language Environment definierten Kommunikation, z. B. um anzuzeigen, dass ein Prozess beendet wurde.

Die Enclave ist eine Sammlung von Routinen, die eine Anwendung darstellen. Eine Enclave ist das Äquivalent von einer der folgenden Bezeichnungen:

- **Eine Run Unit in COBOL**
- **Ein Programm, (Main C-Funktion und seine Sub-Funktionen), in C/C++**
- **Einer Main Procedure und alle seine Unterprogramme in PLI**
- **Einer JVM und ihre Klassen in Java**

Die Enclave besteht aus einem Hauptprogramm und einer beliebige Anzahl von Unterprogrammen. Die Main Routine ist als erstes in einer Enclave auszuführen; alle nachfolgenden Routinen werden als Unterprogramme behandelt.

Externe Daten sind nur innerhalb der Enclave, in der sie sich befinden, verfügbar. Der Geltungsbereich der externen Daten ist die Enclave. Auch wenn externen Daten identisch Namen in verschiedenen Enclaves aufweisen, sind sie nur innerhalb ihrer Enclave gültig.

Die Threads in einer Enclave können weitere Enclaves erstellen, diese können mehr Threads erstellen, usw.

Für Sonderfälle existiert ein Mechanismus, mit dem Enclaves Daten gemeinsam nutzen können. Ein Beispiel ist die PLI Standard SYSPRINT File. Sie ist shared von mehreren Enclaves innerhalb einer Anwendung.

Es existiert eine Language Environment Message-File, die von mehreren Enclaves gemeinsam genutzt werden kann. Dies ist ein nützlicher zentraler Speicherort für Nachrichten, die während der Ausführung von Programmen innerhalb der Enclaves generiert werden können.

Einer der wichtigen Eigenschaften von CICS Enclave Transaktionen ist, dass der z/OS Work Load Manager sie unabhängig voneinander verwalten kann, auch wenn mehrere Enclaves in dem gleichen Adressenraum laufen.

Den einzelnen aktiven Enclaves in einem Adressraum können ihre eigenen unabhängigen Dispatching und I/O-Prioritäten zugeordnet werden, abhängig von den WLM Goals, die der Benutzer definiert hat. Wichtige Enclave Transaktionen können ihre WLM Goals nicht verpassen, weil weniger wichtige Transaktionen im gleichen Adressraum ausgeführt werden.

Jedes Enclave besteht aus mindestens einem Thread.

Ein Thread ist ein Ausführungs-Konstrukt, das aus synchronen Aufrufen und Terminierungen von Routinen besteht. Ein Thread wird von dem System mit seinem eigenen Runtime-Stack, Befehlszähler und Registern dispatched. Threads können gleichzeitig mit anderen Threads innerhalb einer Enclave existieren.

Der Runtime Stack steuert die Ausführung eines Threads. Er enthält außerdem den Befehlszähler, Register und Condition-Handling Mechanismen. Jeder Thread stellt eine unabhängige Instanz einer Routine dar, die in einer Enclave unter unter Benutzung der Enclave Ressourcen läuft.

Threads benutzen gemeinsam alle Ressourcen einer Enclave. Ein Thread kann den ganzen Speicher innerhalb einer Enclave adressieren. Alle Threads sind gleichberechtigt und unabhängig voneinander und nicht hierarchisch miteinander verbunden.

Weil Threads mit ihrem eigenen Run-Time Stacks arbeiten, können sie gleichzeitig innerhalb einer Enclave laufen und ihren Speicherplatz verwalten. Weil sie gleichzeitig ausgeführt werden können, können Threads verwendet werden, um eine parallele Verarbeitung von Anwendungen (und von ereignisgesteuerten Anwendungen) zu implementieren.

In einer z/OS Enclave kann jeder Thread mit unabhängigen eigenen Performance Zielen ausgeführt werden. Mit z/OS Workload-Management (WLM) kann man z.B. z/OS Performance-Ziele für einzelne DB2-Server-Threads etablieren.

Threads innerhalb einer Enclave erfordern eine kritische Handhabung um zu gewährleisten, dass sie sich nicht gegenseitig beeinflussen. Die Isolation muss gewährleistet sein. Deshalb benutzen CICS-Anwendungen in der Regel keine Threads um mehrere Transaktionen in einem einzigen Enclave laufen zu lassen.

Allerdings verwendet der neue CICS „JVM Server“ Threads! Das OSGi Framework innerhalb jeder JVM stellt die erforderliche Isolation (teilweise) sicher.

19.3 CICS und Java Standard Edition

19.3.1 Task Control Block

Prozesse (und Threads) unter z/OS werden von einem Task Control Block (TCB) gesteuert. Unix benutzt die Bezeichnung Process Control Block (PCB).

Ein TCB ist eine Datenstruktur in dem z/OS-Kernel Adressraum, der die erforderliche Informationen enthält, um einen bestimmten Prozess zu verwalten. Der TCB ist die Manifestation eines Prozesses in z/OS.

Ein TCB umfasst:

- **Den Identifier des Prozesses (Prozessidentifier, oder PID).**
- **Register Inhalte für den Prozess (Mehrzweck, Gleitkomma, Control, Access Register) einschließlich des Programm-Status Wortes (PSW) für den Prozess.**
- **Identifizierung des Adressraums für den Prozess.**
- **Priorität: Ein Prozess mit höherer Priorität bekommt die erste Präferenz (Gegenstück zum "nice"-Command in Unix-Betriebssystemen).**
- **Prozess Accounting Informationen, wie z.B.: wann wurde der Prozess zuletzt ausgeführt, wie viel CPU-Zeit hat er akkumuliert, usw.**
- **I/O Information (z.B. I/O Devices, die dem Prozess zugeordnet wurden, Liste der geöffneten Data Sets, usw.).**

In einer z/OS-Umgebung kann ein Programm in einem von zwei Modi ausgeführt werden:

- **TCB-Modus, in der Regel als Task-Modus bezeichnet,**
- **SRB-Modus (Service Request-Block Modus).**

Die meisten Programme - und alle im Userstatus laufenden Programme – werden als „Task“ ausgeführt. Jeder Thread der Ausführung wird durch einen Task Control Block (TCB) repräsentiert. Ein Programm kann in mehrere Tasks unterteilt werden, was eine Ausführung auf mehreren Prozessoren ermöglicht. Programme im Task Modus können I/O ausführen, auf Events warten und alle Arten von System-Diensten nutzen.

SRBs sind leichtgewichtige und effiziente z/OS Ausführungs-Threads, die nur im Supervisor State verfügbar sind. Der SRB-Modus wird von manchen Kernel Routinen benutzt, um bestimmte Performance-kritischen Funktionen auszuführen. Wenn zum Beispie ein I/O-Interrupt auftritt, dispatches z/OS einen SRB zu dem entsprechenden Adressen Raum, um einen ECB (Event Control Block) zu benachrichtigen, und um möglicherweise andere Verarbeitungsvorgänge auszuführen.

Der SRB-Modus ist weitgehend ungenutzt außerhalb des Betriebssystems und außerhalb von Middleware-Produkten wie DB2. Ein Grund dafür ist: Der SRB-Modus ist nur im Supervisor-Status verfügbar. Die meisten Programmierer glauben, SRB-Modus-Funktionen sind zu schwierig zu testen und zu debuggen. Als Folge läuft fast aller z/OS Anwendungs-Code im Task-Modus. Der SRB-Modus wird nur dort eingesetzt, wo er absolut notwendig ist.

Windows hat Prozesse Control Blöcke ähnlich zu z/OS TCBs. Das Windows Äquivalent eines SRB wird als "Fibre" bezeichnet.

19.3.2 CICS Quasi-reentrant TCB

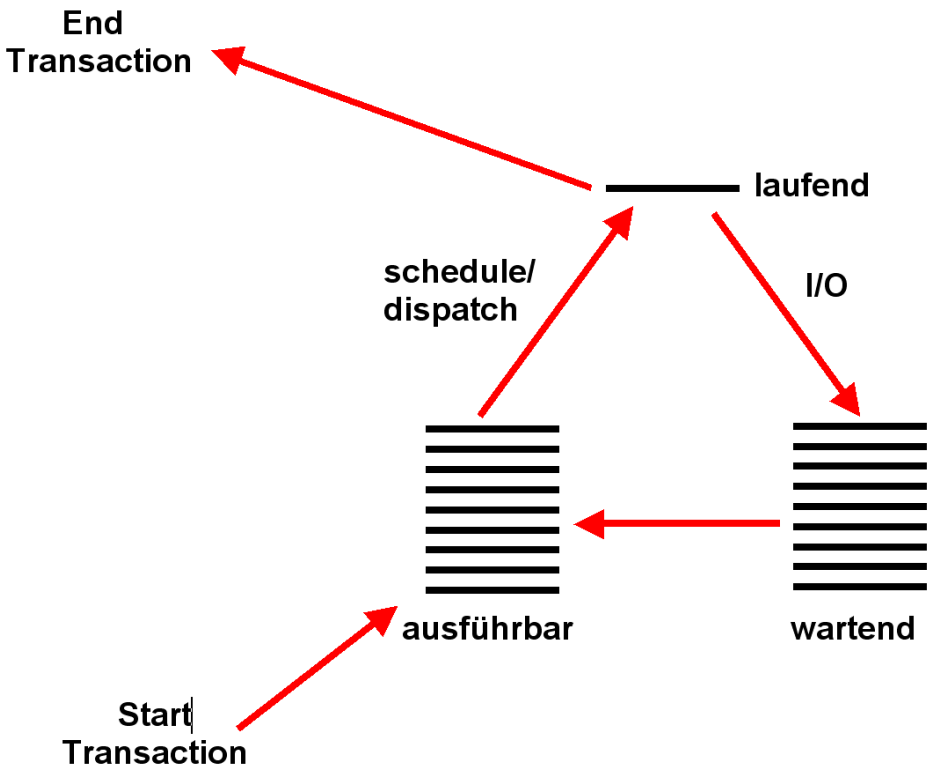


Abb. 19.3.1
Transaction State

Traditionell laufen alle CICS Transaktionen in einem einzigen Adressraums unter der Steuerung eines einzigen Prozesses (dem CICS Nucleus) und einem einzigen TCB (genannt der Quasi Reentrant TCB (QR TCB). Jedes Mal, wenn die Transaktion einen EXEC CICS-Befehl ausführt, wird die Steuerung an den CICS Nucleus transferiert. Der Programmierer muss sicherzustellen, dass die Pfad Länge zwischen EXEC CICS Kommandos kurz genug ist, um eine akzeptable Reaktionszeit (Antwortzeit) zu garantieren.

Ein in Java geschriebenes CICS-Programm benutzt keine EXEC CICS Befehle. Auch kann ein Java-Programm Ressourcen außerhalb der Kontrolle von CICS verwenden. RMI/IIOP ist ein Beispiel. Die Pfad Länge kann nicht gesteuert werden. Daher ist es nicht möglich, dass Java CICS Transaktionen mit einem QR TCB arbeiten.

19.3.3 Quasi-reentrant TCB

Unter z/OS, existieren mehrere Varianten des TCB. Es gibt zwei verschiedene Alternativen für das Betreiben eines CICS Subsystems, wobei jede mit einer anderen Art von TCB arbeitet.

In der traditionellen Alternative laufen alle Anwendungsprogramme unter einer einzigen, CICS-managed, Task Control Block (TCB), dem quasi-reentrant (QR) TCB. In diesem Fall müssen alle CICS Anwendungsprogramme quasi-reentrant geschrieben sein.

CICS quasi-reentrant Anwendungsprogramme werden durch den CICS Dispatcher unter dem QR TCB aufgerufen. Bei der Ausführung unter diesem TCB kann ein Programm sicher sein, dass keine anderes quasi-reentrant Programm ausgeführt werden kann, bis es die Kontrolle (die Verfügungsgewalt über die einzige CPU) aufgrund eines EXEC CICS-Commands aufgibt. An diesem Punkt wird die Transaktion unterbrochen (Waiting oder Ready State). Während dieser Warte Zeit kann eine andere Transaktion das gleiche, quasi-reentrant Anwendungsprogramm ausführen. Dies bedeutet, ein quasi-reentrant Anwendungsprogramm kann gleichzeitig von mehr als einer Task benutzt werden, obwohl nur eine Task es in jedem Augenblick ausführen kann.

Es existiert nur eine einzige Kopie des quasi-reentrant Anwendungsprogramms im Hauptspeicher. Um sicherzustellen, dass parallel laufende Transaktionen, welche das gleiche quasi-reentrant Anwendungsprogramm benutzen, sich nicht mit der Nutzung des Arbeitsspeichers stören, unterhält CICS eine getrennte Kopie des Arbeitsspeichers für jede Transaktion. Wenn somit eine Kopie eines Anwendungsprogramms gleichzeitig von 11 Tasks benutzt wird, existieren 11 Kopien des Arbeitsspeichers in dem entsprechenden dynamischen CICS Speicherbereich (Dynamic Storage Area, DSA), siehe Band 1, Abschnitt 8.3.7.

Achten Sie darauf, wenn ein Programm langwierige Berechnungen durchführt, weil ein Anwendungsprogramm die Kontrolle (Verfügungsgewalt über die CPU) von einem EXEC CICS Command bis zum nächsten behält. Die Verarbeitung von anderen Transaktionen unter dem QR TCB ist während dieser Zeit ausgeschlossen. CICS terminiert eine Transaktion, die nicht vor Ablauf eines angegebenen Zeit Intervalls die Kontrolle abgibt..

Der CICS QR TCB bietet Schutz durch die ausschließliche Kontrolle der globalen Ressourcen nur dann, wenn alle Transaktionen, die auf diese Ressourcen zugreifen unter dem QR TCB laufen. Es bietet keine automatischen Schutz vor anderen Prozessen, die gleichzeitig unter einem anderen (offenen) TCB ausgeführt werden.

19.3.4 Open TCB

Wenn alle Transaktionen unter einem QR TCB laufen, impliziert dies, dass eine CICS Region nur mit einer einzigen CPU läuft. Es ist aus diesem Grund, dass mehrere Application Owning Regions (AOR) beliebt sind (siehe Abb. 9.3.4). Jede AOR läuft auf einer anderen CPU.

Das Open Transaction Environment (OTE) ist ein Umfeld, in dem

- **Eine einzelne CICS-Region mehrere CPUs benutzen kann, und**
- **CICS Anwendungscode nicht-CICS Dienste (Einrichtungen außerhalb des Umfangs der CICS API) innerhalb des CICS Adressraum ausführen kann, ohne Interferenz mit anderen Transaktionen.**

.

Das Problem ist, ruft eine Transaktion einen Service außerhalb der CICS-Adressenraums auf, kann diese Transaktion für eine lange Zeit zu sperren, ohne dass der CICS Nucleus dessen bewusst sein muss. Aus diesem Grund läuft jede Anwendung (Transaction), die das Open Transaction Environment (OTE) nutzt, unter einem eigenen open TCB. In der CICS Region können mehrere aktive (parallel laufende) Programme mit eigenen open TCBs existieren anstatt eines einzigen aktiven Programms mit einem QR TCB. Während der QR TCB vom CICS Nucleus dispatched wird, erfolgt das open TCB Dispatching durch den z/OS Kernel. Wenn eine open TCB Transaktion einen non-CICS Service aufruft, die den TCB blockiert, hat dies keine Auswirkungen auf andere open TCB CICS Anwendungen.

Es gibt auch andere Verwendungen für das Open Transaction Environment. Allerdings hat das alles seinem Preis. Die Komplexität wächst, und die Performance kann schlechter werden. So ergänzt der offene TCB den QR TCB, aber ersetzt ihn nicht, und der QR TCB bleibt sehr beliebt. In einer gegebene z/OS-Installation können einige CICS Regionen mit dem QR TCB, und andere mit dem open TCB laufen.

Es existieren mehrere Varianten des open TCB. Einige von ihnen sind speziell für CICS Java-Anwendungen bestimmt.

Details unter
http://www.cedix.de/VorlesMirror/Band2/OTE.pdf
http://www.cedix.de/VorlesMirror/Band2/OpenTCB.pdf

19.3.5 Programming CICS Applications in Java Enterprise Edition

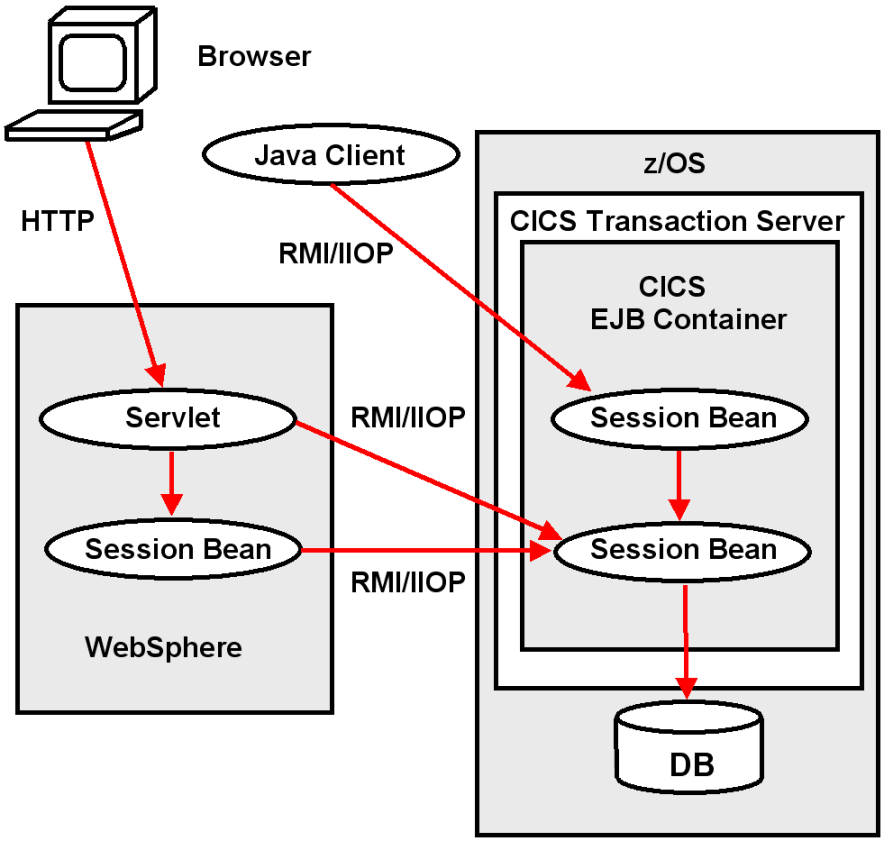


Abb. 19.3.2
Benutzung von Enterprise Java Beans in einer CICS Anwendung

Die meisten CICS-Anwendungen wurden bisher in Cobol, PL/I oder Assembler programmiert. C/C++ wurde weniger häufig verwendet. Mit der Einführung von Java wurde es möglich, CICS-Anwendungen auch in Java zu programmieren. Viele Experten gehen davon aus, dass Java die dominierende Sprache werden wird, um neue CICS-Anwendungen zu programmieren. Es gibt drei alternative Möglichkeiten, um CICS-Anwendungen in Java zu schreiben:

1. **Der z/OS "High Performance Java Compiler" hat eine Option, Objekt-Code wie ein Cobol oder PL/I Compiler zu generieren. Dieser Objekt Code wird gelinked und geladen wie jede andere Code. Der Java Quellcode ist auf andere Plattformen übertragbar, der Objekt-Code aber nicht. Der Java Standard unterstützt diese Möglichkeit nicht, sie existiert außerhalb des Java Standards.**

2. **Alternativ kann der z/OS High Performance Java Compiler Byte Code generieren. Der Byte-Code wird innerhalb einer JVM, die als Teil des CICS Transaction Server installiert wird, ausgeführt. Es wird die JSE an Stelle der JEE benutzt.**

   Es gibt jedoch ein Problem mit diesem Ansatz. Die Java-Sprache erlaubt nicht den Einsatz von EXEC-Anweisungen. Deshalb wird die EXEC CICS-Schnittstelle durch eine Reihe von Java Bibliothek Aufrufe ersetzt, durch die „JCICS“ Schnittstelle.

3. **Beide bisher genannten Ansätze benutzen keinen EJB-Container. Wenn Sie JEE Einrichtungen mit CICS verwenden möchten, können Sie eine JEE EJB Container (mit einer eigenen JVM) innerhalb des CICS Transaction Servers installieren, siehe Abb. 9.3.2 . Dies ist in Wirklichkeit ein Corba ORB. Es ermöglicht die Verwendung von RMI/IIOP.**

Die letzten beiden Ansätze erfordern die Verwendung des open TCB, da die Java-Programme Einrichtungen außerhalb von CICS verwenden können.

CICS unterstützt keine Entity Beans. Es benutzt eigene Verfahren, um die Persistenz zu managen.

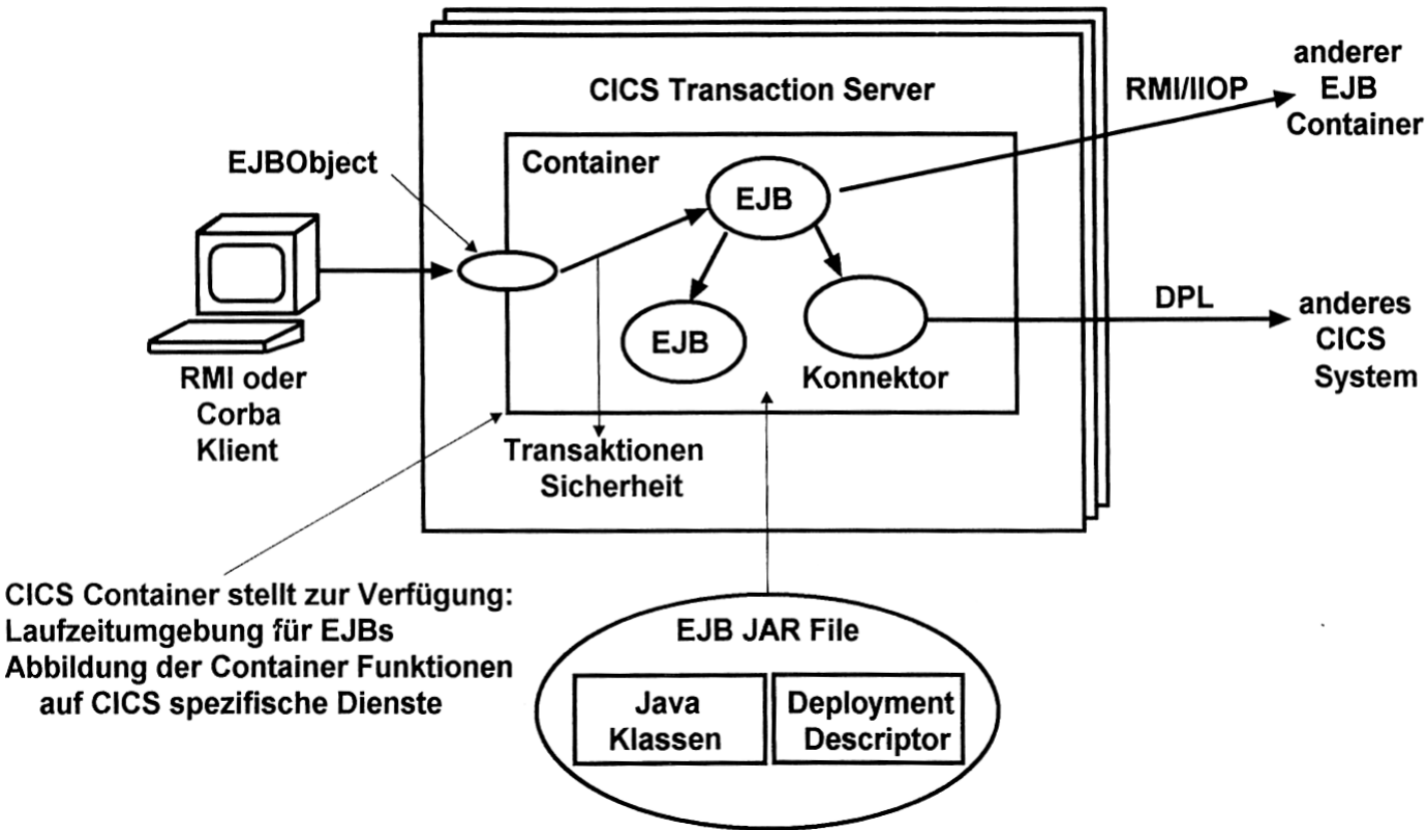


Abb. 19.3.3
CICS EJB Container

CICS Session Beans können über RMI/IIOP aufgerufen werden. Jede Java-Klasse oder jeder Corba Client ist in der Lage, mit RMI/IIOP auf eine CICS Session Bean zuzugreifen. CICS Session Beans können ihrerseits anderen EJBs außerhalb CICS via RMI/IIOP aufrufen. Außerdem können sie mittels DPL mit Programmen in anderen CICS Servern kommunizieren, auch wenn diese z.B. in Cobol geschrieben wurden, siehe Abb. 19.3.3.

Im Vergleich zu anderen Umgebungen können Enterprise Beans in einem CICS EJB Container alle CICS Transaction Management Services wie System Log Management, Performance Optimization, Runaway und Deadlock Detection sowie Monitoring und Statistics nutzen.

Der CICS EJB Container benutzt das X/Open Distributed Transaction Processing Model.

19.3.6 Programmieren von CICS Transaktionen in der Java Standard Edition

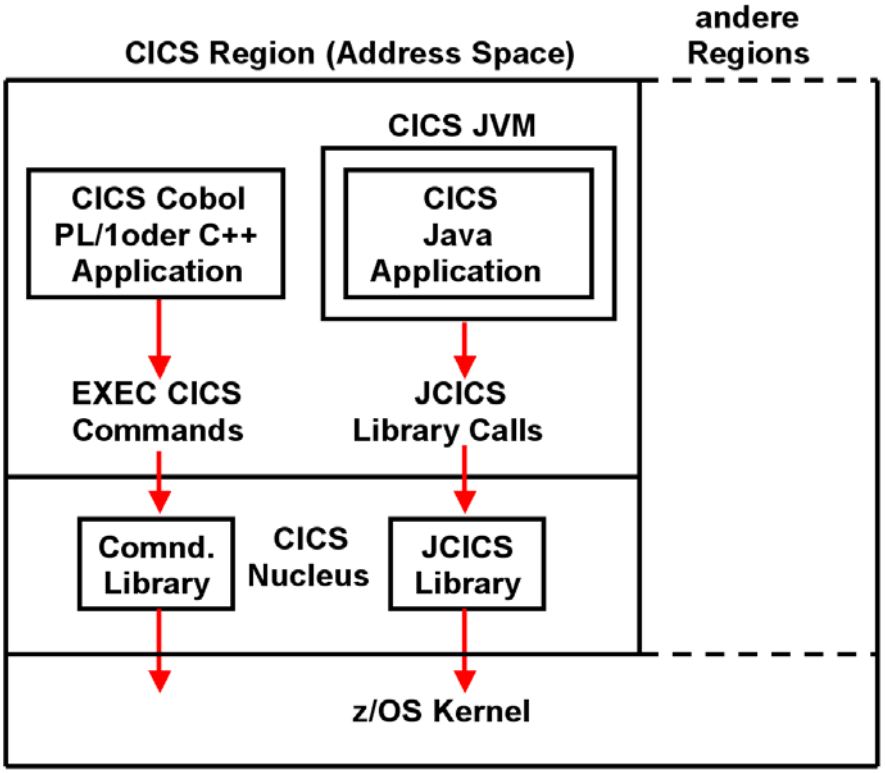


Abb. 19.3.4
JCICS Java Anwendungsprogrammierung

JEE Server wie WebSphere, Web Logic, Netweaver, und Open-Source Produkte wie Jboss und Geronimo sind sehr beliebte Transaction Processing Produkte. CICS kann ebenfalls einen JEE-Server integrieren. Jedoch ist die JSE (Java Standard Edition) eine überraschend attraktive Alternative für CICS Java-Programme. Dies ist der Grund:

- Ein JEE Server integriert einen Transaktion Monitor in eine bestehendes JEE Infrastruktur, die viele Funktionen aufweist, die nicht direkt mit der Transaktionsverarbeitung zu tun haben. JNDI und RMI/IIOP sind Beispiele. Viele dieser Funktionen werden von CICS eigentlich nicht benötigt, und stellen einen überflüssigen Overhead dar.
- CICS integriert Java in einer vorhandenen Transaktion Processing Infrastruktur. OTE ermöglicht es, JSE Programmen direkt unter CICS laufen zu lassen.

Wie alle anderen Java-Programme laufen CICS JSE Programme innerhalb einer JVM. Sie können Seite an Seite mit Cobol, PL/I und C/C++ CICS Programmen ausgeführt werden.

Java hat aber keine Einrichtungen um die EXEC CICS Aufrufe zu implementieren. Deshalb benutzen Java-Programme statt dessen die JCICS Bibliotheksroutinen. Die JCICS Bibliotheksaufrufe ersetzen eins zu eins die EXEC CICS Commands und nehmen damit den Platz des EXEC CICS Befehle ein.

19.3.7 Parallele Ausführung von JSE Java Transaktionen

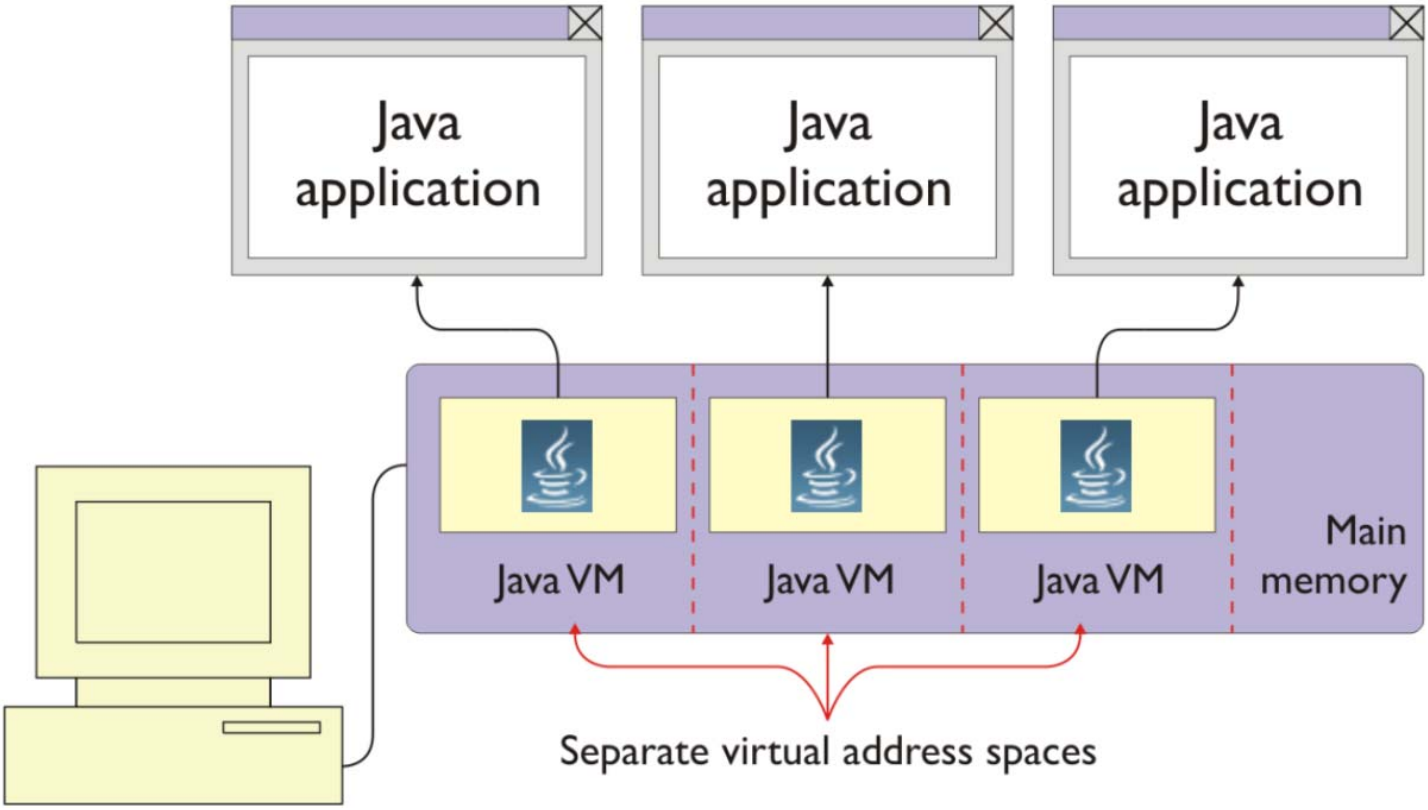


Abb. 19.3.5
Multiprogrammierung mit Java

Der einfachste Weg, JSE Transaktionen parallel auszuführen ist, jeder Transaktion eine separaten JVM zuzuordnen, und eine getrennte z/OS Region jeder JVM. Das ist ziemlich aufwendig in Bezug auf den Verbrauch von CPU-Zyklen. Unter CICS laufen daher alle JSE-Anwendungen genauso wie Cobol und PL/I Anwendungen innerhalb eines einzigen Adressraums.

Es gibt zwei JSE Alternativen, dies zu tun: die JVM Pool Architektur und die JVM Server Architektur. Wir diskutieren die JVM Pool Architektur zuerst.

19.3.8 CICS JVM Pool Architektur

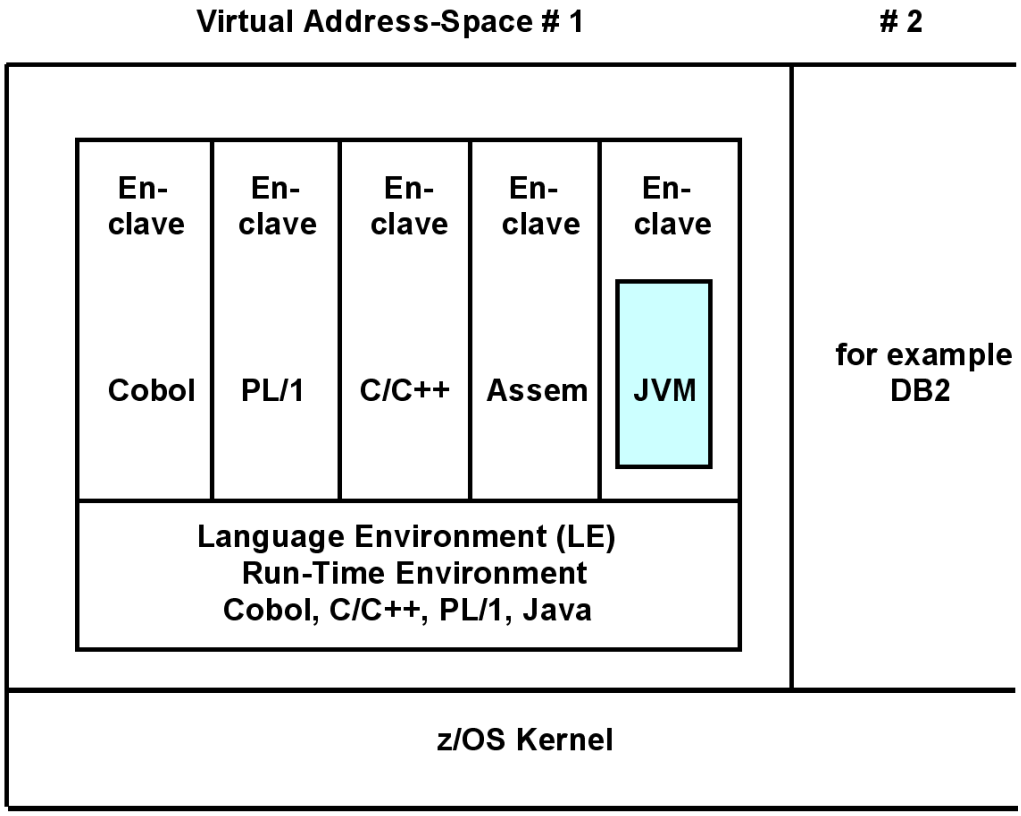


Abb. 19.3.6
Multiple JVMs

CICS-Transaktionen, die in Cobol, C/C++, PL/I oder Assembler geschrieben sind, laufen in getrennten Enclaves; eine Enclave für jede Transaktion.

CICS Java-Transaktionen werden in getrennten JVMs ausgeführt, jeweils eine JVM pro Enclave. Java Threads werden in der Pool Architektur nicht benutzt.

Cobol, PL/I, C/C++ und Java Enclaves können gleichzeitig innerhalb einer CICS-Region laufen.

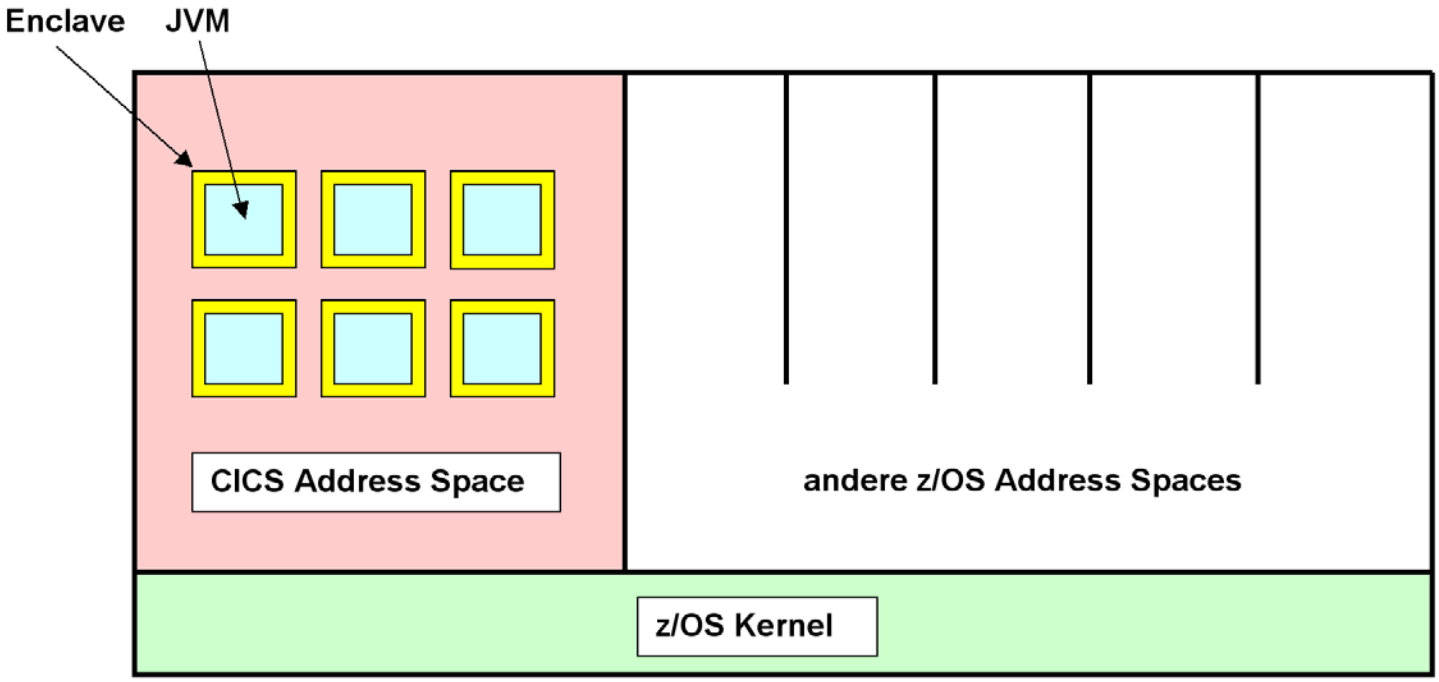


Abb. 19.3.7
JVM und Enclave

Ein wichtiger Punkt ist es die gegenseitige Isolation von mehreren Java Transaktionen (das „I“ in ACID) zu gewährleisten. Die JVM Pool Architektur ist ein gradliniger Ansatz. Hier unterhält CICS mehrere JVMs in seinen Adressraum, als JVM Pool bezeichnet. Einzelne CICS Tasks werden jeweils einer JVM in dem Pool zugeordnet, und nur eine einzige Java Anwendung läuft in jeder JVM. Getrennte JVMs garantieren die Isolation zwischen den Java Anwendungen.

Theoretisch sind > 100 JVMs möglich. Allerdings wird viel Speicherplatz benötigt (~ 20MByte + Heap). Die Anzahl der gleichzeitig laufenden Transaktionen ist durch die Anzahl der JVMs begrenzt, die in eine Region passen. Das Ergebnis sind maximal 20 gleichzeitige JVMs in einer 2 GByte Region.

19.3.9 JVM Zuordnung

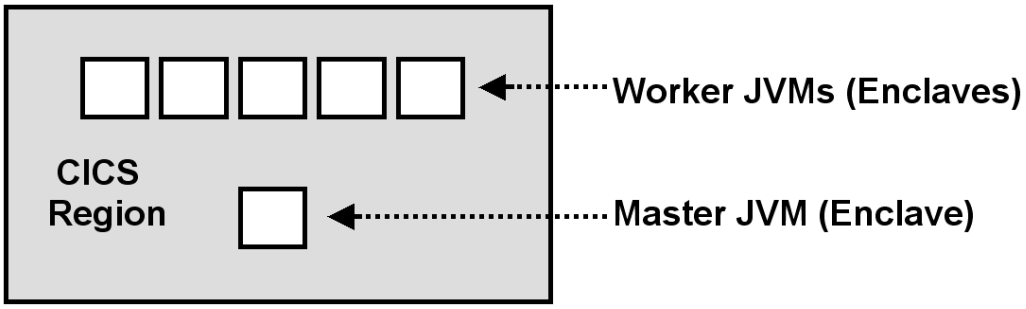


Abb. 19.3.8
Master JVM und Enclave

LE Enclaves können eingesetzt werden um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressenraums voneinander zu isolieren. Sie werden u.a. in transaktionalen Subsystemen wie CICS, IMS und DB2 benutzt. Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines CICS Adressenraums laufen zu lassen.

Es existiert immer eine Enclave für jede JVM. Eine Master JVM Eine Master JVM cloned je nach Bedarf mehrere Worker JVMs und steuert sie. Transaktionen werden in den Worker JVMs ausgeführt.

CICS unterhält eine „Shared Class Cache Facility für die JVM. Mehrere JVMs können gemeinsam einen einzigen Cache mit Class Files nutzen. Die gemeinsam genutzte Cache ersetzt den System-Heap und die Anwendungsklassen für die JVMs. JVMs, die gemeinsam genutzte Klassen verwenden, starten schneller und haben einen geringere Speicherplatzbedarf als JVMs, die dies nicht tun.

Die Master JVM initialisiert u.A. den Shared Class Cache. Die Master JVM wird nicht für die Ausführung von Java Anwendungen benutzt.

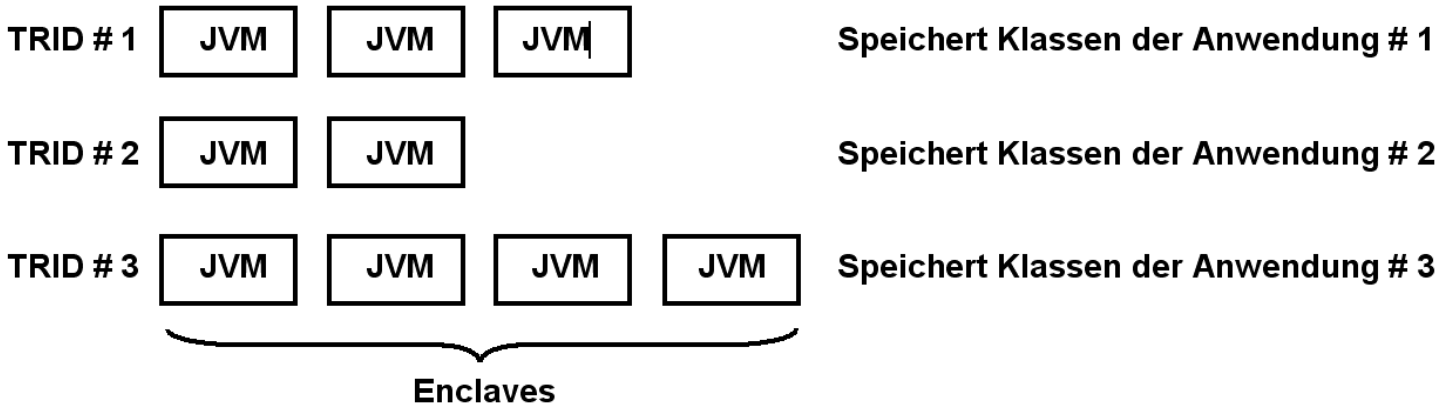


Abb. 19.3.9
Enclaves mit identischen geladenen Klassen

Wenn eine neue Transaktion eintrifft, sucht CICS eine freie Enclave mit Ihrer vorinstallierten JVM, um in dieser die Transaktion auszuführen. Wenn möglich, sucht CICS eine JVM aus, in der die gleiche Anwendung, gekennzeichnet durch ihre TRID, bereits gelaufen ist. In diesem Fall müssen die Anwendungsklassen nicht neu geladen werden.

Nehmen wir an, in einer Java CICS Region sind 20 JVMs vorhanden. In jeder JVM sind die Anwendungsklassen einer spezifischen Java CICS Anwendung gespeichert. Häufig sind die Klassen einer Anwendung in mehr als einer JVM gespeichert.

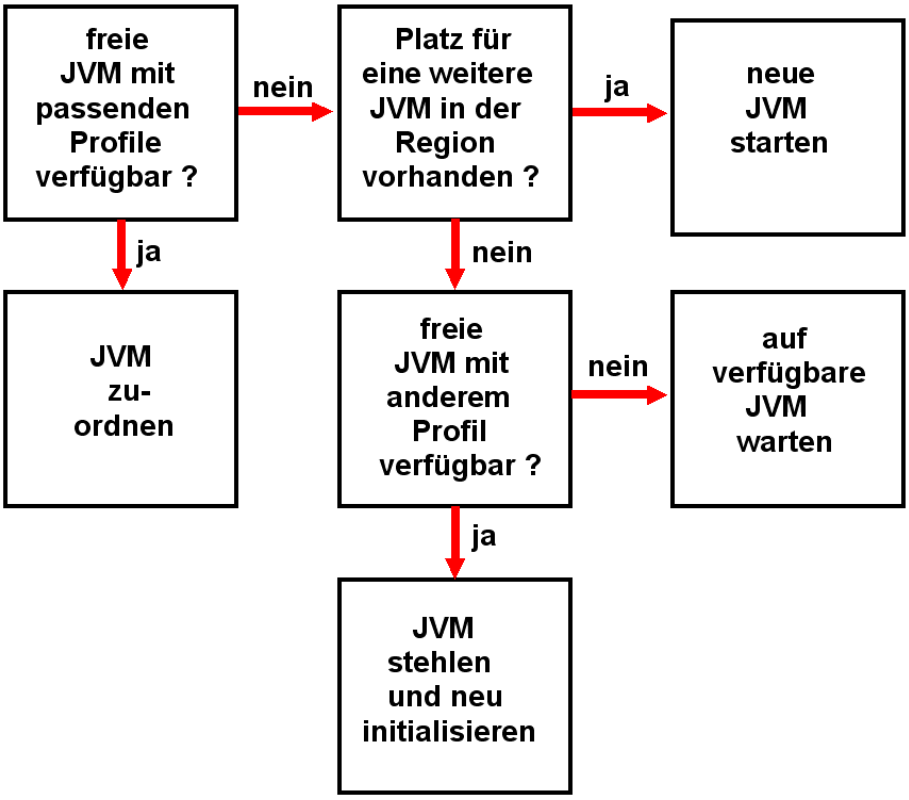


Abb. 19.3.10
Auswahlalgorithmus für die Zuordnung einer JVM

Dargestellt ist der Algorithmus, mit dem beim Starten einer weiteren Transaktion eine passende JVM für deren Bearbeitung ausgewählt wird.

Ein bestimmter Transaktionstyp wird durch seine TRID gekennzeichnet, und kann von mehreren Transaktionen parallel ausgeführt werden. Unterschiedliche Transaktionstypen unterscheiden sich durch unterschiedliche Anwendungsklassen. Wenn eine neue Transaktion gestartet wird, wird nach Möglichkeit eine freie JVM ausgewählt, in der die richtigen Anwendungsklassen bereits geladen sind. Wenn das nicht möglich ist, werden die benötigten Anwendungsklassen in eine verfügbare Worker JVM nachgeladen.

19.3.10 Ausführung einer Folge von Transaktionen

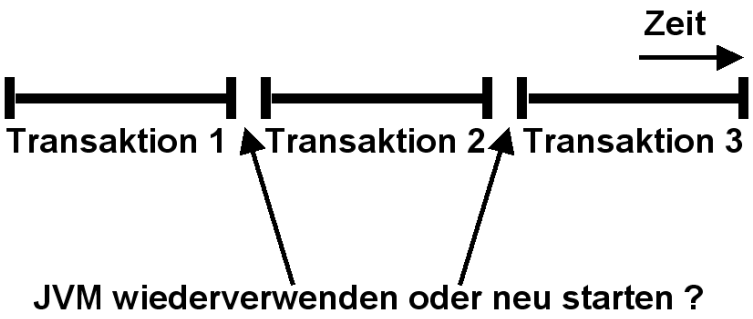


Abb. 19.3.11
Serielle Nutzung einer JVM

Eine Transaktion kann die Ausführung der Folge-Transaktion beeinflussen, indem sie den Zustand (State) der JVM ändert. Beispiele für eventuelle sicherheitskritische Reste der vorhergehenden Transaktion sind:

- **überschriebene statische Variablen**
- **Starten von Threads**
- **geladene native Bibliotheken.**

Klassischer Ansatz: Für jede Transaktion wird eine neue JVM gestartet und nach Abschluss der Transaktion wieder beendet.

Das Hoch- und Herunterfahren einer JVM hat jedoch einen erheblichen Zeitaufwand zur Folge. Bei jeder Initialisierung einer JVM werden

- **60 System Klassen geladen,**
- **700 Array-Objekte und**
- **1000 non-Array-Objekte allokiert und angelegt.**

Pfadlänge bis zu 100 Millionen Maschinenbefehle, ca. 0,1 Sekunde Verarbeitungsdauer, sind möglich.

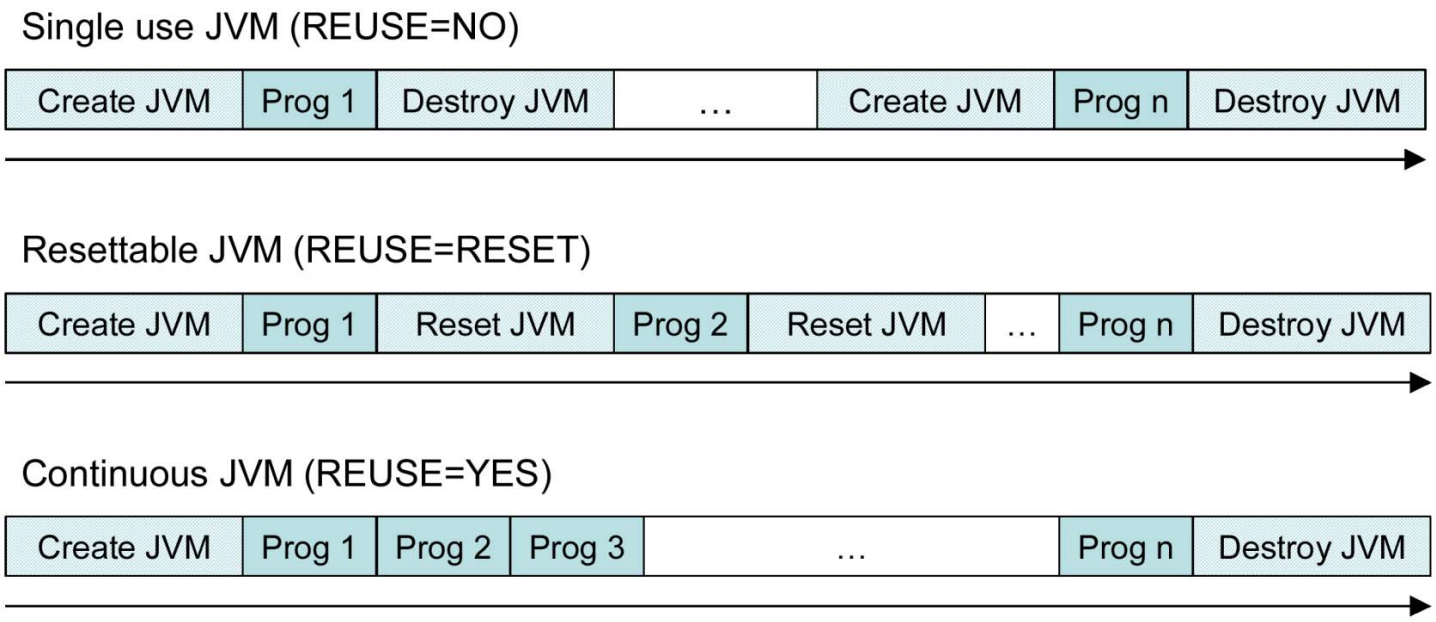


Abb. 19.3.12
CICS JVM Modus

In the CICS JVM Pool Architecture kann eine JVM kann in einem der folgenden Modi laufen:

1. **Single Use mode. Die JVM wird zerstört, und für die nächste Transaktion wird eine neue JVM gestartet.**
2. **Resettable mode. Eine Zusatzeinrichtung bewirkt, dass der ursprüngliche Zustand der JVM wiederhergestellt wird.**
3. **Continuous mode. Es ist Aufgabe des Anwendungs-Programmierers, sicherzustellen, den dass der Zustand der JVM nicht verändert wird..**

Der Single Use Modus hat den Nachteil, dass der Aufwand für das Starten einer JVM sehr hoch ist. Dies hat zur Folge, dass der Single Use Modus aus Performance Gründen nur in Ausnahmefällen eingesetzt werden kann.

Der Continuous Use Modus hat das Problem, dass die JVM nach Beendigung einer Transaktion sich im gleichen Zustand wie zu Beginn einer Transaktion befinden muss. Dies hat der Anwendungsprogrammierer zu gewährleisten. Was hierzu zu tun ist, ist jedoch nicht ausreichend dokumentiert, und erfordert sehr tief gehende Java Kenntnisse, über die ein durchnittlicher Java Programmierer nicht notwendigerweise verfügt.

Dieses Problem löst der Resettable Modus, der jedoch nicht Teil des JEE Standards ist, und somit eine unerwünschte IBM proprietäre Erweiterung des Standards darstellt ist.

Dr. Jens Müller hat in seiner Diplomarbeit: " Anwendungs- und Transaktionsisolation unter Java ", Mai 2005, eine detaillierte Untersuchung der JVM Isolationseigenschaften durchgeführt. Siehe http://www.cedix.de/VorlesMirror/Band2/javaisol.pdf. Viele der Isolationsprobleme können durch eine sorgfältige Programmierung umgangen werden. Im Vergleich zu Cobol CICS Programmen ist dies jedoch eine zusätzliche Herausforderung.

19.3.11 CICS JVM Server Architecture

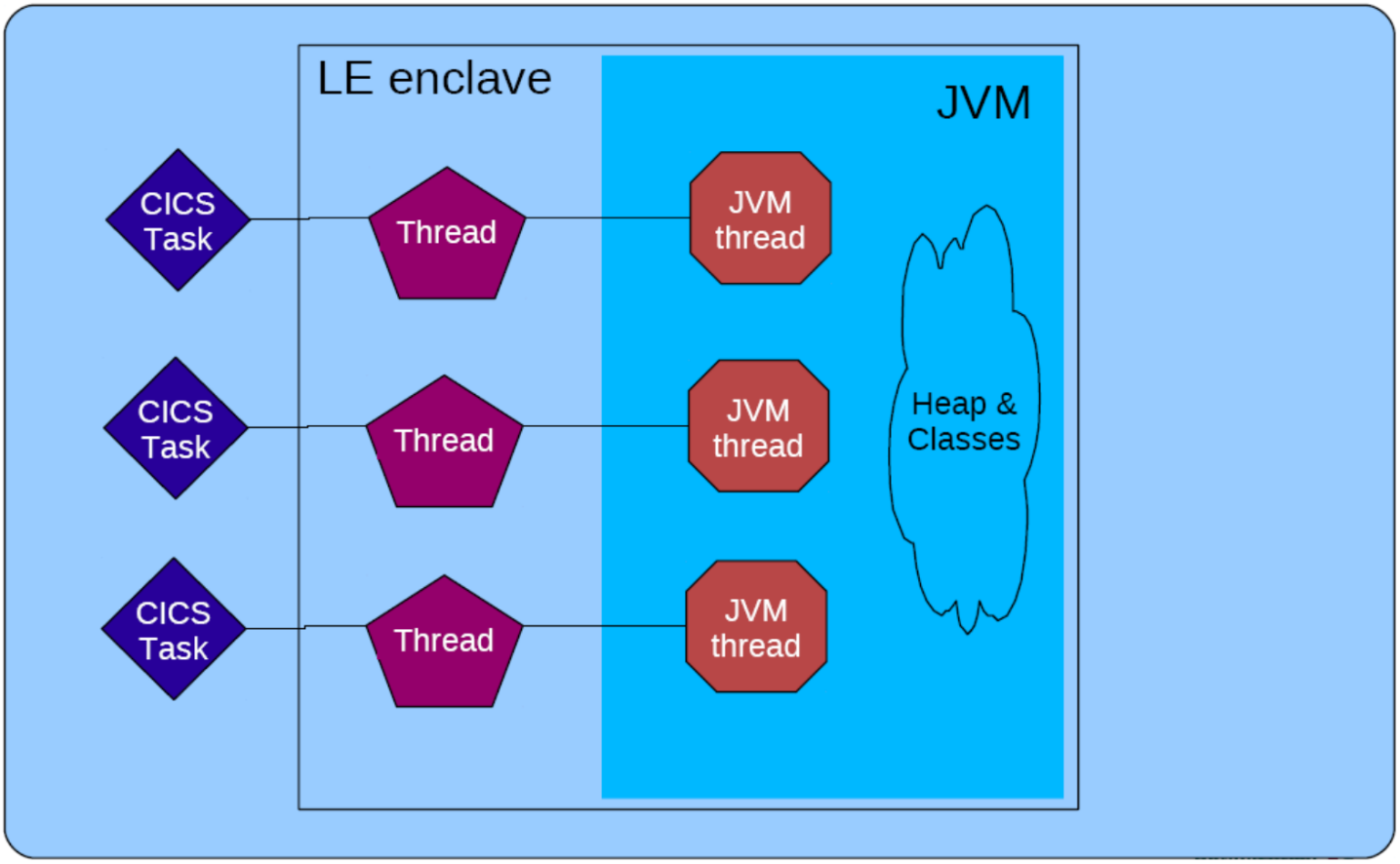


Abb. 19.3.13
Enclave Threads und JVM Threads

Der "JVM Server" ist eine neue Implementierung der JVM. Es laufen mehrere CICS Transaktionen als unabhängige JVM Threads innerhalb einer einzigen JVM. Jeder JVM Thread wird auf einen LE Enclave Thread abgebildet. Die JVM Server Architektur wird möglicherweise die JVM Pool Architektur ablösen.

Über viele Jahre benutzten CICS und WebSphere die gleiche, als „Sovereign“ bezeichnete Implementierung der JVM. Unterschiedliche Optimierungspotentiale führten dazu, dass WAS heute die „J9“ JVM verwendet. Diese und die Sun „HotSpot“ JVM gelten als die performantesten verfügbaren JVMs. Die J9 wird auch in der JSE und der JME eingesetzt.

CICS entwickelte statt dessen die „JVM Server“ JVM, die spezifisch für die CICS Bedürfnisse optimiert wurde. Die JVM Server JVM ermöglicht es, mehrere Transaktionen als Java Threads innerhalb einer einzigen JVM laufen zu lassen.

Die JVM Server Umgebung ermöglicht die gegenseitig Isolation von mehreren Java Anwendungen innerhalb einer JVM durch den Einsatz von Industrie-Standard OSGi-Bundles. Siehe http://www.cedix.de/VorlesMirror/Band2/OSGiArchi.pdf. Das OSGi-Framework innerhalb des JVM Servers bietet erforderliche Quarantäneeinrichtungen für mehrere gleichzeitig ausgeführte CICS Java Programme innerhalb der gleichen JVM. Dazu müssen CICS Java-Anwendungen in einem OSGi-Bundle verpackt werden, und dann als CICS BUNDLE Ressource mit Verweis auf die OSGi-Bundle deployed werden.

Der JVM-Server bietet auch integrierte Statistikfunktion zur Überwachung der JVM Garbage Collection. Dies erleichtert das Performance Tuning von Java CICS Anwendungen.

Der OSGI Ansatz wird auch von dem Android Betriebssystem als Laufzeitumgebung für Java Apps verwendet, siehe http://felix.apache.org/site/presentations.data/OSGi%20on%20Google%20Android%20using%20Apache%20Felix.pdf, oder als Mirror unter http://www.cedix.de/VorlesMirror/Band2/OSGI.pdf.

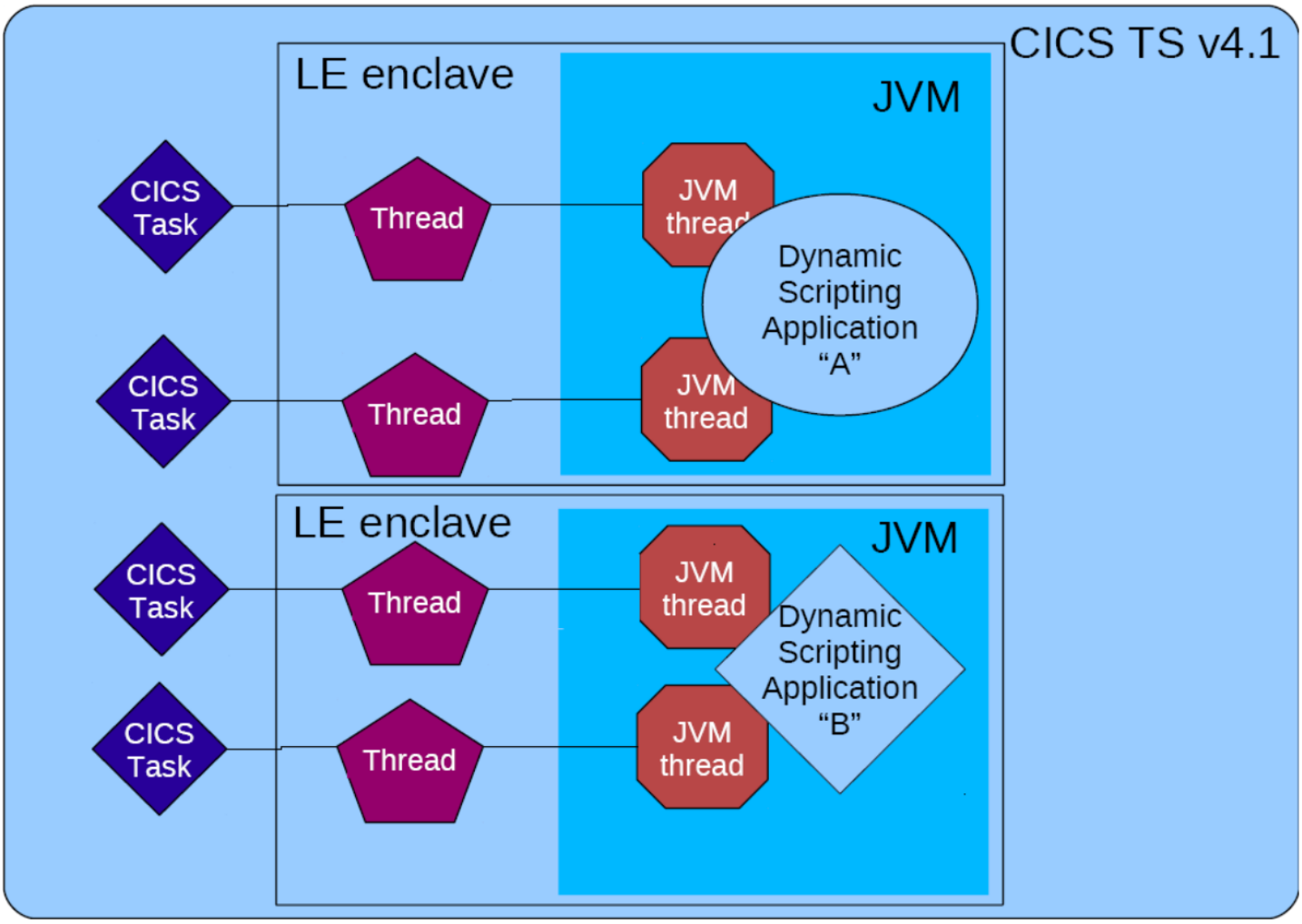


Abb. 19.3.14
Multiple JVMs mit Multiple Threads pro JVM

Es ist möglich, mehrere JVM Server in einem einzigen CICS Adressenraum auszuführen, genauso wie bei der CICS JVM Pool Architektur . Eine praktische Grenze ist etwa 20 JVMs innerhalb eines CICS Adressenraums.

19.3.12 Facit

Wie vollständig ist die Isolierung der Threads innerhalb des JVM Servers ?

Besser als alles, was es vorher gab, aber nicht so vollständig, wie man es gerne hätte.

Robert Harbach hat in seiner Master Thesis: "CICS JVM Server Application Isolation", März 2012, eine detaillierte Untersuchung der JVM Server Isolationseigenschaften durchgeführt. Während die Einführung des OSGi-Frameworks viele Probleme löst, müssen eine Reihe von Isolation Fragen noch geklärt werden. Eine sorgfältige Programmierung kann diese Probleme umgehen. Das Entwicklungsrisiko wächst jedoch mit der Größe und Komplexität einer neuen CICS Java-Anwendung.

Die Master Thesis kann heruntergeladen werden unter

http://www.cedix.de/VorlesMirror/Band2/harbach.pdf

Es ist eine dringende Erfordernis, dass eine zukünftige Version des Java Standards das Thread Isolationsproblem der JVM an der Wurzel anpackt, und eine zufriedenstellende Lösung anbietet. Dieses und verwandte Probleme haben dazu geführt, dass sich Java bisher nur für die Präsentationslogik von neuen Java Anwendungen durchgesetzt hat, während der Business Logik teil häufig immer noch in Cobol oder PL/I programmiert wird.

19.5 Weiterführende Information

Eine Übersicht über den Java Tranaction Service ist enthalten in
http://www.torsten-horn.de/techdocs/jee-transaktionen.htm.

Die erwähnte Diplomarbeit von Dr. Jens Müller: Anwendungs- und Transaktionsisolation unter Java, Mai 2005, können Sie herunterladen unter
http://www-ti.informatik.uni-tuebingen.de/~spruth/DiplArb/jmueller.pdf

Siehe auch
http://www.cedix.de/VorlesMirror/Band2/javaisol.pdf

Eine weitere Arbeit zum Thema Java Isolation finden Sie unter
http://www.cedix.de/VorlesMirror/Band2/JavaIsol05.pdf

Die erwähnte Masterarbeit von Robert Harbach: "CICS JVM Server Application Isolation", March 2012, können Sie herunterladen unter
http://www.cedix.de/VorlesMirror/Band2/harbach.pdf

Siehe auch
http://www.cedix.de/VorlesMirror/Band2/GIIS16-1.pdf

Ein Oracle Tutorial finden Sie unter
http://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

Eine Video Aufnahme einer Vorlesung zum Thema Java Transaction Management ist zu finden unter
http://freevideolectures.com/Course/2979/Java-EE-Programming-Spring-2012/12

Ein Vortrag „Understanding, Monitoring and Managing z/OS Enclaves" ist zu finden unter
http://www.cedix.de/VorlesMirror/Band2/EnclaveWLM.pdf

Eine gute Übersicht über verteilte Transaktionsverarbeitung ist zu finden unter
http://publib.boulder.ibm.com/infocenter/txformp/v7r1/index.jsp?topic=/com.ibm.cics.tx.doc/concepts/c_xopen_intface_fr_accsg_res_mgrs.html

20. Service Oriented Architecture

20.1 XML und DB2

20.1.1 Extensible Markup Language

Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. XML wird bevorzugt für den Austausch von Daten zwischen unterschiedlichen IT-Systemen eingesetzt, speziell über das Internet.

Die vom World Wide Web Consortium (W3C) herausgegebene XML-Spezifikation definiert eine Metasprache, auf deren Basis durch strukturelle und inhaltliche Einschränkungen Anwendungs-spezifische Dialekte definiert werden können. Diese Einschränkungen werden durch zwei alternative Schemasprachen ausgedrückt. Mit Hilfe Der Schemasprachen DTD oder XML-Schema (XSD) können unterschiedliche XML Dialekte für spezielle Anwendungen definiert werden. Bei der Verarbeitung von Textdokumenten wird meistens DTD eingesetzt. Für die Datenverarbeitung, besonders für Web Services, wird in der Regel XSD benutzt um den verwendeten XML Dialekt zu beschreiben.

XSL Transformation, kurz XSLT, ist eine Programmiersprache zur Transformation von XML-Dokumenten. Hiermit ist eine Übersetzung von einem mittels DTD oder XSD definierten XML Dialekt in einen anderen Dialekt möglich.

XSLT baut auf der logischen Baumstruktur eines XML-Dokumentes auf und dient zur Definition von Umwandlungsregeln. XSLT-Programme, sogenannte XSLT-Stylesheets, sind dabei selbst nach den Regeln des XML-Standards aufgebaut.

Die Stylesheets werden von spezieller Software, den XSLT-Prozessoren, eingelesen, die mit diesen Anweisungen ein oder mehrere XML-Dokumente in das gewünschte Ausgabeformat umwandeln.

Ein XML-Dokument besteht aus Textzeichen, in der Regel im ASCII-Code oder Unicode (meist UTF-8) , und ist damit durch einen menschlichen Benutzer lesbar. Binärdaten enthält es per Definition nicht.

20.1.2 HTML und XML

HTML wurde entworfen, um Daten visuell in einem Web Browser darzustellen. Für ein Anwendungsprogramm würde es meistens schwierig sein, allgemein Daten aus einer HTML Seite zur Weiterberarbeitung zu extrahieren, da es nicht weis, wie die Daten in der HTML Seite zu interpretieren sind.

Bei XML besteht eine absolute Trennung von Inhalt und Formatierung. Die Tags beschreiben den Inhalt, so dass geeignete Programme den Inhalt entsprechend den Vorschriften verarbeiten können, die dem Tag irgendwie zugeordnet werden. XML-Dokumente sind nicht dazu bestimmt, direkt ausgegeben zu werden, sondern werden mit geeigneten Programmen weiterverarbeitet.

Da die XML-Tags inhaltlich definiert sind, können diese Programme zur Weiterverarbeitung ganz unterschiedliche Themen betreffen:

- **Formatierung und Ausgabe als HTMl z.B. suchen in Such-Maschinen**
- **Eintrag in Datenbanken z.B. Output mit maschineller Verarbeitung**
- **inhaltlich korrekte Archivierung z.B. Umformatieren auf beliebige Datenformate**

XML kann weiterhin semantische Information in ein Dokument einbauen.

Was ist Semantische Information (oder meaning)? Nehmen Sie als Beispiel eine Buchbeschreibung in www.amazon.de. Ein Leser (oder ein XML Dokument) kann entscheiden, ob der Ausdruck "Braun" sich auf die Farbe des Einbandes oder den Namen des Verfassers bezieht. Eine HTML-basierte Web Search Engine kann das nicht, weil für diese Unterscheidung nicht genügend semantische Information in der HTML Seite vorhanden ist.

Eine spezielle XML Version, XHTML, vereinigt die Möglichkeiten von HTML und XML.

20.1.3 Nachrichtenaustausch zwischen unterschiedlichen Architekturen

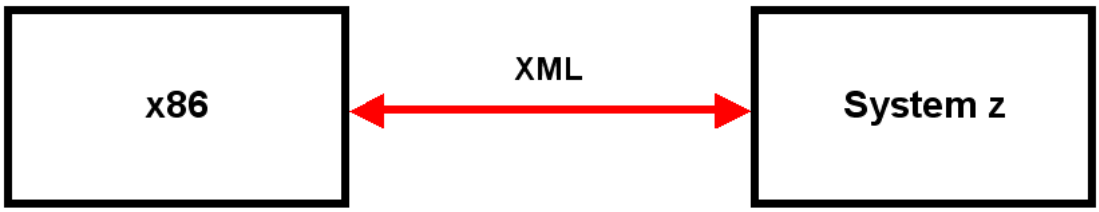


Abb. 20.1.1
Kommunikation mit XML kodierten Nachrichten

XML kann Anwendungen voneinander isolieren. Es stellt ein universelles Datenformat bereit, welches für Input und/oder Output zwischen beliebigen Anwendungsprogrammen benutzt werden kann. Da XML Daten selbst-definierend sind, erhält das empfangene Anwendungsprogramm alle Information die es braucht, um die übermittelten Daten zu verarbeiten, ohne Wissen über das sendende Anwendungsprogramm. Es ist damit eine Alternative zu der Interface Definition Language (IDL) im klassischen oder im CORBA RPC.

Der Hauptvorteil von XML ist, dass es den Kontext mit den Daten überträgt.

Damit ist XML für die Integration von Geschäftsprozessen und deren Anwendungen innerhalb eines Unternehmens gut geeignet.

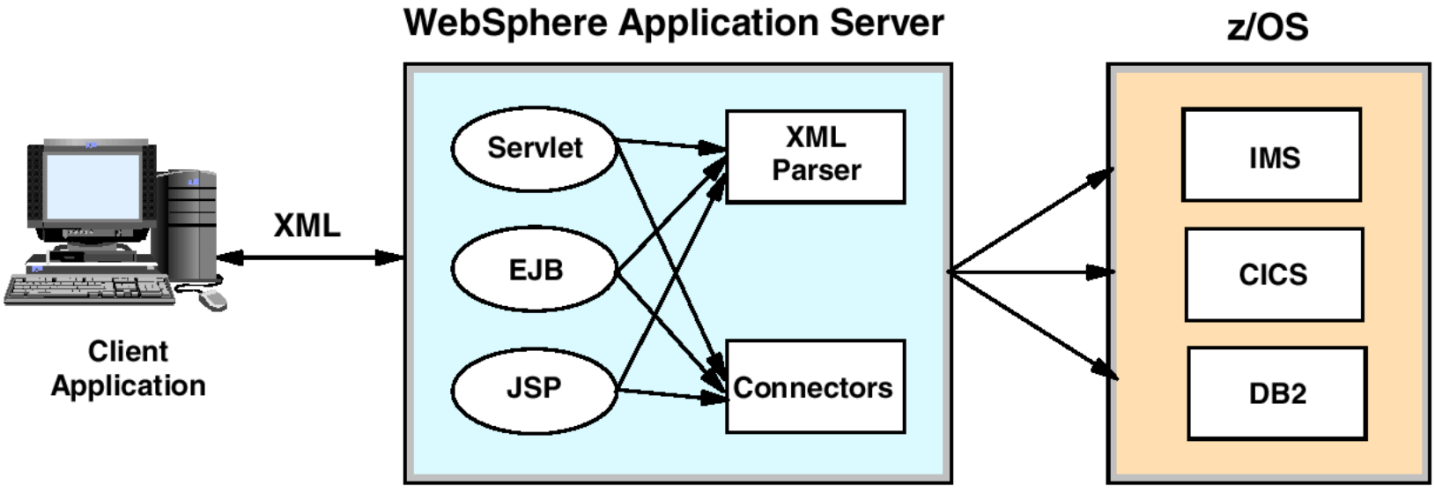


Abb. 20.1.2
Parsing einer XML Nachricht

Abb. 20.1.2 zeigt eine Client/Server Operation mittels XML.

Die Client Anwendung übermittelt Daten im XML Format an den WebSphere Application Server. Die Daten können von einer Serverkomponente wie einem Servlet, einer Java Server Page (JSP), einer Java Bean oder einer Enterprise Java Bean (EJB) entgegen genommen werden. Ein "XML Parser" extrahiert die relevante Information von der empfangenen XML Nachricht.

Als Ergebnis der Parsing Operation kann z.B. eine IMS Message, eine CICS COMMAREA, oder eine JDBC Request entstehen, die an die Business Logik des entsprechenden Backend Systems weitergeleitet wird.

20.1.4 XML und Datenbanken

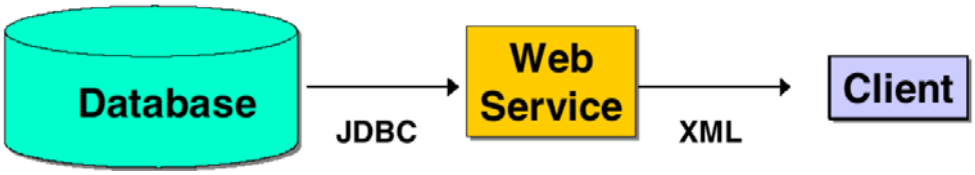


Abb. 20.1.3
Abfrage von XML Daten

Es existieren mehrere Gründe, XML mit einer Datenbank (z.B. DB2 oder IMS) zu speichern. Besonders gebräuchlich ist es, auf Daten im XML Format zuzugreifen, die in einer existierenden Datenbank untergebracht sind. Nehmen wir z.B. an, eine Datenbank enthält Information über Aktienkurse. Ein Web Service könnte auf Anfrage den derzeitigen Aktienkurs als ein XML Dokument wiedergegeben.

Abb. 20.1.4
Speicherung von XML Daten

Ähnlich kann ein Klientenprogramm Daten in der Form eines XML Dokuments an eine Datenbank übertragen. Ein Versicherungsvertreter könnte die Daten für eine neue Versicherung in der Form eines XML Dokumentes eingeben. Innerhalb des Versicherungsunternehmens würde ein Anwendungsprogramm die Daten aus dem XML Dokument extrahieren und in einer Datenbank (z.B. DB2 oder IMS) speichern.

Mit XML entsteht ein neues Datenmodell in der Welt der Datenbanken, und zwar zusätzlich zu den existierenden hierarchischen, relationalen, objekt-orientierten und anderen Datenmodellen. Ein XML-Dokument speichert beliebige Daten nach den Regeln eines bestimmten XML Datenmodells ab, ist also so etwas wie eine Datenbank, welche die eigentlichen Rohdaten enthält. Das XML Datenmodell ist ein Baum mit Ästen, Zweigen und Blättern, wobei die Blätter die eigentlichen Daten speichern.

So, wie mit SQL aus einer DB2 Datenbank Aggregationen und Verdichtungen extrahiert werden, können mit dem Werkzeug XSLT aus demselben XML-Dokument verschiedenartigste Outputs erzeugt werden.

- **Eine XML enabled Database verwendet ein non-XML Datenmodell und bildet ab (maps) bei jedem Zugriff Instanzen dieses Modells (z.B. SQL Tables mit den gewünschten Daten) auf eine XML Darstellung (Instanzen des XML Datenmodells).**
- **Eine XML native Database benutzt das XML Datenmodell direkt (speichert Daten im XML Format).**

Bei einer XML-enabled Datenbank existiert allerdings eine Abhängigkeit von den Eigentümlichkeiten des Datenbank-Systems und der darunter liegenden Betriebssystem-Plattform. SQL Datenbanken sind auf Grund von proprietären Erweiterungen des SQL Standards durch Hersteller wie Oracle oder IBM nicht miteinander kompatibel. Bei einer native XML/XSLT Datenbank sind dagegen sowohl die Daten (das XML-Dokument) als auch die Transformationsregeln (eine oder mehrere XSLT-Dateien) reine Textdateien, die mit jedem Standard-Editor auf jedem Betriebssystem bearbeitet werden können.

Einfache bzw. validierende Parser sowie Parser, die zusätzlich das XSLT-Dokument laden und es auf das XML-Dokument anwenden, gibt es ebenfalls auf jeder Plattform. Damit ist die eigentliche Entwicklung eigener Dokument Type Definitionen, das Schreiben der gewünschten XML-Dokumente sowie die Entwicklung des XSLT-Codes möglich, ohne Betriebssystem-spezifische Besonderheiten beachten zu müssen.

20.1.5 Zwei unterschiedliche Arten ein XML Dokument zu speichern

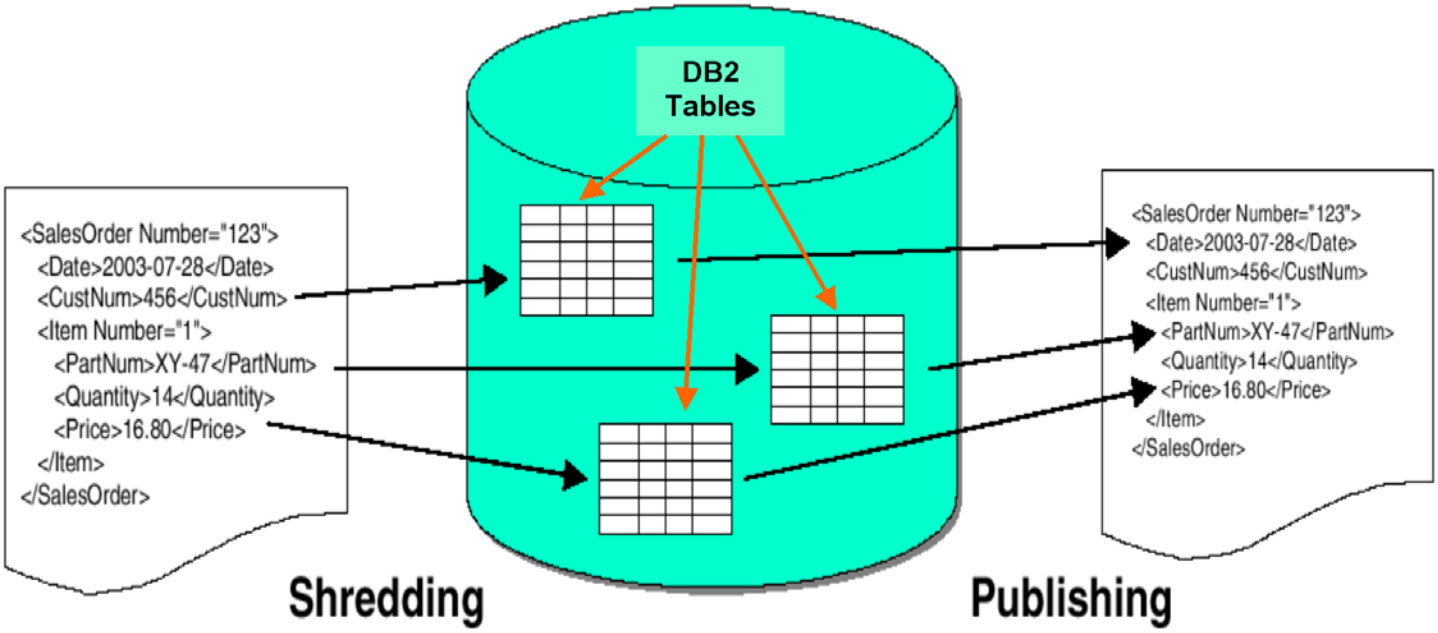


Abb. 20.1.5
Publishing und Shredding in einer XML – enabled DB2 Datenbank

Angenommen eine XML enabled Database, z.B. DB2. Ein Prozess, der Daten aus einer Datenbank extrahiert und daraus ein XML Dokument erzeugt wird als publishing oder composition bezeichnet. Der umgekehrte Prozess (Daten aus einem XML Dokument extrahieren und in einer DB2 Datenbank speichern) wird als shredding oder decomposition bezeichnet.

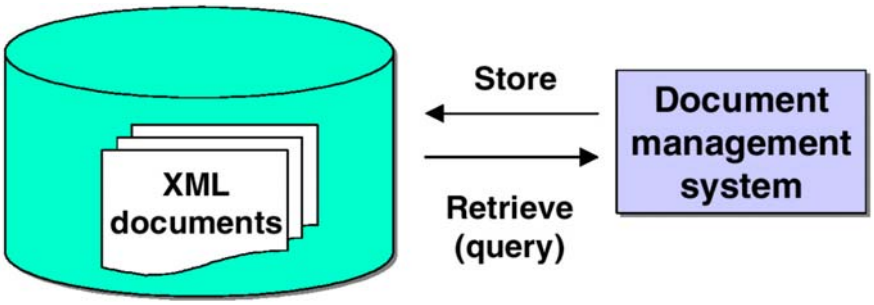


Abb. 20.1.6
Eine native XML Datenbank speichert Informationen im XML Format

Die XML Dokumente in Abb. 20.1.5 sind data-centric . In anderen Worten, die Dokumente enthalten Daten mit einer regulären Struktur.

Andersartig sind document-centric Dokumente. Beispiele sind Endbenutzer Dokumente, Vertriebsbroschüren und Webseiten. Document-centric Dokumente haben eine irreguläre Struktur. Hier kann eine native XML Database nützlich sein.

Verwaltung und Abfrage ist die am häufigsten benutzte Funktion bei document-centric Dokumenten. Beispielsweise erfordern große Projekte wie z.B. die Entwicklung eines neuen Flugzeuges einen sehr großen Umfang an Endbenutzer Dokumentation. Die Verwaltung dieser Dokumente in einer Datenbank ist eine kritische Aufgabe. Mittels XML (und DTD) ist es leicht, neue Dokumente aus Fragmenten bisheriger Dokumente zu erstellen. Strukturierte Abfragen über den Inhalt dieser Dokumente sind möglich.

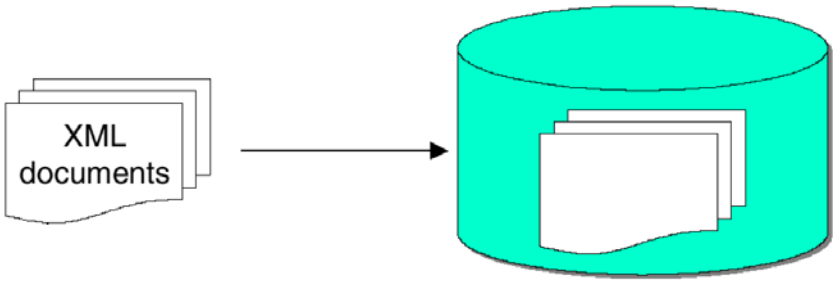


Abb. 20.1.7
Speichern von XML Dokumenten in einer Datenbank

Wir benutzen in dem folgenden Beispiel das in Abb. 2.1.8 dargestellte Sales Order Dokument.

```
<SalesOrder Number="123">
   <OrderDate>2003-07-28</OrderDate>
   <CustomerNumber>456</CustomerNumber>
   <Item Number="1">
      <PartNumber>XY-47</PartNumber>
      <Quantity>14</Quantity>
      <Price>16.80</Price>
   </Item>
   <Item Number="2">
      <PartNumber>B-987</PartNumber>
      <Quantity>6</Quantity>
      <Price>2.34</Price>
   </Item>
</SalesOrder>
```

Abb. 20.1.8
XML Version des Sales Order Dokuments

Es existieren zwei unterschiedliche Arten, um das XML Sales Order Document zu speichern:

1. **XML enabled Datenbank**
2. **XML native Datenbank**

20.1.6 Speichern in einer XML – enabled DB2 Datenbank

Number	Date	Customer
123	2330-07-28	456
...	...	...

DB2 sales order table

SONumber	Number	PartNumber	Quantity	Price
123	1	"XY-47"	14	16.80
123	2	"B-987"	6	2.34
...	...	...	...	...

Abb. 20.1.9
XML- enabled DB2 Version des Sales Order Dokumentes

In der XML- enabled DB2 Version des in Abb. 20.1.8 dargestellten Sales Order Dokumentes werden die zwei in Abb. 20.1.9 gezeigten DB2 Tabellen benötigt. Es ist kein XML innerhalb der DB2 Datenbank sichtbar. Das XML Dokument existiert nur außerhalb der Datenbank. Es kann jederzeit aus den Daten in der Datenbank erstellt werden. Ebenso kann es als Quelle für neu in der Datenbank zu speichernde Daten benutzt werden.

Ein Schema ist eine formale Beschreibung der Struktur von Daten. Bei einer XML-enabled Database stimmt das Datenbank Schema mit dem XML Schema überein. Ein unterschiedliches XML Schema wird für jedes Datenbank Schema benötigt.

Literatur: XML for DB2 Information Integration, July 2004, SG24-6994-00.

20.1.7 Speichern in einer native XML DB2 Datenbank

Als Alternative kann das XML Dokument selbst in einer native XML DB2 Datenbank gespeichert werden. Hierzu werden vier DB2 Tabellen benötigt, die in Abb. 20.1.10 – 20.1.13 dargestellt sind.

ID	Name
34	"SalesOrder123.xml"

Abb. 20.1.10
1. Documents table

DocumentID	ElementID	ParentID	Name	OrderInParent
34	1	NULL	"SalesOrder"	1
34	2	1	"OrderDate"	1
34	3	1	"CustomerNumber"	2
34	4	1	"Item"	3
34	5	4	"PartNumber"	4
34	6	4	"Quantity"	2
34	7	4	"Price"	3
34	8	1	"Item"	4
34	9	8	"PartNumber"	1
34	10	8	"Quantity"	2
34	11	8	"Price"	3

Abb. 20.1.11
2. Elements table

DocumentID	AttributeID	ParentID	Name	Value
34	1	1	"Number"	123
34	2	4	"Number"	1
34	3	8	"Number"	2

Abb. 20.1.12
3. Attributes table

DocumentID	TextID	ParentID	Value	OrderInParent
34	1	2	"2003-07-28"	1
34	2	3	"456"	1
34	3	5	"XY-47"	1
34	4	6	"14"	1
34	5	7	"16.80"	1
34	6	9	"B-987"	1
34	7	10	"6"	1
34	8	11	"234"	1

Abb. 20.1.13
4. Text table

In diesem Fall ist XML innerhalb der Datenbank sichtbar. Es werden vier DB2 Tabellen benötigt. Aus diesen vier Tabellen kann das in Abb. 20.1.8 dargestellte Sales Order Dokument wiederhergestellt werden.

Die Datenbank enthält Information wie Element Type und Attribute Namen. Ein einziges Datenbank Schema reicht für das Speichern aller XML Dokumente aus. In anderen Worten, das Datenbank Schema modelliert XML Dokumente, nicht die Daten in diesen Dokumenten. Datenbanken, welche XML auf diese Art benutzen werden als native XML Database bezeichnet.

20.1.8 IMS

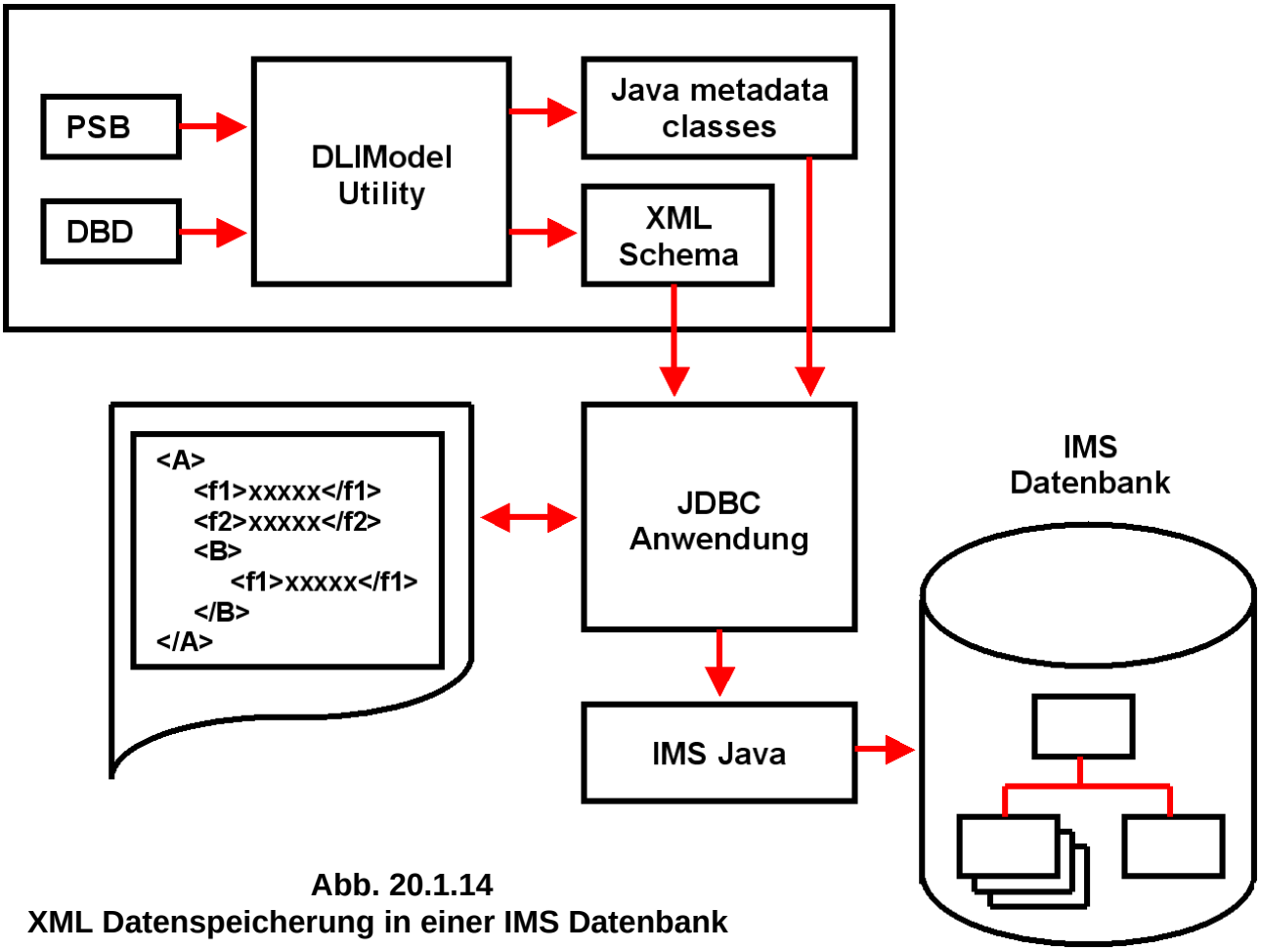


Abb. 20.1.14
XML Datenspeicherung in einer IMS Datenbank

DB2 und IMS sind die beiden wichtigsten Mainframe Datenbanken. IMS ist nicht relational; Daten werden in der Form von hierarchischen Bäumen gespeichert (siehe Band 1, Abschnitt 3.4.3). Es existieren viele Situationen, wo das Speichern und Auslesen von Daten mit IMS schneller abläuft als mit DB2. Auf der anderen Seite ist die Flexibilität von DB2 viel besser.

Da XML und IMS Datenbanken beide hierarchisch sind, ist es besonders einfach, XML Dokumente in einer IMS Datenbank zu verwalten. Beispielsweise kann man:

- **XML Dokumente aus allen Arten von existierenden IMS Datenbanken erzeugen. Das ist z.B. nützlich für Business-to-Business on Demand Transaktionen, oder für den Datenaustausch innerhalb einer Organisation.**
- **Eintreffende XML Dokumente innerhalb einer IMS Datenbank abspeichern. XML Dokumente werden decomposed (shredded) abgespeichert. Hierzu werden eintreffende Dokumente geparsed; die Datenelemente und Attribute werden in Feldern der IMS-Segmente als normale IMS Daten gespeichert. Dieses Verfahren ist für Data-centric, weniger für Document-centric XML Dokumente geeignet.**

Ein IMS PSB, (Program Specification Block) spezifiziert die Datenbank, auf welche das Anwendungsprogramm zugreift. Ein IMS DBD ist der Database Description Control Block.

20.2 Web Services

20.2.1 Was ist ein Web Service?

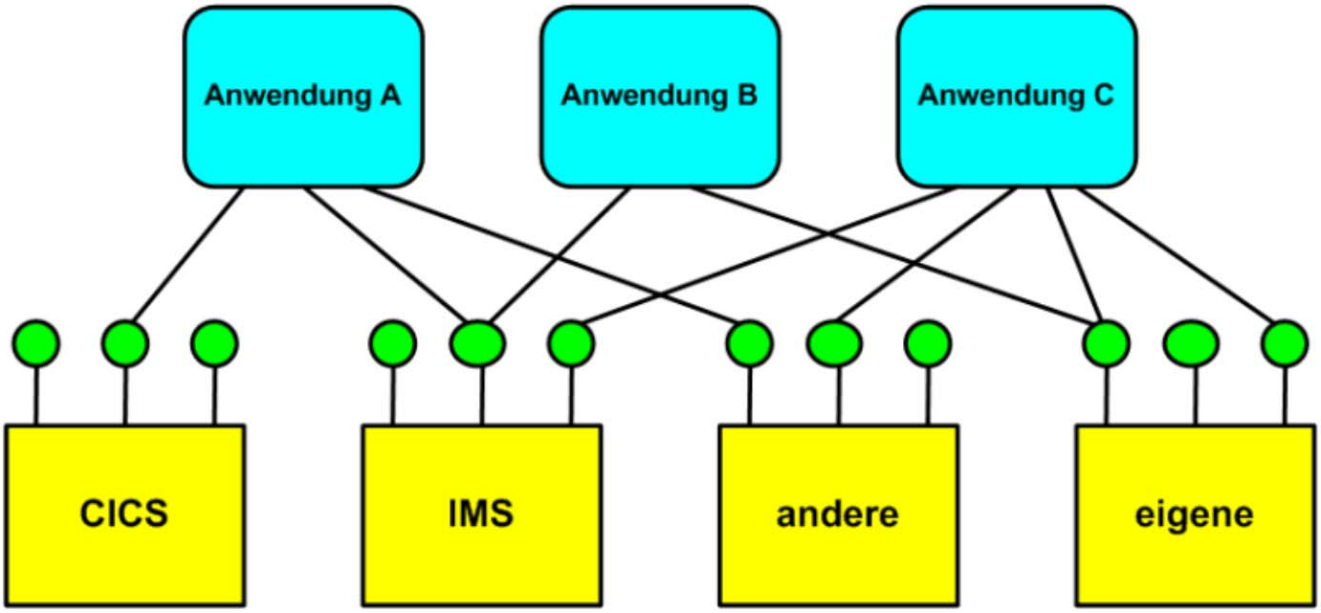


Abb. 20.2.1
Web Service als Lösung für das Application Integration Problem

Frage: Wie können in einem Unternehmen die unterschiedlichen Klientenanwendungen (unterschiedliche Plattformen, unterschiedliche Betriebssysteme, unterschiedliche Sprachen) flexibel auf einen größeren Satz Serveranwendungen zugreifen ?

Die moderne Antwort ist, indem die Kommunikation zwischen Klient und Server als „Web Service“ implementiert wird.

Im Prinzip ist dies ein Service, den man über ein Standard Internet Protokoll (wie z.B. SMTP, FTP und andere) aufrufen kann, und der XML für die Beschreibung von Daten, Nachrichten, Schnittstellen usw. benutzt. Gemeint ist aber fast immer ein Service der über http mit dem Simple Object Access Protocol (SOAP) aufgerufen wird. Die Schnittstelle des Services wird hierbei mittels der Web Service Definition Language (WSDL) beschrieben.

Von der W3C Web Services Architecture Working Group stammt die folgende Definition für Web Services:

“A Web Service is a software application identified by a Uniform Resource Identifier *(*URI*)* , whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.

A uniform Resource Identifier (URI) is a compact string of characters used to identify or name a resource on the Internet.”

Beispiel Yahoo Portal

Die meisten Yahoo Dienste kommen in Wirklichkeit von anderen Web Sites: Reisen, Wetter, Landkarten, oder Web Suche mit Hilfe von Google. Diese Dienste sind in das Yahoo Portal als eigene Dienste integriert.

20.2.2 Uniform Resource Identifier

Ein Uniform Resource Identifier (Abk. URI; engl. für „einheitlicher Bezeichner für Ressourcen“) ist ein Identifikator und besteht aus einer kompakten Zeichenkette, die zur Identifizierung einer abstrakten oder physischen Ressource dient. URIs werden zur Bezeichnung von Ressourcen (wie Webseiten, sonstigen Dateien, Aufruf von Web Services, aber auch z. B. E-Mail-Empfängern) im Internet und dort vor allem im WWW eingesetzt.

Beispiele für Uniform Resource Identifier sind:

http://www.ietf.org/rfc/rfc2396.txt (URL)

ftp://ftp.is.co.za/rfc/rfc1808.txt

ldap://[2001:db8::7]/c=GB?objectClass?one

mailto:John.Doe@example.com

telnet://192.0.2.16:80/

tel:+0049-341-97-32211

Eine URL ist ein Spezialfall einer URI.

URIMAP Definitionen sind Resource Definitionen, welche die URIs von HTTP oder Web Service Requests abbilden. Sie enthalten Information, wie diese Requests abzuarbeiten sind.

Im Zusammenhang mit Web Services besteht ein URI häufig aus einer URL plus Port Nr.

20.2.3 Der modernste Remote Procedure Call

Nach dem Sun RPC, dem DCE RPC, CICS DPL, Corba und RMI ist ein Web Service die modernste Ausprägung eines Remote Procedure Calls.

Hierbei definiert SOAP die Implementierung des eigentlichen RPC. WSDL ist die Beschreibung der Schnittstelle. Das Besondere eines Web Service besteht darin, dass

- **Nachrichten zwischen Klient und Server im XML Format übertragen werden,**
- **die Beschreibung der Schnittstelle eines Services ebenfalls im XML Format (in WSDL) erfolgt.**

Ähnlich wie der CORBA Naming Service und der Java JNDI (oder Java Registry) verfügt auch der Web Service über einen eigenen Naming und Directory Service, als „Universal Description, Discovery and Integration“ (UDDI) bezeichnet.

UDDI ist jedoch umstritten und wird nur begrenzt eingesetzt. In vielen Fällen verwenden Web Services vorhandene Directory Dienste wie LDAP oder Microsoft Active Directory, die praktisch den gleichen Funktionsumfang bieten.

Web Services erlauben vielfach eine „quick and dirty“ Implementierung eines Prototypen. Geht man über das Prototyp Stadium hinaus, wächst die Komplexität einer Implementierung oft dramatisch an. Was ursprünglich fehlte:

- **Sicherheit - HTTPS kann benutzt werden, ist aber unabhängig vom Web Service Mechanismus,**
- **skaliert schlecht,**
- **Transaktionssteuerung, Flusssteuerung,**

wurde zwischenzeitlich durch Erweiterungen des Web Service Standards verbessert, auf Kosten der Komplexität.

Das Simple Object Access Protocol (SOAP) ist weder simple noch Objekt-orientiert. Deshalb benutzt man heutzutage nur noch den Begriff SOAP, nicht aber mehr den Begriff "Simple Object Access Protocol“.

20.2.4 Web Services Technologien

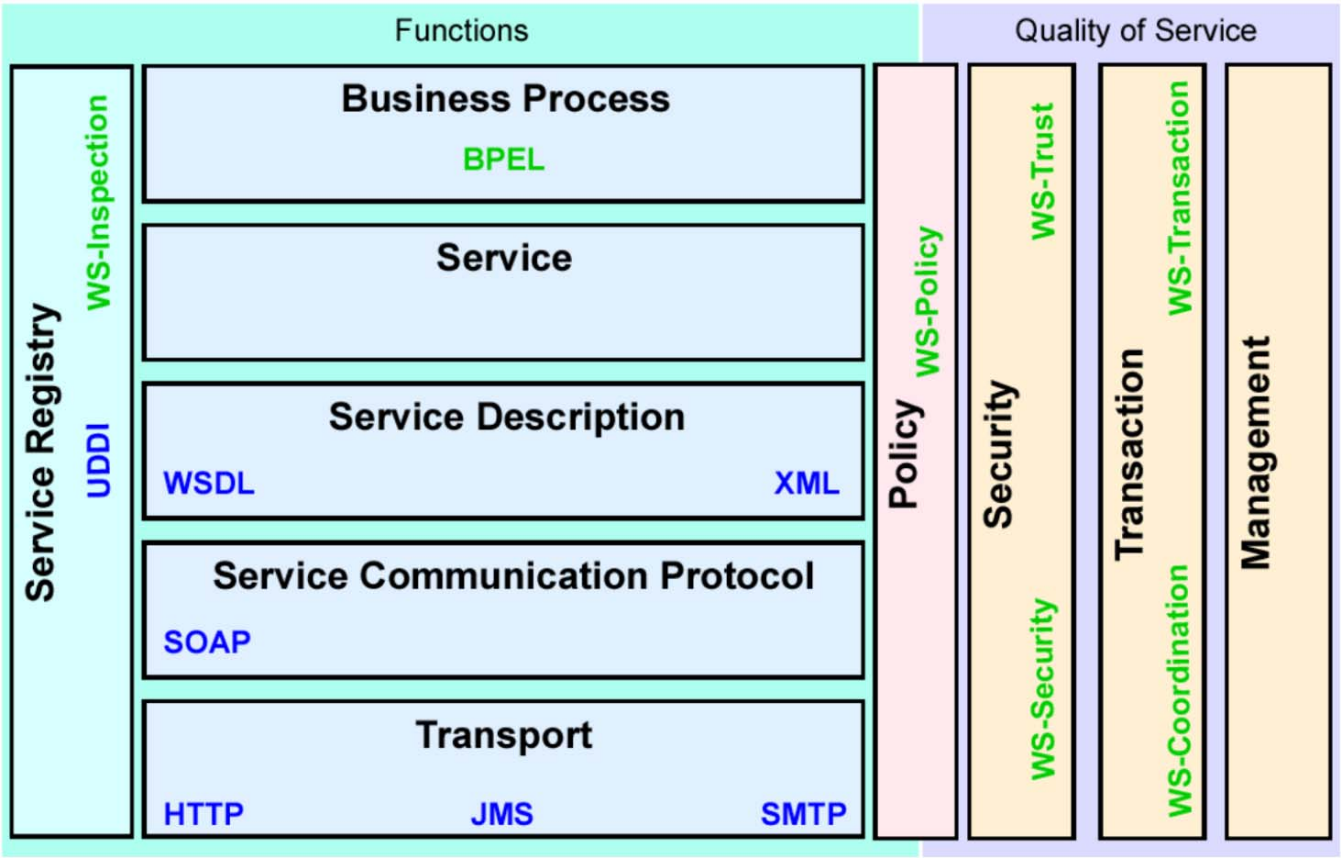


Abb. 20.2.2
Bestandteile des Web Services Architecture Stack

Über die Jahre entwickelte sich Web Services zu einer komplexen, vielschichtigen Architektur. Die ursprüngliche Konfiguration bestand aus SOAP, WSDL und UDDI und spezifizierte http als Übertragungsprotokoll.

SOAP ist ein Remote Procedure Call (RPC) Protokoll. Es benutzt häufig HTTP für den Transport (Alternativen sind alle Protokolle, die Text übertragen können, z.B. FTP, MQSeries). XML beschreibt das Format der übertragenen Daten. Es ersetzt damit z.B. die IDL, die beim klassischen RPC oder bei CORBA benutzt wird, oder die Schnittstellen-Beschreibung bei Java RMI. SOAP ist unabhängig von der verwendeten Programmiersprache und dem jeweiligen Betriebssystem.

WSDL ist eine XML- Beschreibung der Schnittstellen-Definitionen eines Web Service, vergleichbar mit der Schnittstellen-Beschreibung bei Java RMI. WSDL beschreibt Formate der Anforderungs- und Antwort- Nachrichtenströme, mit denen Funktionsaufrufe an andere Programm-Module abgesetzt werden.

UDDI ist ein XML Dienstekatalog (vergleichbar mit einer XML Version von LDAP). UDDI stellt ein Verzeichnis von Adress- und Produktdaten sowie Anwendungs-Schnittstellen der verschiedenen Web Service Anbieter dar.

Die Web Services Entwicklung begann als eine IBM - Microsoft Kooperation, und erhielt breite Unterstützung in der Industrie. Spätere Erweiterungen lösten das Firewall - HTTP Problem, verbesserten Sicherheit und Reifegrad, und fügten Transaktionsdienste hinzu.

Web Services bilden einen Baustein für eine **Service-Orientierte Architektur** (SOA), siehe Abschnitt 20.4.

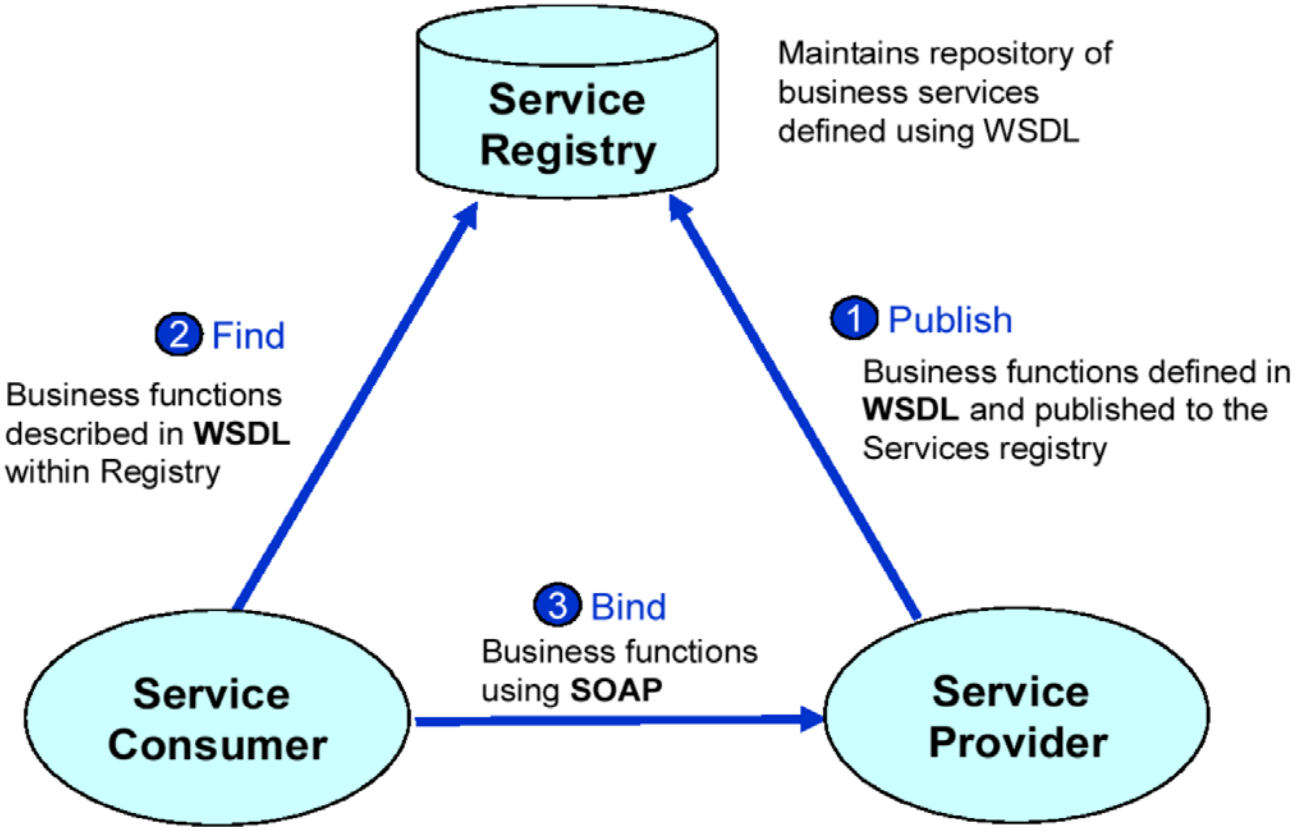


Abb. 20.2.3
Vorgehensweise

1. Ein Service Provider (Server) erstellt einen neuartigen Web Service und publiziert diese Tatsache durch einen Eintrag (Reklame) in einem Verzeichnis, dem Registry. UDDI ist der Registry Standard für Web Services; es können aber (und werden häufig) auch andere Registry Implementierungen benutzt werden.
2. Ein Interessent (Klient, Service Consumer) findet einen ihn interessierenden Service im Registry und liest die Beschreibung und Zugriffsdaten.
3. Der Klient greift auf den Service zu.

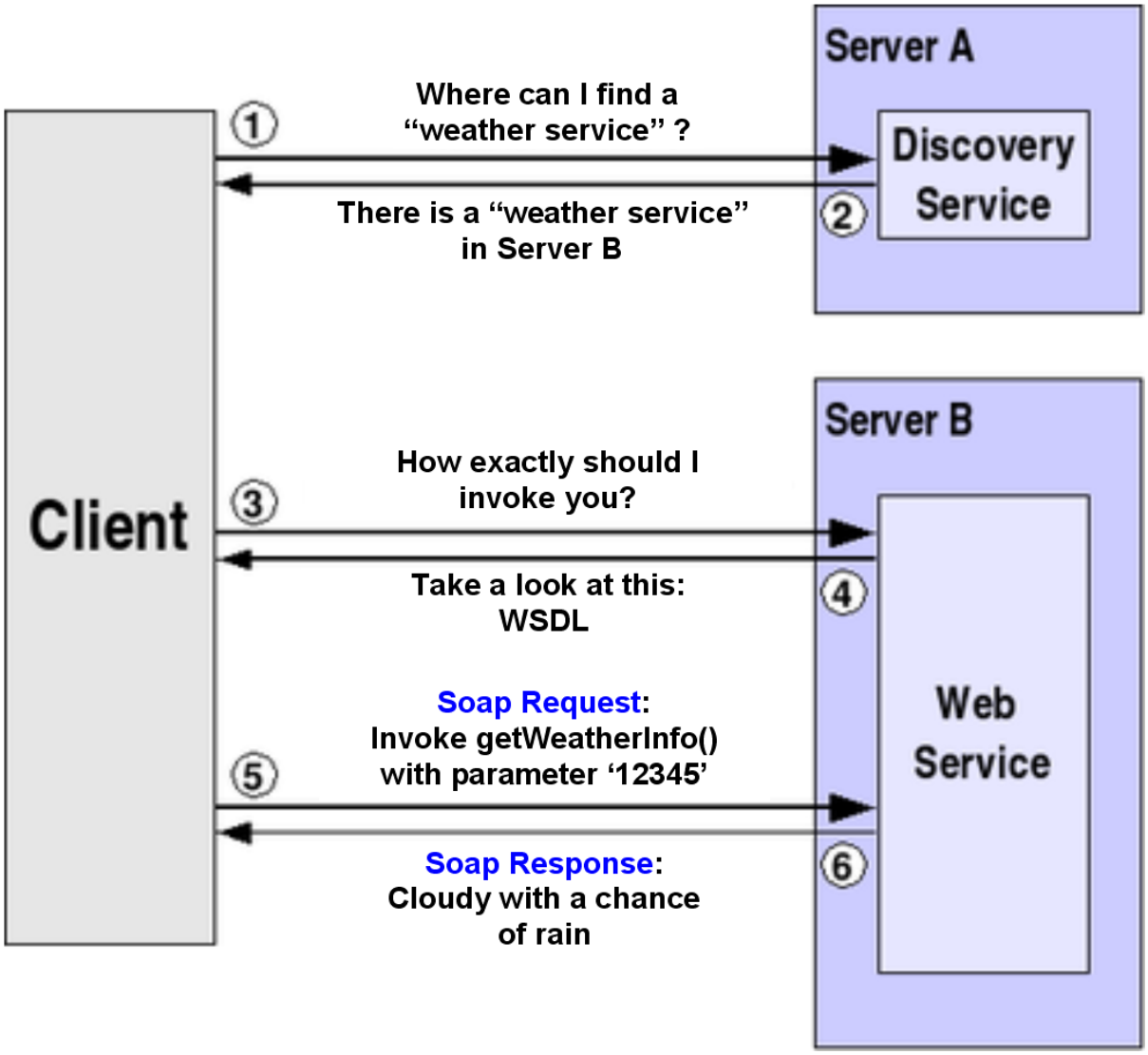


Abb. 20.2.4
Beispiel: Aufruf eines Web Service

1. Ein Klient, der an einem Wetterbericht interessiert ist, greift auf einen UDDI Server zu (Server A).
2. Der UDDI Server (Registry) schlägt dem Klienten den Server B vor. Server B stellt Wetterberichte zur Verfügung.
3. Der Klient greift auf den WSDL Service des Servers B zu.
4. Die WSDL Antwort (im XML Format) enthält Anweisungen, wie man auf den Wetterdienst zugreift.
5. Der Klient benutzt das SOAP Protokoll um die Wetterinformation abzufragen.

20.2.5 SOAP

SOAP ist ein zustandsloses Protokoll (wie http), welches zum Austausch von One-way Nachrichten zwischen SOAP Klienten (Sender) und SOAP Server (Empfänger) konzipiert ist. Mit SOAP lassen sich darüber hinaus komplexere Interaktionsformen (beispielsweise Request/Response oder Request/Multiple Response) anhand der Kombination von einfachen Nachrichten und zugrundeliegenden Transportprotokollen realisieren.

SOAP macht prinzipiell keine Aussagen über

- **die Semantik der zu transportierenden Daten,**
- **das Routing von Nachrichten,**
- **Verlässlichkeit des Transfers oder**
- **die Überwindung von Firewalls.**

Es stellt lediglich ein Framework zum Nachrichtenaustausch dar. Hierzu baut es auf vorhandenen Transport-Protokollen wie HTTP oder SMTP auf.

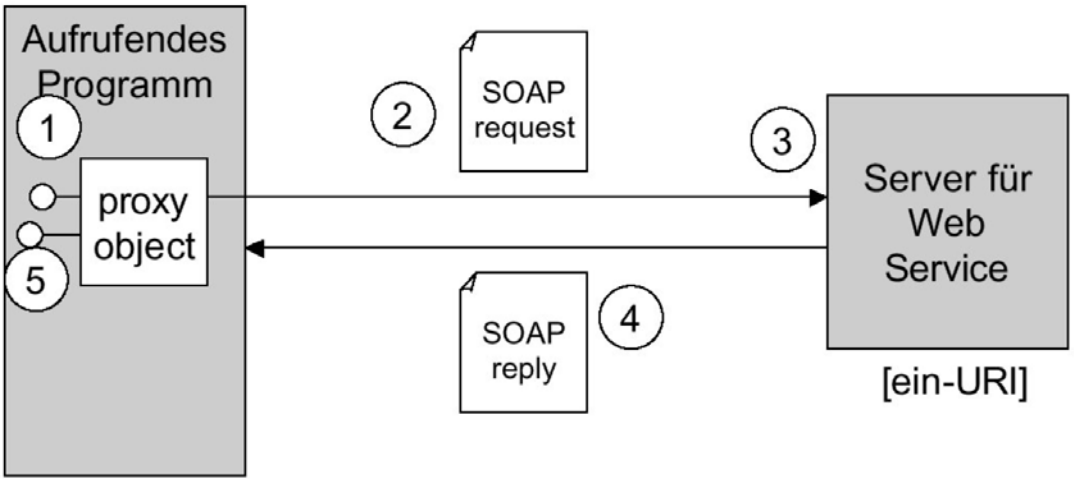


Abb. 20.2.6
SOAP Funktionsweise

Beim
SOAP Remote Procedure Call erfolgt der Aufruf des remote Web Service über eine HTTP Message Request. Die Antwort des Web Service Servers erfolgt über eine HTTP Message Response.

An Stelle einer IDL (Interface Definition Language) Beschreibung wird eine XML Beschreibung eingesetzt.

20.2.6 Klassischer Remote Procedure Call

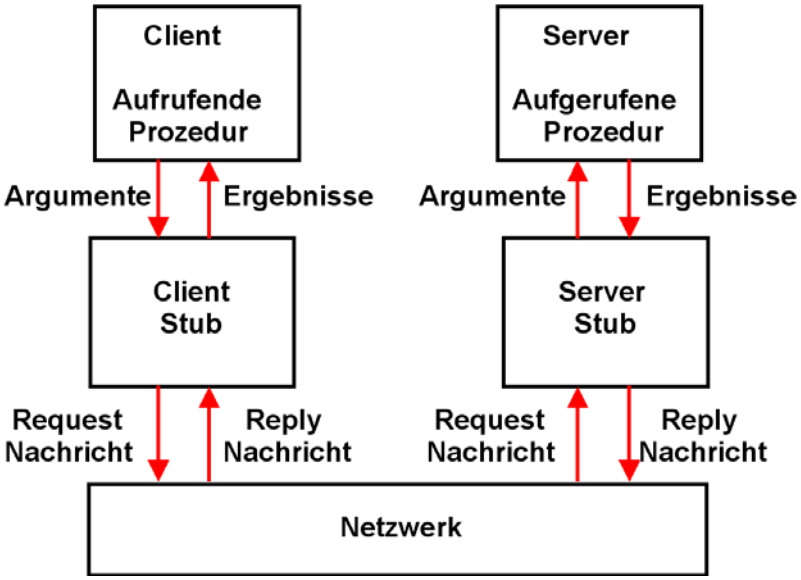


Abb. 20.2.7
Stubs (und Skeletons) des Remote Procedure Calls

Client und Server laufen als zwei getrennte Prozesse.

Die beiden Prozesse kommunizieren über Stubs. Stubs sind Routinen, welche Prozedur-Aufrufe auf Netzwerk RPC Funktionsaufrufe abbilden.

Ein Server-seitiges RPC Programm stellt den Klienten seine Dienste über eine Interface (Schnittstelle) zur Verfügung. es definiert seine Interface unter Benutzung einer Interface Definition Language (IDL).

Ein Stub Compiler liest die IDL Schnittstellen-Beschreibung und produziert zwei Stub Proceduren für jede Server Procedure (client side stub und server side stub).

Der klassische RPC benutzt die Bezeichnung Server Stub. Modernere RPCs verwenden statt dessen die Bezeichnung Skeleton. Die Bezeichnungen Server Stub und Skeleton sind austauschbar.

Stubs bzw. Skeletons implementieren eine RPC Runtime. Sie übersetzen Aufrufe der Client API in Sockets und dann in Nachrichten, die über das Netz übertragen werden.

Beim CICS Distributed Program Link (DPL) benötigt der Aufruf eines entfernten Systems lediglich dessen Name (TRID) und die Bezeichnung der COMMAREA. Eine IDL ist nicht erforderlich.

20.2.7 Web Services Remote Procedure Call

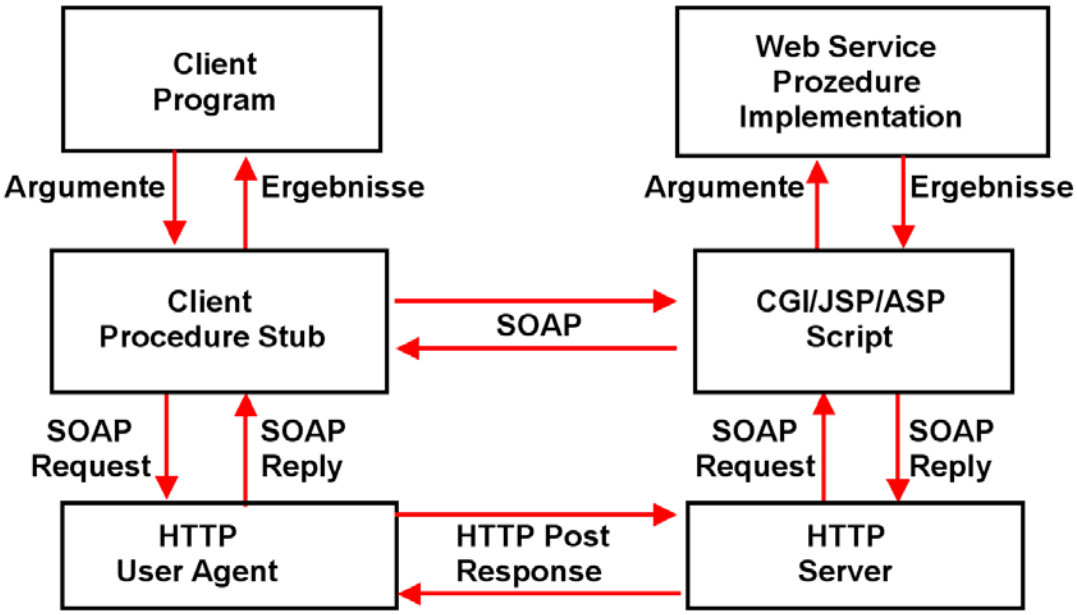


Abb. 20.2.8
Stubs (und Skeletons) des SOAP Remote Procedure Calls

Der Client Procedure Stub erzeugt eine SOAP Message, die an den Server Stub weitergegeben wird.

Client und Server Stubs sind häufig in Java geschrieben und benutzen z.B. Java Server Pages. Alternativen sind CGI Programme oder (im Falle von Microsoft Implementierungen) Active Server Pages. Aber auch Cobol ist auf der Server Seite häufig zu finden.

20.2.8 SOAP Vor- und Nachteile

Für die Nutzung entfernter Ressourcen und zum Aufruf entfernter Methoden gibt es mehrere klassische Ansätze. Ein einfaches Verfahren ist die Anfrage mit Parametern an einen Web-Server, der den Inhalt zurückliefert.

Zur Verteilung von Programmen haben sich CORBA, RPC, aber auch RMI bewährt. Web Services haben den Vorteil, dass das Protokoll HTTP verbreitet ist.

CORBA und RMI haftet der Nachteil an, dass ihre Kommunikation einen offenen Port erfordert, den Sicherheitsbeauftragte in einem Unternehmen nicht unbedingt tolerieren. Zudem sind CORBA/RMI Lösungen nicht wirklich einfach zu implementieren. Idealerweise verheiratet eine Technik beide Vorteile: http-Kommunikation und objektorientierte Konzepte. Web Servers benutzen HTTP und damit standardmäßig Port 80.

HTTP hat den Vorteil (und den Nachteil) dass es für Web-Seiten in der Regel keine Beschränkungen durch Firewalls gibt. Die beiden Kommunikationsparteien regeln mit SOAP einen entfernten Methodenaufruf, der die Parameter und Ergebnisse in XML kodiert und überträgt.

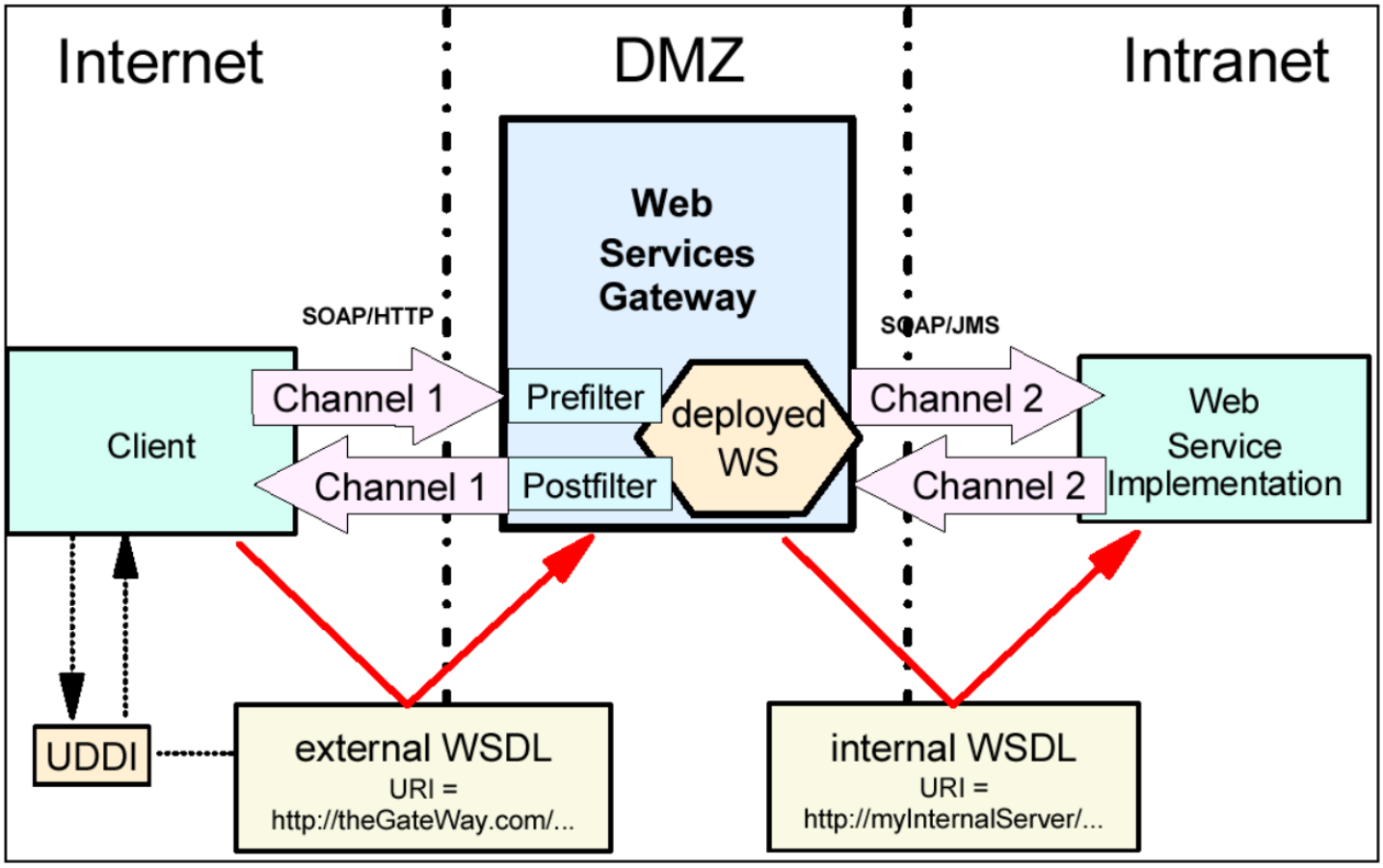


Abb. 20.2.5
Das Simple Object Access Protocol ist in Wirklichkeit alles andere als „simple“

SOAP war in der Anfangszeit recht populär, weil es im Vergleich zu existierenden RPC Alternativen die Implementierung einer „quick und dirty“ Feasibility Demonstration ermöglichte. Daher das Wort „Simple“ in der Bezeichnung Simple Object Access Protocol.

Dies geschah jedoch dadurch, dass man auf Funktionen verzichtete, die für Industry Strength Anwendungen erforderlich sind. Ein Beispiel ist die Benutzung von HTTP und Port 80, was den Internet Zugriff ermöglicht und den Enterprise Firewall umgeht. Für eine quick und dirty Feasibility Demonstration ist das attraktiv. Nach der Erweiterung um Industry Strength Eigenschaften, z.B. durch ein Web Services Gateway, werden Web Services genau so komplex wie z.B. CORBA.

Durch die unabhängigen und anerkannten Web-Standards bietet SOAP gegenüber RMI oder CORBA und auch sonstigen RPC Protokollen den Vorteil, nicht an eine Programmiersprache gebunden zu sein.

SOAP selbst beschreibt die Art und Weise, wie Inhalte (Serialisierung) und die XML-Daten übertragen werden. Die Übertragung selbst hat mit SOAP aber eigentlich nichts zu tun, denn dafür sind andere Protokolle definiert.

Da Microsoft federführend bei der Spezifikation ist und das Dotnet-Framework seine Kommunikation vollständig auf SOAP aufbaut, wird es in der Zukunft eine Reihe von Java-Clients geben, die auf SOAP-Dienste von anderen Anbietern aufbauen, z.B. Microsoft.

Den Vorteilen stehen allerdings auch einige Nachteile gegenüber.

Die XML-Repräsentation der Dokumente macht die Datenmengen groß; ein Parsen der Parameter und Ergebnisse auf beiden Seiten ist erforderlich. In Umgebungen, die performante Übertragung fordern – etwa bei der Kommunikation mobiles Endgerät und Server –, wird sich SOAP deshalb nicht so schnell durchsetzen. Für z/OS existiert ein spezielles Hardware/Software Produkt, die XML Data Appliance, welche auf der Server Seite den XML Parsing Aufwand verbessert. Siehe Abschnitt 14.3.14 .

Des Weiteren ist auch die Sicherheit von SOAP-Verbindungen problematisch. Ein Client könnte sich mit einem Server verbinden und einen Aufruf starten, obwohl die Berechtigung fehlt. Sicherheitseigenschaften müssten erst auf der Server-Seite implementiert werden.

Die in Klartext übertragenen Nachrichten bilden ein weiteres Problem, was jedoch durch SSL gelöst werden kann.

Dies ist der Ablauf einer SOAP Anfrage: Ein Client-Programm besorgt sich, wie bei entfernten Programmen üblich, eine Referenz auf das entfernte Objekt. Das ist eine URL auf einem Server. Der Server empfängt eine normale http- POST-Anfrage. Diese enthält eine XML-kodierte Nachricht (Content-Typ ist einfach text/hmtl), in der die aufzurufende Methode und ihre Parameter kodiert sind. Der Server nimmt diese Nachricht entgegen, parsed das empfangene XML-Dokument und leitet die Anfrage an die Methode weiter. Diese produziert die Ausgabe, die wiederum als XML-Dokument über die Antwort vom Server zum Client geschickt wird. Der Client nimmt das Ergebnis entgegen, und die Kommunikation ist beendet.

SOAP bietet für entfernte Methodenaufrufe einige Standard-Datentypen an. Zu diesen gehören einfache Datentypen wie Ganzzahlen, Fließkommazahlen, Zeit- und Datumsangaben und Binärdaten.

Weiterhin unterstützt SOAP zusammengesetzte Datentypen wie Strukturen und Arrays. Wie diese Daten nun tatsächlich in eine XML-Nachricht umgesetzt werden, braucht man glücklicherweise nicht zu wissen. Als Endanwender kommen wir mit der Nachricht nicht in Kontakt.

20.2.9 Web Service Definition Language

Die Web Services Description Language (WSDL) ist eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Web Services zum Austausch von Nachrichten auf Basis von XML. WSDL ist ein industrieller Standard des World Wide Web Consortium W3C.

WSDL beschreibt einen Web Service als eine Menge von Schnittstellen, die mehrere mögliche Interaktionen anhand von Operationen mit einer Anwendung spezifizieren. Auf die Anwendung kann über definierte Adressen unter Verwendung festgelegter Kommunikationsprotokolle zugegriffen werden. Es werden im Wesentlichen die Operationen definiert, die von außen zugänglich sind, sowie die Parameter und Rückgabewerte dieser Operationen. Im Einzelnen beinhaltet ein WSDL-Dokument funktionelle Angaben zu:

- **der Schnittstelle,**
- **dem Zugangsprotokoll, und**
- **alle notwendigen Informationen zum Zugriff auf den Service, in maschinenlesbarem Format.**

Für die WSDL Beschreibung eines Web Services werden sechs XML Sprachelemente definiert:

types
Definition der Datentypen, die zum Austausch der Nachrichten benutzt werden
message (Nachricht)
Abstrakte Definitionen der übertragenen Daten, bestehend aus mehreren logischen Teilen, von denen jeder Teil mit einer Definition innerhalb eines Datentypsystems verknüpft ist.
portType (andere Bezeichnung Interface)
Eine Menge von abstrakten Arbeitsschritten (Operations), die von einem (oder mehreren) Ports unterstützt werden. Es existieren vier Typen (Operations) von ausgetauschten Nachrichten:

- **One-way: Der Service bekommt eine Input-Message vom Client.**
- **Request-response: Der Service bekommt einen Request (Input-Message) vom Client und sendet eine Antwort (Output-Message).**
- **Solicit-response: Der Service sendet eine Message und erwartet eine Antwort vom Client.**
- **Notification: Der Service sendet eine Output-Message.**

binding (Bindung)
Bestimmt das konkrete Protokoll und Datenformat für die Arbeitsschritte und Nachrichten, die durch einen bestimmten Port-Typ gegeben sind. Das Binding wird normalerweise mittels SOAP verwirklicht. andere Möglichkeiten sind IIOP, Dotnet, Java Message Service (JMS), oder WebSphere MQ
port (andere Bezeichnung endpoint)
Spezifiziert eine Adresse für eine Bindung, also eine Kommunikationsschnittstelle, üblicherweise ein URI. Eine Menge von Ports definiert normalerweise einen Service. In WSDL 2.0 wurde die Bezeichnung zu Endpoint geändert.
service (Service)
Fasst eine Menge von verwandten Ports zusammen.

20.2.10 Cobol Copy Book

Ein populärer Ansatz ist es, ein vorhandenes CICS Cobol Programm zu erweitern, indem man die Fähigkeit hinzufügt, als Web Service aufgerufen werden zu können..

Cobol Programmierer verwenden häufig eine als Copybook bezeichnete Struktur. Ein Copybook ist eine Datei mit Source Code, welcher zur Compile Zeit in ein Cobol Programm mittels COPY Statements eingefügt wird.

Vergleichbare Funktion in anderen Sprachen sind:

#include ...	**(C and C++),**
<!--#include ... -->	**(HTML),**
<%@ include ... %>	**(JSP)**

```
01   PKLR1-DETAIL-LOAN-RECORD.
    10  PKLR1-BASIC-SECTION.
        20  PKLR1-SORT-CONTROL-FIELD.
            30  PKLR1-USER-IDENT         PIC X(1).
            30  PKLR1-EXTRACT-CODE       PIC X(1).
                88  PKLR1-DATE-RECORD           VALUE '*'.
                88  PKLR1-DATA-RECORD           VALUE '0'.
                88  PKLR1-END-OF-FILE           VALUE '9'.
            30  PKLR1-SECTION            PIC X(1).
            30  PKLR1-TYPE               PIC X(1).
            30  PKLR1-NUMERIC-STATE-CODE PIC X(2).
            30  PKLR1-CONTRACT-NUMBER    PIC X(10).
        20  PKLR1-PAR-PEN-REG-CODE       PIC X(1).
        20  PKLR1-VALUATION-CODE.
            30  PKLR1-MORTALITY-TABLE    PIC X(2).
            30  PKLR1-LIVES-CODE         PIC X(1).
            30  PKLR1-FUNCTION           PIC X(1).
            30  PKLR1-VAL-INTEREST       PIC S9(2)V9(3) COMP-3.
            30  PKLR1-MODIFICATION       PIC X(1).
            30  PKLR1-INSURANCE-CLASS    PIC X(1).
            30  PKLR1-SERIES             PIC X(5).
        20  PKLR1-POLICY-STATUS          PIC X(2).
        20  PKLR1-PAR-CODES.
            30  PKLR1-PAR-TYPE           PIC X(1).
            30  PKLR1-DIVIDEND-OPTION    PIC X(1).
            30  PKLR1-OTHER-OPTION       PIC X(1).
        20  PKLR1-ALPHA-STATE-CODE       PIC X(2).
```

Abb. 20.2.9
Beispiel eines Cobol Copybooks

Häufig wird ein Copybook benutzt, um Input und Output Files in einem Cobol Programm zu beschreiben. Siehe das Beispiel in Abb. 20.2.9 .

Angenommen, wir benutzen Web Services für den Aufruf eines existierenden CICS Cobol Anwendungsprogramms. Es existiert Software, welche aus einem derartigen Copybook automatisch XML WSDL Information erzeugt. Hierzu benutzt man den Cobol Parser für die Erzeugung der XML Metainformation.

20.3 Web Services und CICS

20.3.1 Accessing CICS from the Web

Wir schauen uns drei unterschiedliche Ansätze an, wie man vom WWW (z.B. mit einem Browser) auf CICS zugreifen kann.

Die ersten beiden Ansätze benutzen den WebSphere Application Server als einen Zwischenschritt. Wir nehmen an, dass hierbei WebSphere und CICS auf dem gleichen z/OS System (oder Sysplex) laufen. Zur Erinnerung: es ist eine populäre Alternative, WebSphere auf einem getrennten Unix, Linux, oder Windows Server (oder zBX Blade) zu betreiben.

Der dritte Ansatz verzichtet auf WebSphere, und benutzt statt dessen die "Web Services" Komponente des CICS Transaction Server.

20.3.2 Alternative 1 – WebSphere Application Server ohne XML

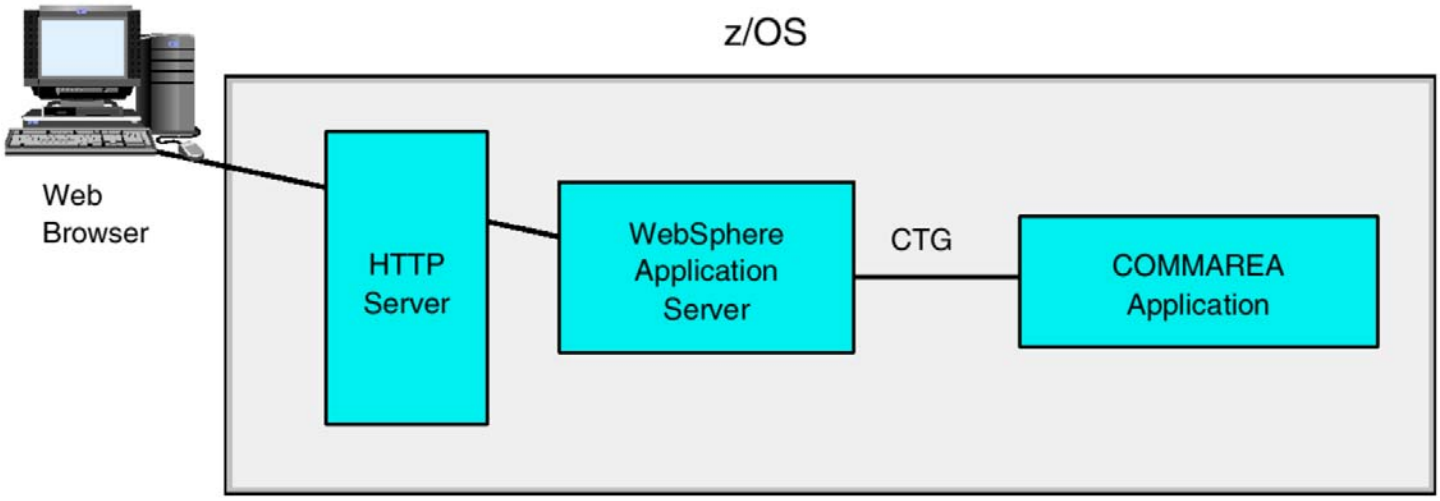


Abb. 20.3.1
Java Servlet und das CICS Transaction Gateway

Im ersten Ansatz senden wir eine HTTP Nachricht an das CICS Transaction Gateway (CTG). Dieses läuft als eine EJB in dem EJB Container eines Websphere Application Servers. Dieser wiederum läuft unter z/OS Unix System Services (USS).

In diesem ersten Ansatz wird kein XML eingesetzt.

Der Nachrichtenfluss erfolgt so:

1. **Mittels eines Web Browsers (Form Tag einer HTML Seite) gibt der Endbenutzer die Transaktion Input Daten ein.**
2. **Die Transaction Input Daten werden von dem Web Browser an den z/OS HTTP Server (Web Server) übertragen, und von dort an ein Servlet in dem WebSphere Application Server weitergereicht.**
3. **Das Servlet verbindet sich mit einem CICS Resource Adapter (in diesem Fall dem CICS Transaction Gateway), um über eine lokale Gateway Connection die CICS Transaktion auszuführen. Das Servlet agiert als ein TCP/IP Client für den CICS Resource Adapter.**
4. **Das CICS Transaction Gateway baut eine EXCI Request unter Benutzung der Java Native Interface (JNI) auf, und sendet die Nachricht an den CICS Transaction Server.**
5. **Über EXCI wird die Nachricht an die CICS COMMAREA mittels CICS sowie MRO (Band 1, Abschnitt 9.3.2) oder XCF gesendet.**
6. **Über EXCI erzeugt CICS eine Antwort an das CICS Transaction Gateway. Das CICS Transaction Gateway gibt die Antwort an das Servlet weiter.**
7. **Mit Hilfe des Servlets (oder einer Java Server Page) wird die Antwort an den Browser weitergereicht.**

Angenommen, die Ausführung der Transaktion erfordert Zugriffe auf zwei unterschiedliche Transaction Monitore auf zwei geographisch verteilten Systemen. Dies erfordert das 2-Phase Commit Protokoll.

Das CICS Transaction Gateway implementiert die JEE Connector Interface. Dazu kann es mit der z/OS Resource Recovery Services (RRS) Komponente zusammenarbeiten. Resource Recovery besteht aus den Programmen und Schnittstellen, die es WebSphere für z/OS und CICS ermöglichen, Änderungen in mehrfachen geschützten Ressourcen (z.B. Datenbanken) mit Hilfe von Two-Phase Commit durchzuführen.

20.3.3 Alternative 2 – WebSphere verarbeitet XML Nachrichten

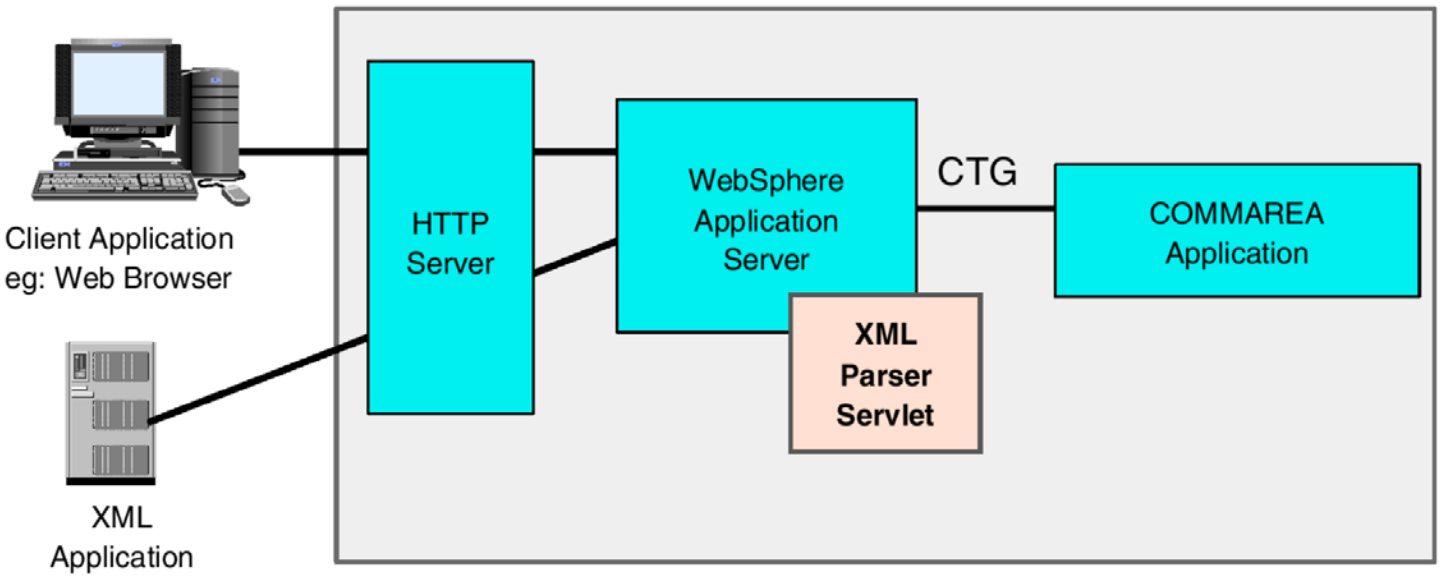


Abb. 20.3.2
Verarbeitung einer XML Nachricht durch ein Servlet

Ein zweiter Ansatz benutzt eine XML Request Nachricht. Ein XML Parser Servlet übersetzt die Nachricht in eine CTG Nachricht.

Die Client Application übergibt die XML Nachricht an den WebSphere Application Server. Ein WebSphere Application Server XML Converter (z.B. ein Servlet) parses und übersetzt die XML Nachricht mittels einer XSLT Transformation. Die resultierende Nachricht kann von einer Server Komponente (z.B. Servlet, JSP, Java Bean, und/oder Enterprise Java Bean) weiter verarbeitet werden.

Diese Komponente (z.B. eine EJB) kann herausfinden, wie die Nachricht aus dem XML Dokument weiter verarbeitet werden soll. Die Nachricht kann dann in eine IMS Message, eine CICS COMMAREA, oder eine SQLJ oder JDBC Request übersetzt, und an das entsprechende Enterprise Information System weitergereicht werden. Im Falle von CICS kann sie für eine Verbindung mit dem CICS Transaction Gateway benutzt werden.

Das RDz (Eclipse based) Integrated Development Environment kann für das Mapping der Datenstrukturen eingesetzt werden, und kann die Logik für die Transformation erzeugen. Konnektoren wie der IMS Connector für Java oder das CICS Transaction Gateway können benutzt werden, um die Nachricht an den Transaction Manager weiterzugeben.

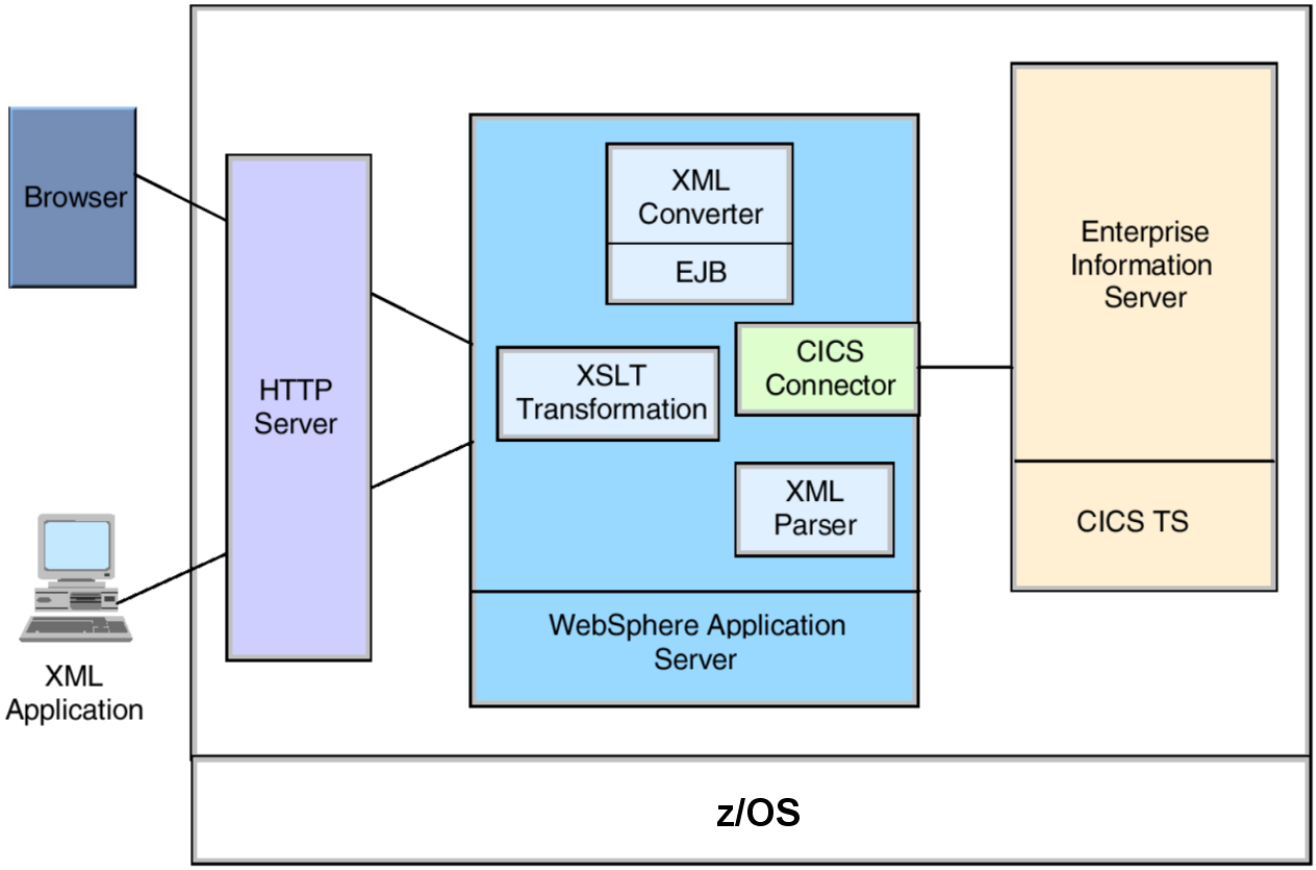


Abb. 20.3.3
Parsing der Nachricht als Input für das CICS Transaction Gateway

Dargestellt sind die einzelnen Komponente die benutzt werden um die XML Nachricht zu übersetzen und die Verbindung zu dem CICS Connector, z.B. dem CICS Transaction Gateway herzustellen.

Dies sind die einzelnen Schritte:

1. **Eine HTTP Request wird von dem XML Client an den HTTP Server gesendet.**
2. **Die HTTP request ruft ein Servlet auf. Das Servlet läuft innerhalb der JVM der WebSphere Application Server Servlet Engine. Das Servlet führt zunächst eine evtl. erforderliche Transformation durch, und benutzt dann die Parser APIs um das Source XML Dokument zu übersetzen. Mit Hilfe der Java Gateway Class wird die lokale Gateway Connection aufgebaut.**
3. **Das Servlet benutzt die EXCI Request Class um die Request an den CICS Transaction Server weiterzureichen. Die COMMAREA Request wird an CICS mittels MRO oder XCF übertragen.**
4. **The EXCI Response wird an das CICS Transaction Gateway übergeben.**
5. **Das Servlet benutzt die Parser APIs um aus der COMMAREA ein XML Dokument zu erzeugen.**
6. **Das Servlet sendet die XML Antwort an den Klienten.**

Parsing und Transformation Servlets müssen derzeitig noch für jede neue Anwendung manuell erstellt werden. Der Vorteil des WebSphere Application Servers als an Integrations-Plattform ist die saubere Trennung zwischen Präsentation, Business Logic und Daten.

Diese beiden bisher diskutierten Beispiele benutzen den WebSphere Application Server und Enterprise Java Beans (EJB) als Zwischenschritt für den Zugriff auf CICS. Eine dritte Alternative (CICS Web Services) benötigt keine WebSphere, sondern benutzt die XML Unterstützung, die Teil des CICS Transaction Servers ist. Diese Unterstützung besteht vor allem aus Message Handlers und aus Pipelines.

20.3.4 Alternative 3 – CICS Web Services

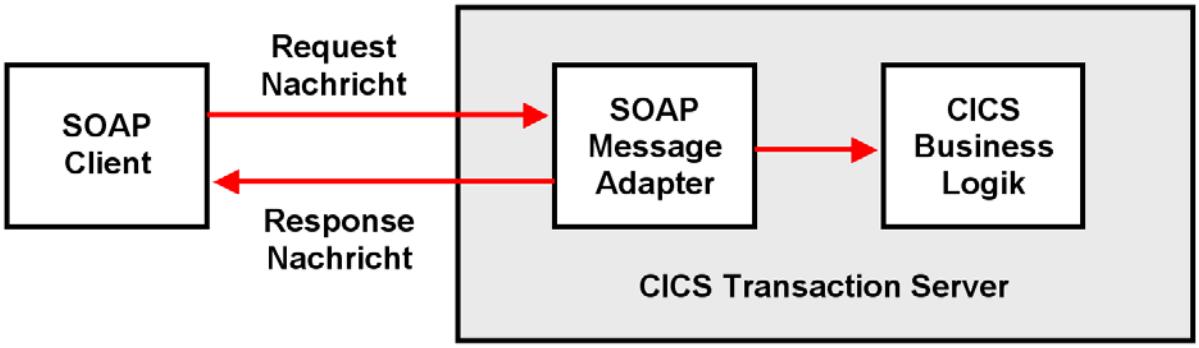


Abb. 20.3.4
Web Services und CICS

Zur z/OS Version des CICS Transaction Servers gehört ein Simple Object Access Protocol (SOAP) Message Adapter. Hierfür wird WebSphere nicht benötigt. Mit Hilfe von Web Services kann auf neue oder existierende CICS Anwendungen zugegriffen werden, die in einer beliebigen Programmiersprache geschrieben sind. Dies geschieht dadurch, dass eine XML-basierte SOAP Message in eine COMMAREA abgebildet wird.

Die SOAP Request kann die CICS Anwendung entweder über HTTP oder über WebSphere MQ aufrufen.

In der in Abb. 20.3.4 gezeigten Konfiguration arbeitet CICS als Web Services Provider. Alternativ können CICS Anwendungen auf Web Services zugreifen, die auf anderen Rechnern verfügbar sind. In diesem Fall arbeitet CICS als Web Services Requester. Diese Einrichtung ist z.B. für Business-to-Business (B2B) Anwendungen interessant.

20.3.5 Pipeline and Message Handler

Der CICS SOAP Message Adapter besteht aus Pipelines. Ein CICS Pipeline Programm besteht unter anderem aus Message Handler Programmen. Ein Message Handler ist ein Programm, welches eigenständig Web Service Requests und Responses verarbeiten kann. Message Handler bearbeiten u.A. Aufgaben wie:

- **Verarbeiten von SOAP Nachrichten,**
- **Header Verarbeitungsprogramme,**
- **Security Handler für Teile des Web Service Security Rahmenwerkes**
- **Andere, darunter auch selbst geschriebene Message Handler.**

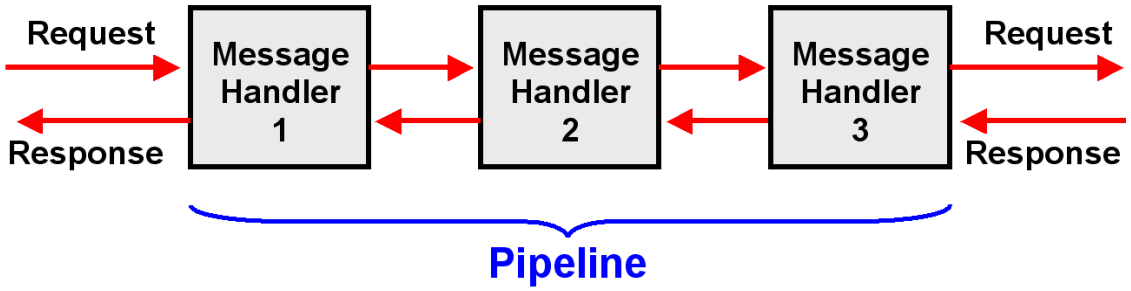


Abb. 20.3.5
Der SOAP Message Adapter besteht aus einer Serie von Bausteinen

Eine Pipeline ist eine Gruppe von Message Handlers die der Reihe nach ausgeführt werden. Dabei wird der Output von einem Message Handler als Input für den nächste Message Handler benutzt.

Eine CICS Pipeline Configuration File ist eine XML File, welche die Message Handler Programmme sowie die SOAP Header Processing Programme beschreibt. All diese Information wird in einem HFS (Unix System Services Hierarchical File System) Directory gespeichert.

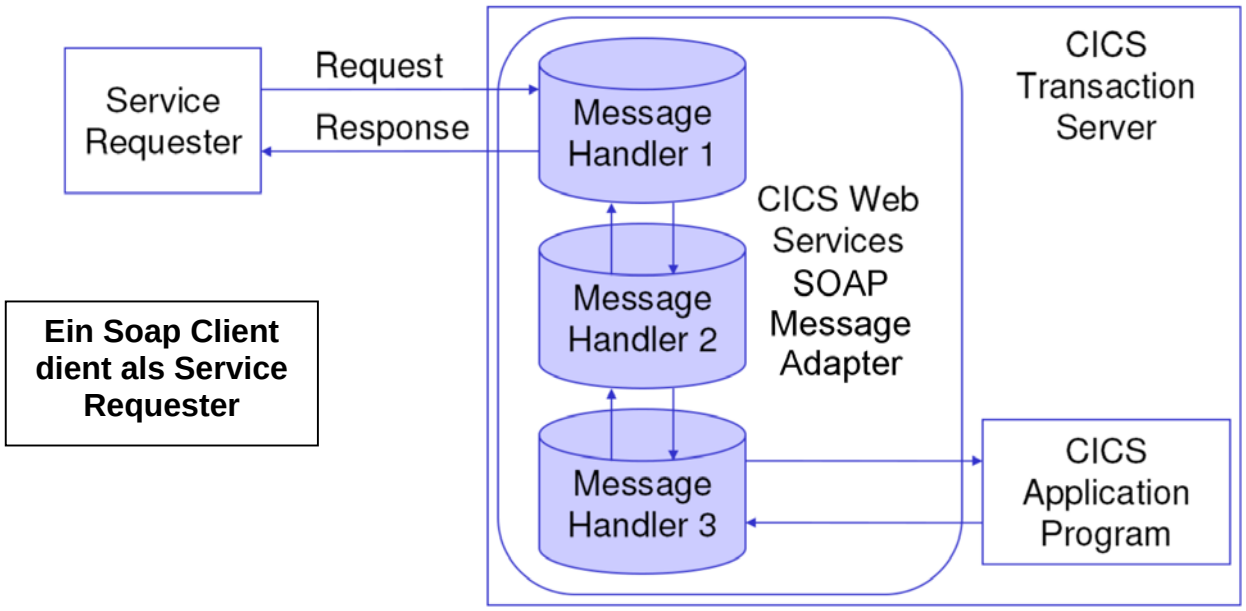


Abb. 20.3.6
CICS Web Service Provider

20.3.6 CICS als Web Services Provider oder Requester

Message Handler und Pipelines können für beliebige unterschiedliche Aufgaben eingesetzt werden, u.a. auch für SOAP.

Integriert in das Internet, und unter Benutzung von Web Services, kann CICS eine dieser beiden Rollen übernehmen:

- **CICS als ein Web Service Provider**
- **CICS als ein Web Service Requester**

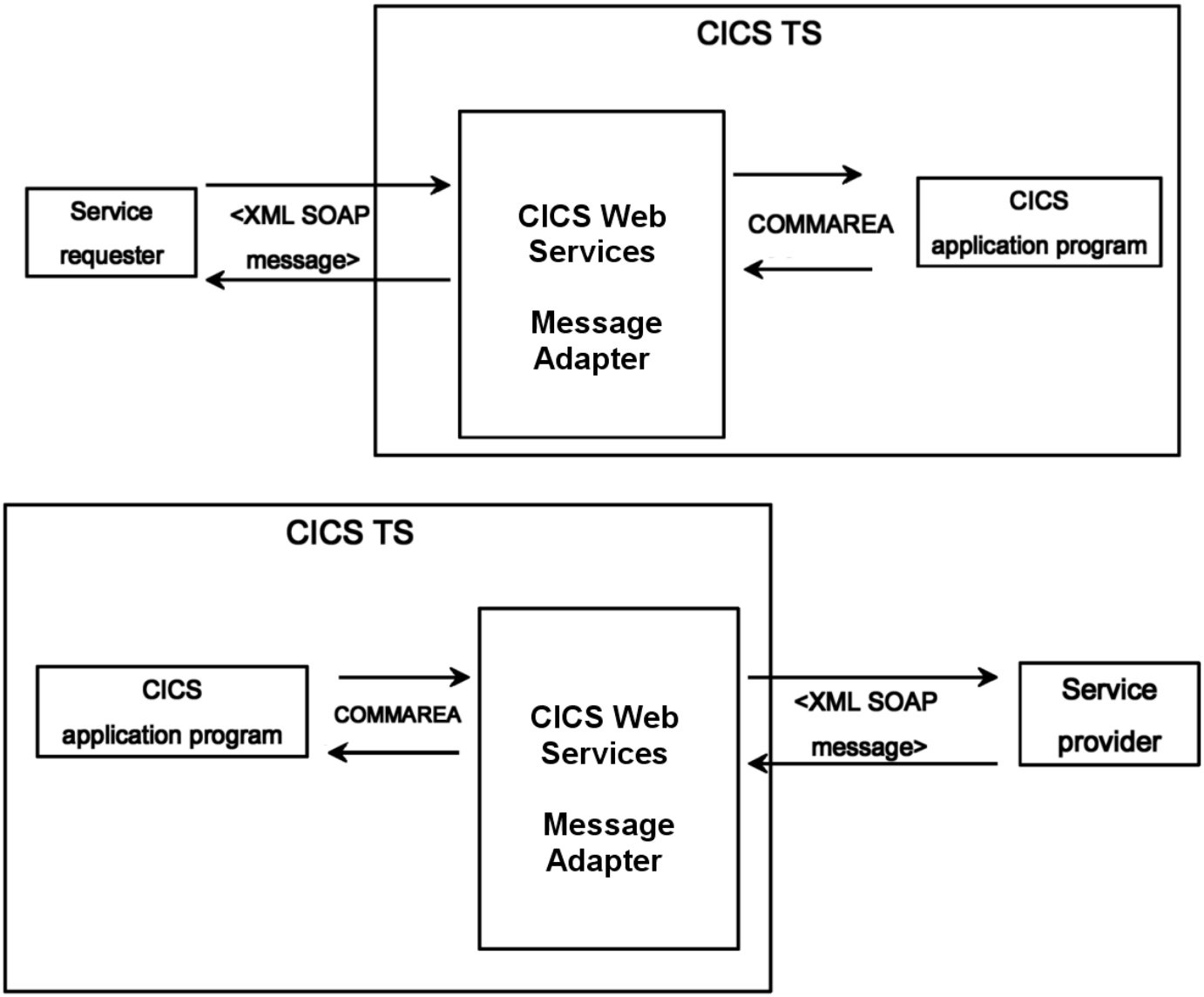


Abb. 20.3.7
Zwei unterschiedliche Rollen als Provider oder Requestor

Der CICS SOAP Message Adapter besteht aus Pipelines die Service Provider unterstützen, sowie aus Pipelines welche Service Requester unterstützen. Eine Pipeline kann als Service Requester Pipeline oder als Service Provider Pipeline konfiguriert werden, aber nicht als beides.

Eine CICS Service Provider Pipeline ist eine Pipeline, welche (1) inbound SOAP Messages empfängt, (2) deren Inhalt verarbeitet, und (3) eine Antwort erzeugt. Eine CICS Service Requester Pipeline ist eine Pipeline, welche (1) outbound SOAP Messages sendet, (2) eine Antwort empfängt, und (3) den Inhalt der Antwort verarbeitet.

CICS verfügt über spezielle SOAP Message Handler Programme (SOAP Message Adapter). Hiermit kann eine Pipeline als eine SOAP Node konfiguriert werden:

- **Eine Service Provider Pipeline ist der SOAP Empfänger einer Request und der SOAP Sender für die Antwort.**
- **Eine Service Requester Pipeline ist der ursprüngliche SOAP Sender einer Request und der SOAP Empfänger für die Antwort.**

http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/IBM/SG247206.pdf

20.3.7 Verarbeitung der Service Request

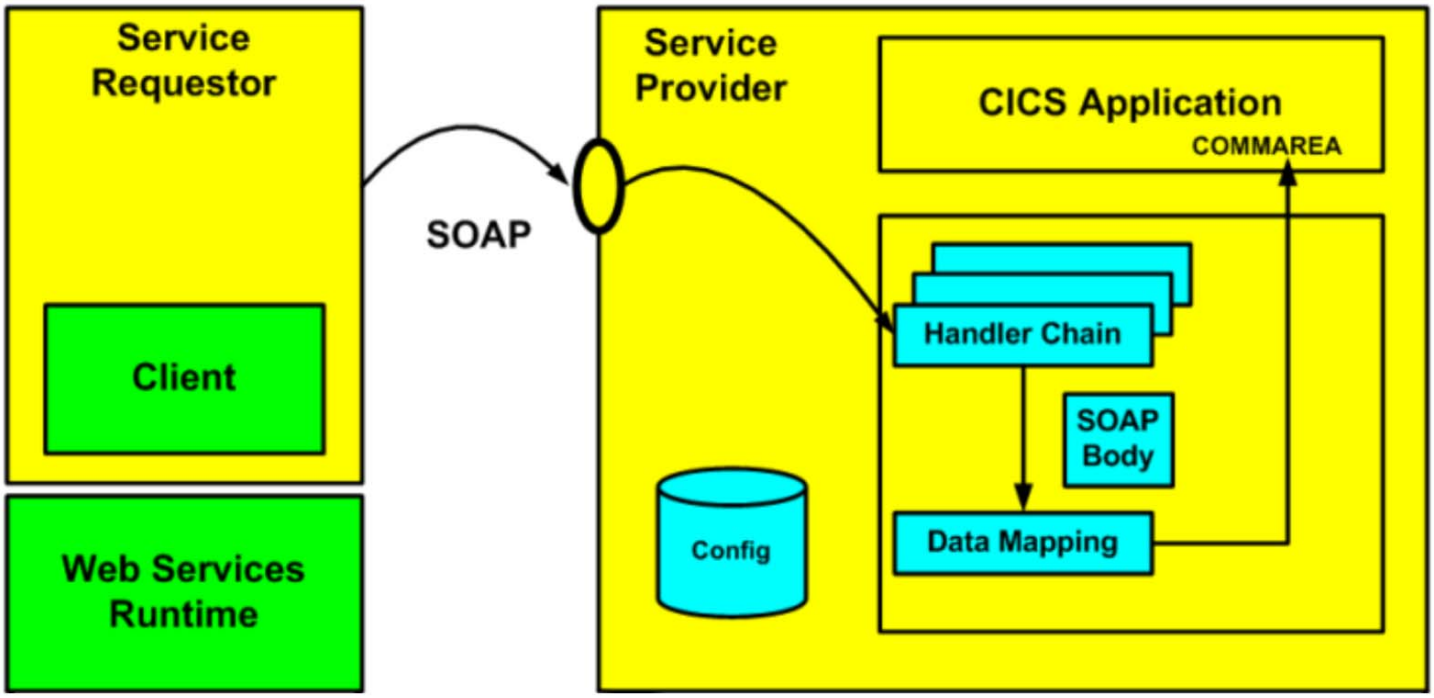


Abb. 20.3.8
Verarbeitung einer XML Nachricht

Abb. 20.3.8 ist eine Zusammenfassung von Abb. 20.3.9. Handler Chain ist eine andere Bezeichnung für Pipeline.

Eine SOAP Nachricht trifft auf einem Port ein, der von CICS überwacht wird. Eine Pipeline (Message Handler Chain) extrahiert den SOAP Body von der SOAP Message und übersetzt ihn in eine COMMAREA.

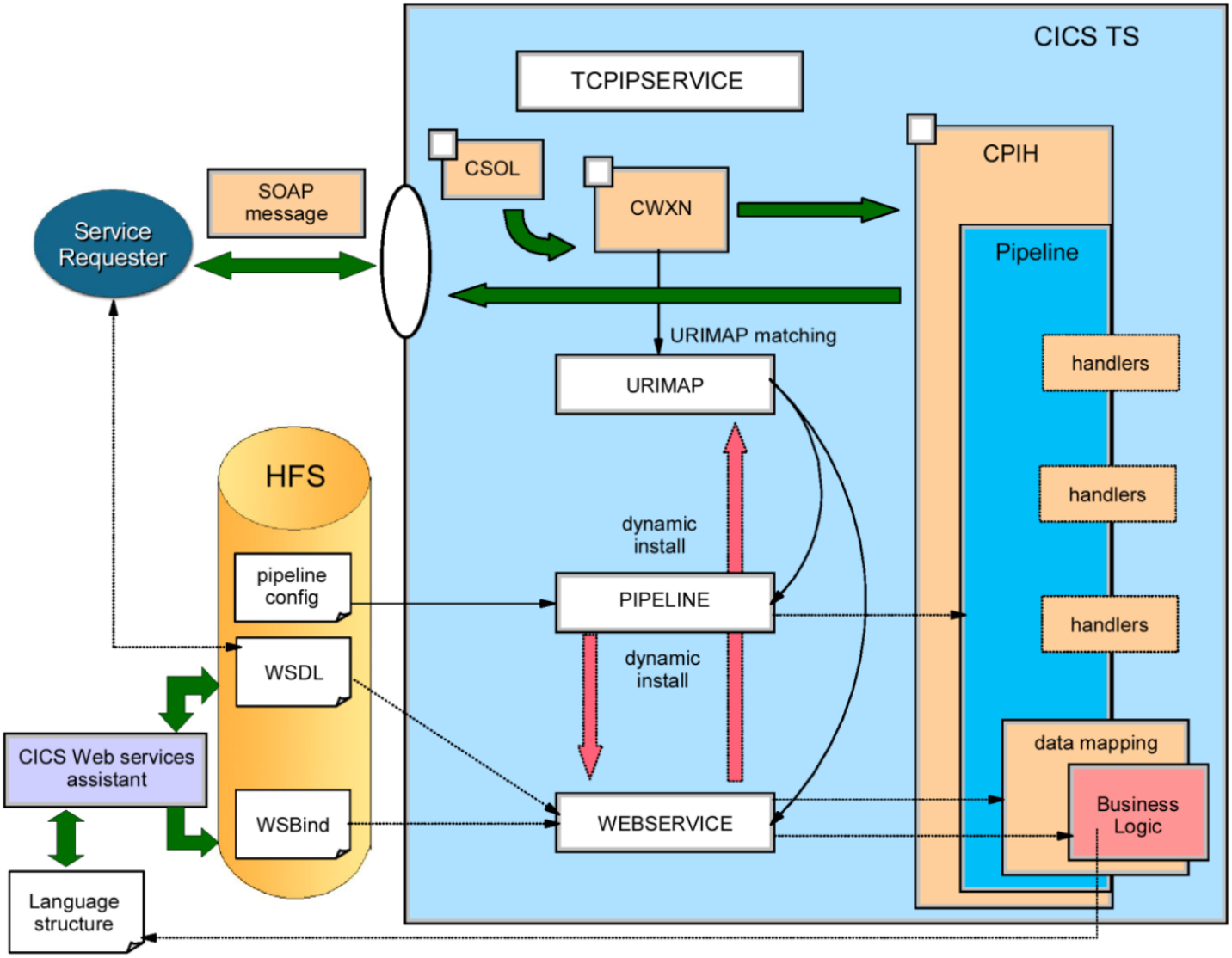


Abb. 20.3.9
Verarbeitung einer XML Nachricht - Details

Der CICS Transaction Server beinhaltet 3 Transaktionen für die Verarbeitung von Web Services Requests: CSOL, CWXN, and CPIH.

Die Sockets Listener Transaktion CSOL überwacht einen Port auf eintreffende HTTP Requests. Dieser Port ist in der TCPIPSERVICE Resource Definition spezifiziert. TCPIPSERVICE Resource Definitions- werden benutzt, um die Zuordnung zwischen Port Nummern und CICS Services festzulegen (einschließlich CICS Web Services). CICS benutzt die TCPIPSERVICE Definition eines Ports um herauszufinden, welcher CICS Service aufgerufen werden soll, wenn eine Nachricht auf diesem Port eintrifft.

Wenn die SOAP Nachricht eintrifft, ordnet CSOL die Transaktion zu, die in dem TRANSACTION Attribut der TCPIPSERVICE Definition angegeben ist. Normalerweise wird dies die Web Attache Transaktion CWXN sein.

CWXN findet den URI (Unified Resource Identifier) in der HTTP Request. Es durchsucht die URIMAP Resource Definitionen File. Diese enthält als Urimams bezeichnete Einträge für solche Urig, die der Server zu empfangen erwartet. Bei der passenden URIMAP hat das USAGE Attribut den Wert PIPELINE, und das PATH Attribut stimmt mit der URI in der HTTP Request überein. Angenommen, CWXN findet eine solche URIMAP. Es benutzt die PIPELINE

und WEBSERVICE Attribute der URIMAP Definition, um den Namen der PIPELINE und WEBSERVICE Definitionen zu finden. Es benutzt diese, um die eintreffende Request zu verarbeiten. CWXN benutzt weiterhin das TRANSACTION Attribut der URIMAP Definition, um den Namen der Transaktion zu finden, welche die Pipeline verarbeiten soll. Unter normalen Umständen wird dies die CPIH Transaktion sein.

CPIH beginnt damit die Verarbeitung der Pipeline. Es benutzt die PIPELINE Definition um den Namen der Pipeline Cohnfiguration File zu finden. CPIH benutzt die Pipeline Configuration File um herauszufinden, welche Message Handler Programme und SOAP Header Processing Programme aufgerufen werde sollen.

Ein Message Handler in der Pipeline (normalerweise ein CICS-interner SOAP Message Handler) entfernt den SOAP Envelope der eintreffenden Request und übergibt den SOAP Body an die Data Mapper Funktion.

CICS benutzt den DFHWS-WEBSERVICE Container, um den Namen der gewünschten WEBSERVICE Definition an den Data Mapper zu übergeben. Dieser benutzt die WEBSERVICE Definition, um die eintreffende XML Request auf COMMAREA abzubilden. Der Data Mapper ruft das zugehörige CICS Anwendungsprogramm auf.

Dieses ist sich der Tatsache nicht bewusst, dass es als ein Web Service ausgeführt wird. Es führt eine normale Verarbeitung aus, und übergibt seine Output COMMAREA an den Data Mapper.

Dieser Output wird vom Data Mapper in ein SOAP Body Format konvertiert.

Alle Configuration Files sind in dem Hierarchical File System von z/OS Unix System Services abgespeichert.

Anmerkung: Die heutigen Implementierungen des CICS Transaction Servers beinhalten zusätzlich zu COMMAREA eine als "Container" bezeichnete Alternative. COMMAREA ist aus historischen Gründen in der Größe auf 32 KByte begrenzt. Der „Container" bietet zusätzliche Flexibilität auf Kosten zusätzlicher Komplexität an.

20.3.8 Facit

Schwierig, nicht war ?

Doch dies ist die Art, wie wir Enterprise Class Anwendungen erstellen. Die Programmierung von CICS Web Services wird wesentlich erleichtert, wenn Sie diese unter Benutzung des Eclipse RDz Plug-In durchführen.

Dies ist einer der Hintergründe. Es existieren geschätzt 20 – 30 verschiedene Verfahren, wie man auf ein CICS Programm zugreifen kann. Das klassische Verfahren ist natürlich BMS. Die meisten anderen Verfahren benutzen das Internet.

Weil CICS so weit verbreitet ist, haben nicht nur IBM, sondern auch zahlreiche weitere Software Hersteller auf Wünsche Ihrer Kunden reagiert und Verfahren für die unterschiedlichsten Anwendungs-Szenarios entwickelt. Fast allen ist gemeinsam, dass sie das Internet benutzen, und COMMAREA als Schnittstelle zur Präsentationslogik einsetzen.

Um möglichst viele Gemeinsamkeiten zu erreichen, hat man zahlreiche weitere Schnittstellen definiert, und mehrere Ebenen der Indirektion eingezogen. Transaktionen wie CSOL, CWXN, and CPIH oder die Benutzung der TCPIPSERVICE Definition sind hierfür ein Beispiel.

20.4 SOA Konzepte

20.4.1 Prozess

In der Informatik benutzen wir gerne die Vokabel „Prozess“. Die Frage ist : Was ist ein Prozess ?

Hier sind einige Beispiele für die unterschiedlichen Benutzungen des Begriffes „Prozess“:

Prozess aus Sicht der Informatik oder des Betriebssystems
Halbleiter Herstellungsprozess
Prozess vor einem Amtsgericht
Prozess aus organisatorischer Sicht eines Unternehmens (Business Process, Geschäftsprozess)

Elemente eines Geschäftsprozesses können sein:

Prozesskosten
Prozesskostenrechnung
Prozesskostenmanagement
Prozessanalyse
Prozessorganisation
Prozessorientierung
Prozessmodellierung
Prozessverantwortlicher
Business Process Reengineering
Geschäftsprozessoptimierung

Der Vorstand eines Unternehmens versteht unter Prozess die Ausführung eines Geschäftablaufes, allgemein als Geschäftsprozess bezeichnet. Dieser Begriff hat mit dem Prozessbegriff in der Informatik (Ausführung eines Programms und Steuerung durch den Scheduler/Dispatcher des Betriebssystemkernels) nur wenig zu tun.

20.4.2 Geschäftsprozess

Ein Geschäftsprozess (Business Process) ist ein Rezept für das Erreichen eines wirtschaftlichen Ergebnisses. Jeder Geschäftsprozess hat Inputs, Methoden und Ergebnisse. Die Inputs sind eine Voraussetzung, die vorhanden sein müssen, bevor das Verfahren in die Praxis umgesetzt werden kann. Wenn die Methoden die Inputs übernehmen werden bestimmte Ergebnisse erstellt.

Ein Geschäftsprozess ist eine Sammlung von verwandten strukturellen Aktivitäten, die einen bestimmten Ergebnis für einen bestimmten Kunden produzieren.

Ein Geschäftsprozess kann Teil eines größeren, umfassenderen Prozesses sein und kann andere Geschäftsprozesse, in seine Methoden einbeziehen.

Der Geschäftsprozess kann als ein Kochbuch für die Führung eines Unternehmens betrachtet werden. Dies sind Beispiele für Geschäftsprozesse:

- **"Nehmen Sie den Telefon Anruf entgegen",**
- **"Eine Bestellung aufgeben",**
- **"Produzieren einer Rechnung"**

Ein Computer kann (oder auch nicht) für das Ausführen eines Geschäftsprozesses benutzt werden. Vor 150 Jahren wurden alle Geschäftsprozesse ohne Verwendung eines Computers ausgeführt. Heute werden fast alle Geschäftsprozesse werden mit Hilfe einer IT-Infrastruktur ausgeführt.

20.4.3 Beispiele von Anwendungsarchitekturen

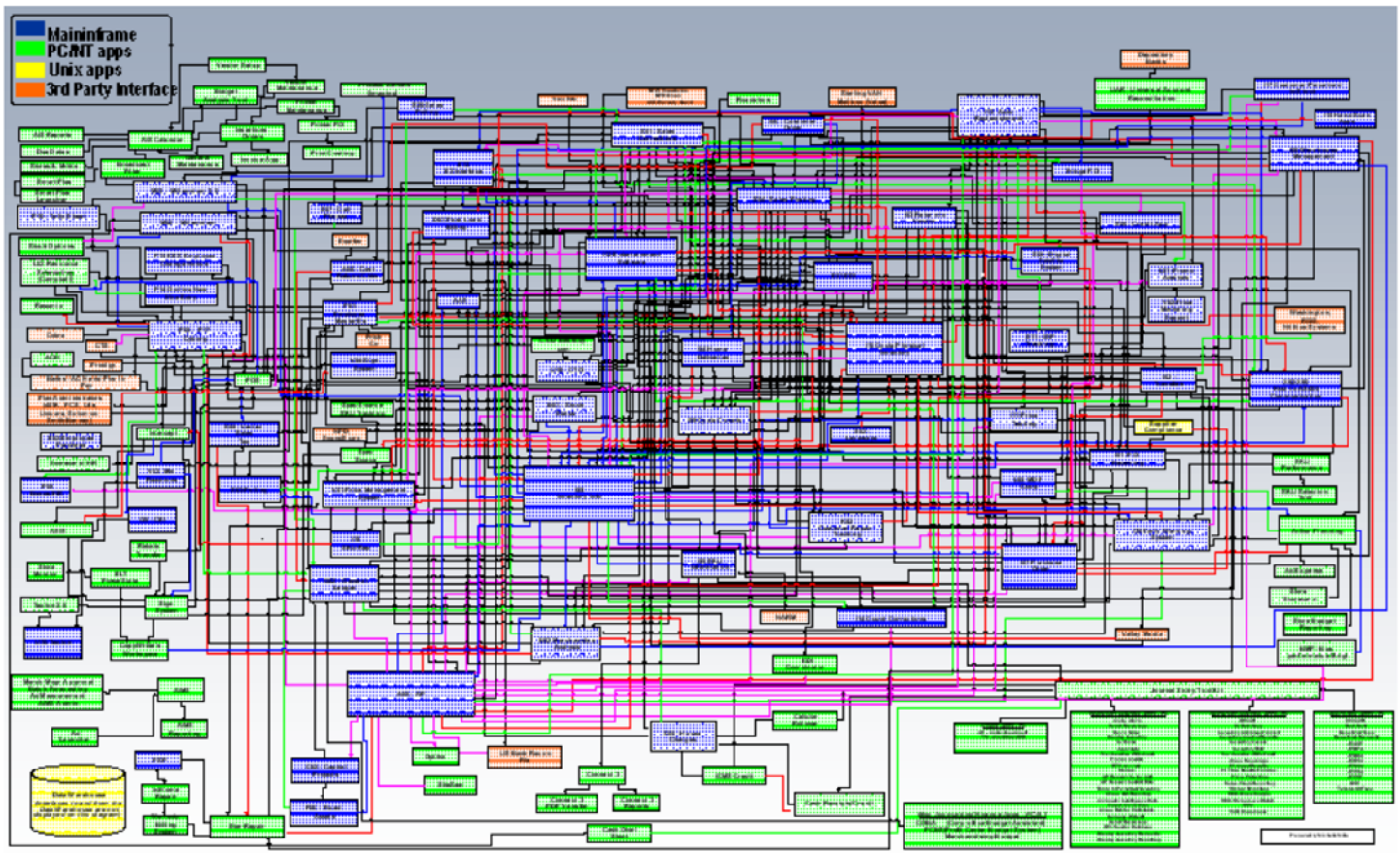


Abb. 20.4.1
Anwendungsarchitektur einer Consumer Electronics Firma

Abb. 20.4.1 zeigt an Hand eines konkreten Beispiels einer USA Consumer Electronics Company die Verknüpfungen der einzelnen IT Anwendungen untereinander. Jede der IT Anwendungen implementiert einen Geschäftsprozess.

Die gute Nachricht bei einer derartigen IT-Infrastruktur ist: Sie funktioniert in der Regel und arbeitet zuverlässig.

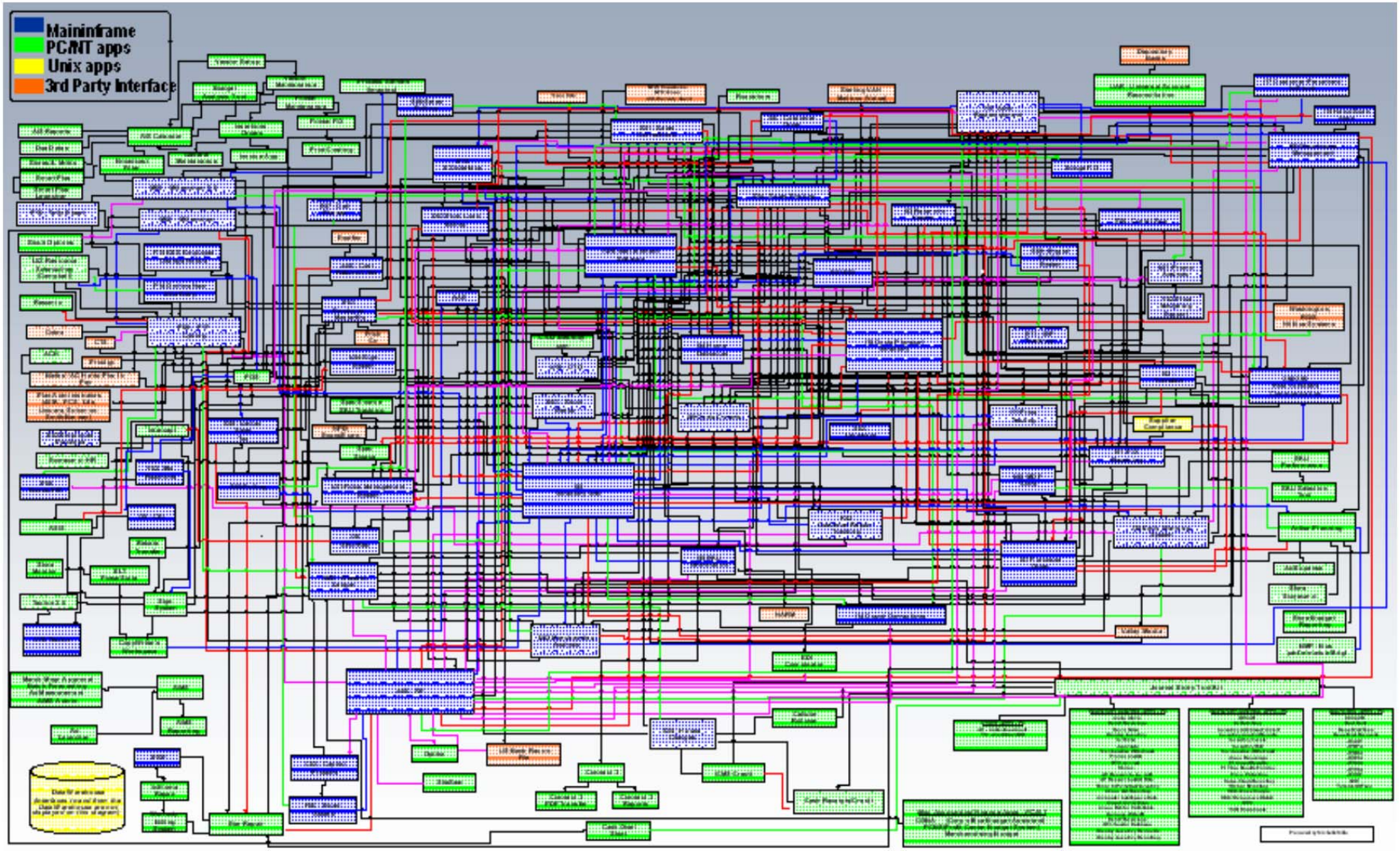


Abb. 20.4.1
Anwendungsarchitektur einer Consumer Electronics Firma

Die schlechte Nachricht bei einer derartigen IT-Infrastruktur ist: Änderungen sind in der Regel nur mit einem großem Aufwand durchführbar, weil das Zusammenspiel der einzelnen Komponenten (IT Anwendungen) nicht beeinträchtigt werden darf.

Eine Service Orientierte Architektur bemüht sich um eine Umstrukturierung, welche Änderungen leichter möglich macht.

Ein weiteres Beispiel: Ein Airbus besteht aus 8 Millionen Bauteilen (part numbers), die häufig mehrfach in unterschiedlichen Bauteilgruppen eingesetzt werden. Das Netzwerk der Zusammenhänge muss bei Änderungen und Verbesserungen der Bauteile ständig konsistent gehalten werden. Beispiel: wenn Sie die Abmessungen eines Bauteils verändern, passt es evtl. räumlich nicht mehr in allen Bauteilgruppen, in denen es eingesetzt wird. Anything, that can go wrong, will.

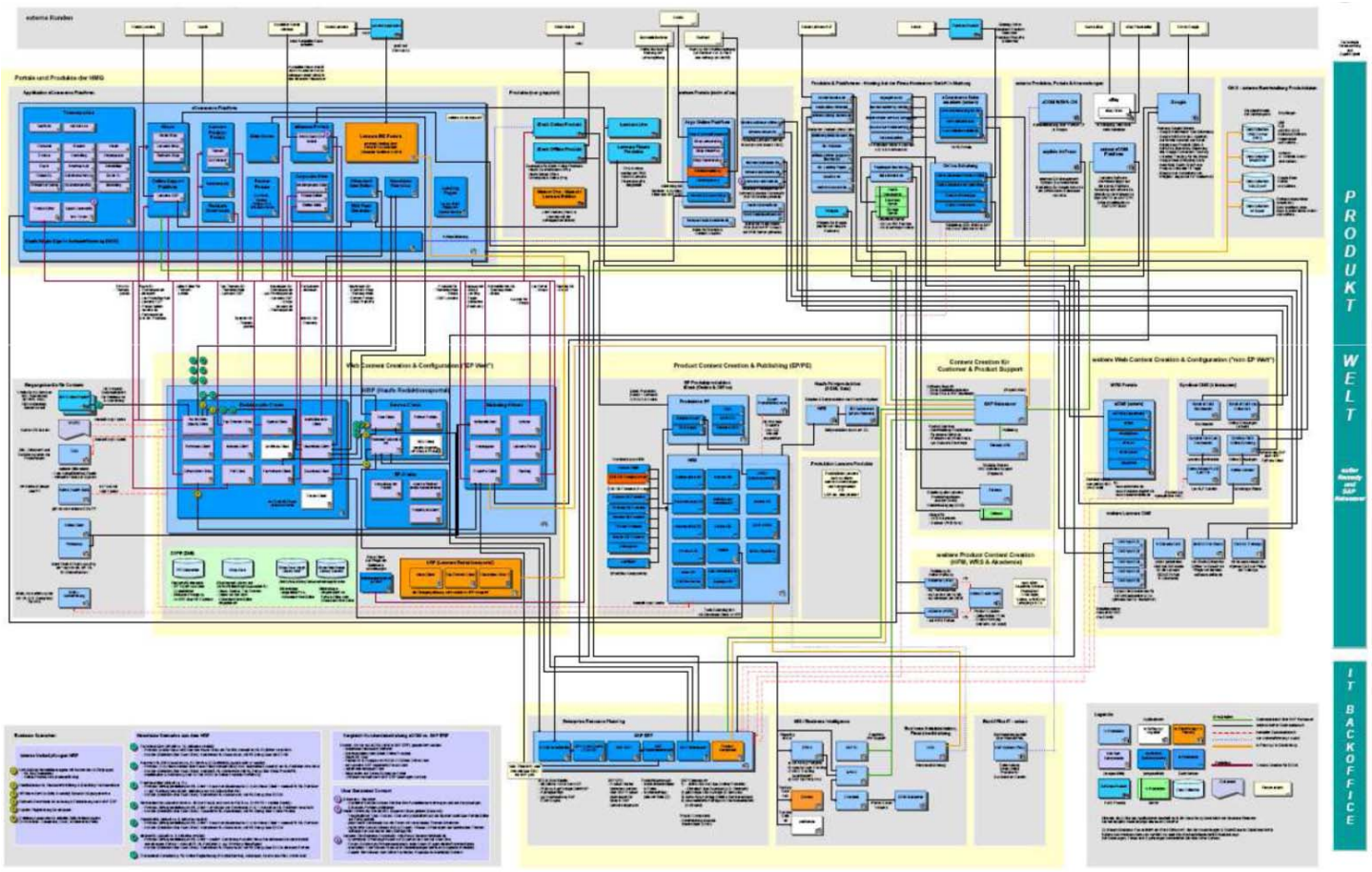


Abb. 20.4.2
Ein weiteres Beispiel

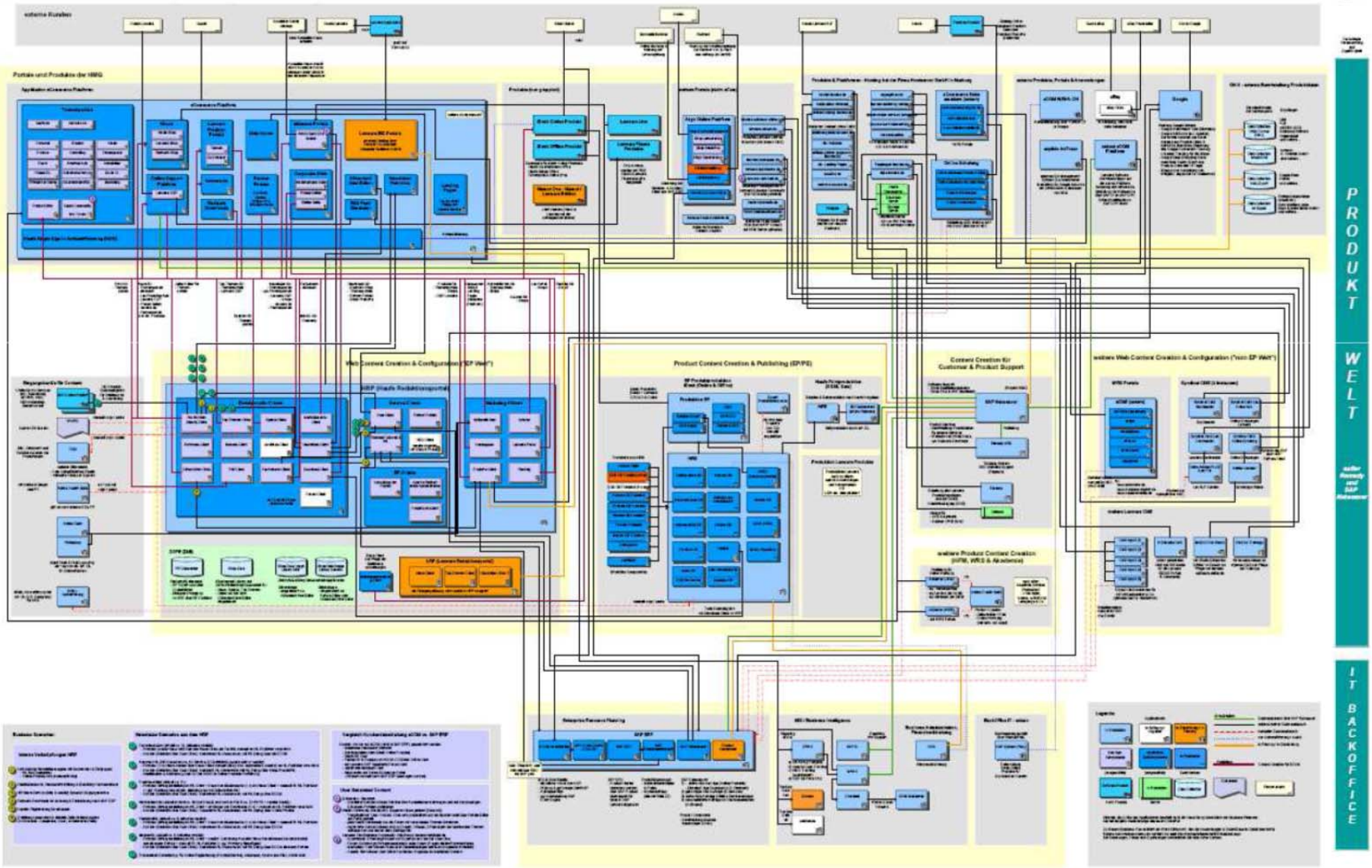


Abb. 20.4.2
Ein weiteres Beispiel

20.4.4 Workflow

Ein Geschäftsprozess (Business Process) ist die Definition eines Ablaufes und den damit verbundenen Tätigkeiten, welcher zur Durchführung eines Geschäftes (Auftrags) notwendig ist. Beispiele von Geschäftsprozessen sind:

- **Beantragung eines Kredites in einer Bank**
- **Abwickelung einer Bestellung bei einem Versandhaus**
- **Überweisung der Renten zum Monatsende an die Pensionäre eines Automobilkonzerns**

Ein Geschäftsprozess kann sehr komplex sein. Ein Beispiel ist die Überweisung der Betriebsrente an 100 000 ehemalige Daimler-Mitarbeiter. Es existieren viele Empfängerbanken, von denen manche im Ausland sind und Überweisungen in unterschiedlichen Währungen erfordern. Es existieren individuell unterschiedliche Steuerabzüge, Renten Erhöhungen und Kürzungen, Lohnpfändungen, ...

Die einzelnen Verarbeitungsschritte eines Geschäftsprozesses werden als fachliche Aktivitäten (Business Process Actions) bezeichnet. Beispiele für fachliche Aktivitäten in einem Versandhaus sind z.B.:

- **Plausibilitätsprüfung einer Bestellung**
- **Überprüfung ob die Bestellung aus dem vorhandenen Lagerbestand befriedigt werden kann**
- **Ausstellen der Versandpapiere**
- **Rechnungsstellung**

Vor der Einführung von Computern wurden fachliche Aktivitäten von Hand ausgeführt. Mit der Einführung des Computers wurde es möglich, fachliche Aktivitäten zu automatisieren. Die Realisierung der fachlichen Aktivitäten in einer IT-Umgebung werden als technische Aktivitäten (technical actions) bezeichnet. Hierbei kann eine fachliche Aktivität auch durch eine Transaktion, bestehend aus mehreren technischen Aktivitäten realisiert werden. Ein Beispiel ist eine Transaktion, bestehend aus zwei Java Methoden, welche eine Lastschrift und Gutschrift durchführen.

Eine technische Aktivität wird durch ein „Fachmodul“ (eine IT Anwendung) realisiert.

Die Ausführung eines Geschäftsprozesses erfolgt durch das serielle oder parallele Abarbeiten einer Reihe von fachlichen Aktivitäten, wobei diese in der Regel als technische Aktivitäten implementiert sind.

Wird die Ausführung einzelner technische Aktivitäten eines Geschäftsprozesses durch eine IT Umgebung unterstützt, spricht man von einem Workflow. Ein Workflow steuert die serielle und/oder parallele Ausführung einer Reihe von technischen Aktivitäten. Transaktionale Workflows sind Workflows, welche ACID (Atomicity, Consistency, Isolation, Durability) Eigenschaften auf Workflow Ebene garantieren. Ein derartiger Workflow wird auch als „long running transaction“ bezeichnet. Eine long running transaction besteht aus vielen Einzelschritten (short running transactions), welche die Umsetzung von einer Vielzahl von fachlichen Aktivitäten in entsprechende technische Aktivitäten darstellen.

Die von einem bestimmten Geschäftsprozess aufgerufenen IT Anwendungen werden häufig auch von anderen Geschäftsprozessen benutzt. Es existiert ein komplexes Netzwerk von Verknüpfungen zwischen den einzelnen IT Anwendungen.

20.4.5 Anforderungen an die IT-Implementierung eines Geschäftsprozess

Performance

Bis zu 10 000 Transaktionen pro Sekunde

Mandantenfähigkeit

Anwendungstechnische Unterstützung für eigenständige, juristisch unabhängige Unternehmen. Erfordert z.B. getrennte physische Datenhaltung; eventuell Verschlüsselung von Transaktionsdaten; Ablaufverfolgung der Aufträge und Abrechnung der erbrachten Leistungen pro Mandant

Realtimefähigkeit

Direkte Bearbeitung der Transaktion , und Weiterleitung z.B. an Disposition und Buchung

Unterbrechungsfreier Betrieb

Zero Downtime

Komponentenbasiertes System

Hinzufügen, Verändern und Ersetzen von Banken oder Versicherungsprodukten im laufenden Betrieb

Standardsoftware

Anbindung bzw. Integration von Vendor Software

Prozesssteuerung

Service Level Agreement (SLA) Steuerung
Transaktionssteuerung,
Verarbeitungssteuerung

20.4.6 Geschäftslogik, Steuerungslogik und IT Infrastruktur

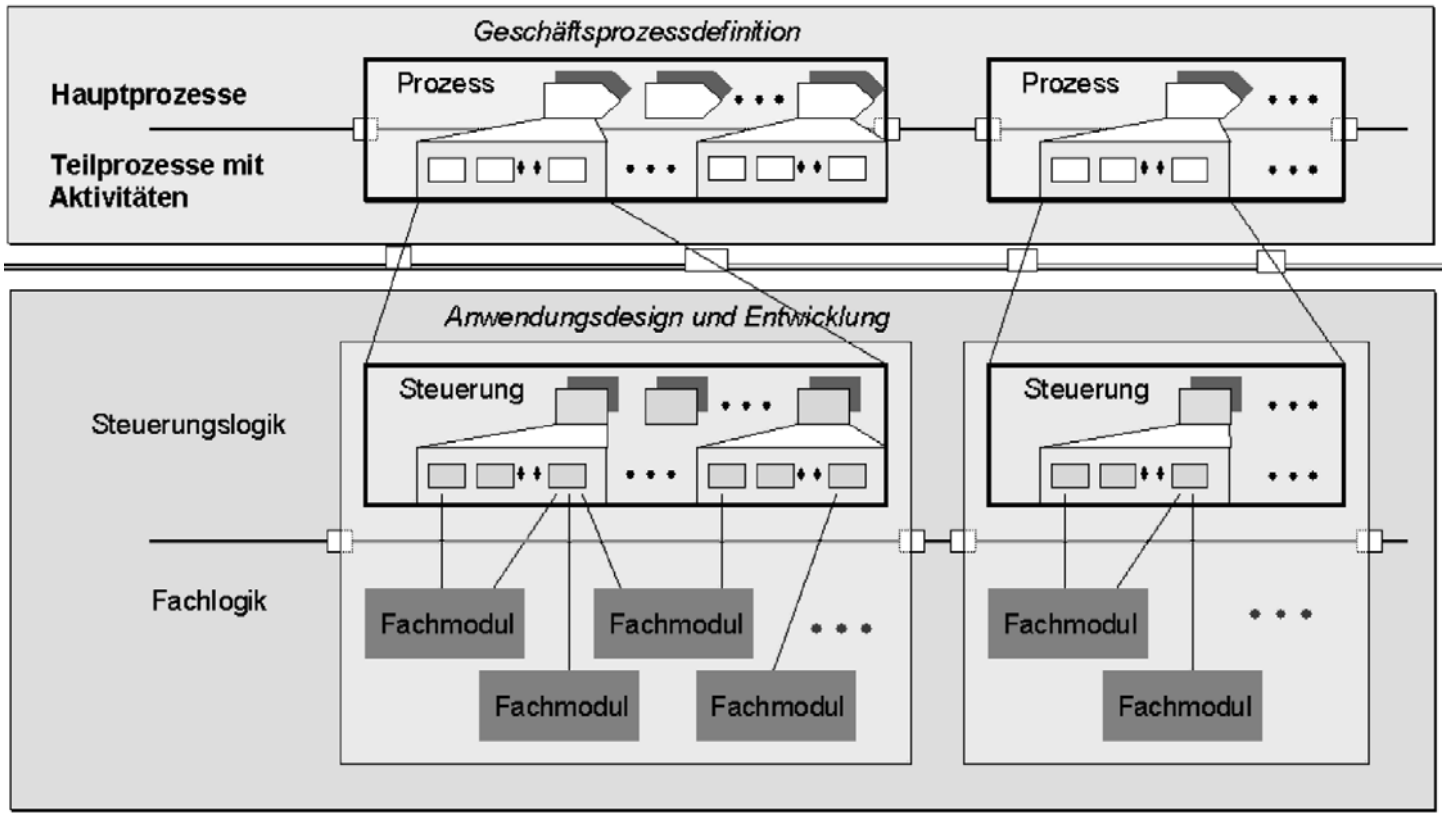


Abb. 20.4.3
Untergliederung der Geschäftsprozesse

Abb. 20.4.3 zeigt eine Trennung der Geschäftsprozesse von der IT Implementierung durch Steuerungslogik (z.B. einen oder mehrere Workflow Prozesse) und der Fachlogik, bestehend aus einzelnen Fachmoduls, welche die technischen Aktivitäten implementieren.

Der Geschäftsprozess stellt einen betriebswirtschaftlichen Hauptprozess dar, der durch mehre betriebswirtschaftliche Teilprozesse mit zugeordneten Aktivitäten besteht.

Die Steuerungslogik besteht typischerweise aus einem Workflow Prozess, der die einzelnen fachlichen Aktivitäten möglichst automatisch steuert.

Die eigentliche Arbeit wird durch die Fachmodule erledigt.

Joachim Franz, Wilhelm G. Spruth: "Reengineering von Kernanwendungssystemen auf Großrechnern". Informatik Spektrum, Band 26, April 2003
http://www.cedix.de/Publication/Mirror/reeng13.pdf

20.4.7 Aufteilung in Master Workflow und Subworkflow

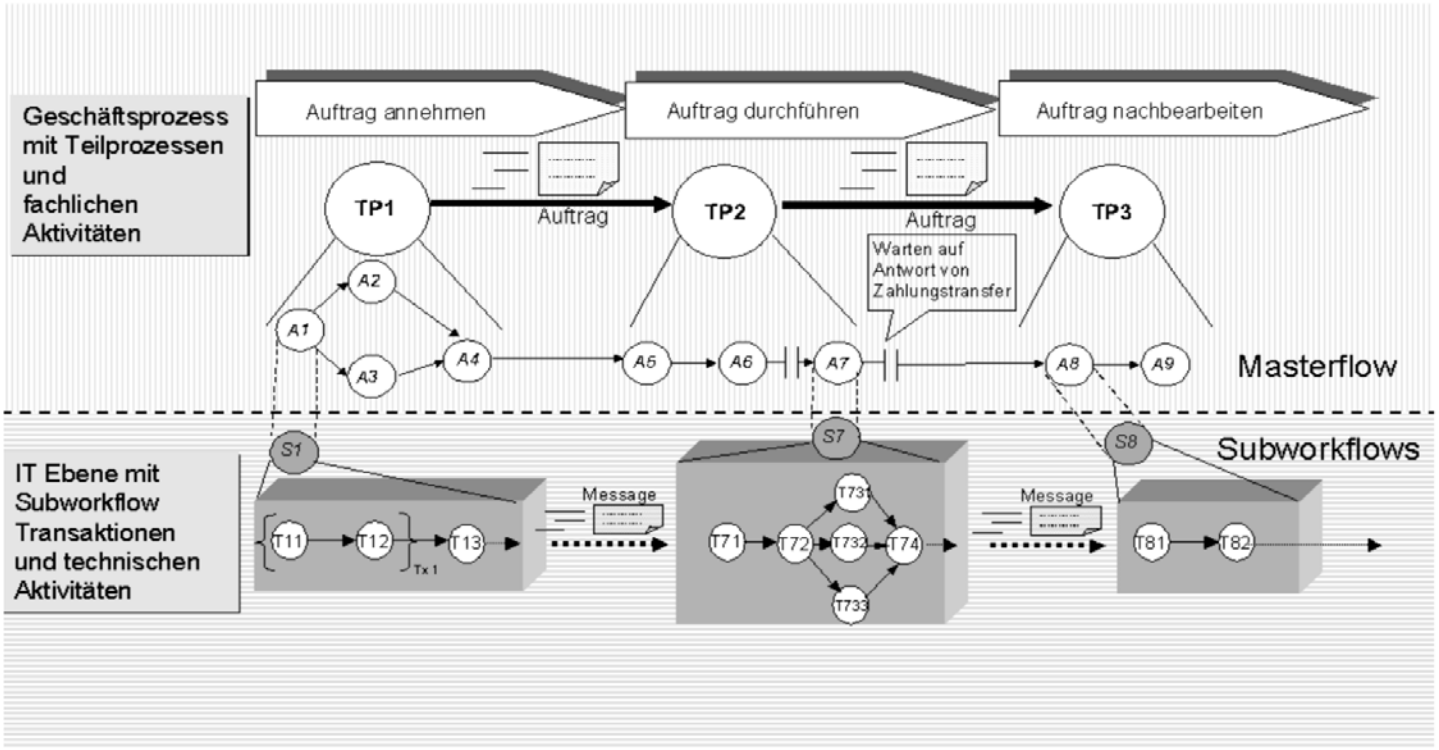


Abb. 20.4.4
Abbildung der fachlichen Aktivitäten auf technische Aktivitäten

Fachliche Aktivitäten sind:

- **A1: Auftrag fachlich prüfen**
- **A2: Disposition für Auftrag prüfen**
- **A3: Empfänger Banken bestimmen**
- **A4: Ausführungszeiten bestimmen**
- **A5: Zahlungspositionen, Konten, Bank, Land zusammenfassen**
- **A6: Leitwege bestimmen**
- **A7: Zahlungstransfer durchführen**
- **A8: Zahlung fiskalisch verbuchen**

Beispiele für technische Aktivitäten sind:

Subworkflow 1

- **T11: Auftragsprüfung auf Plausibilität**
- **T12: Auftragsprüfung fachlich**
- **T13: Status und Bestätigungsmeldung an Auftraggeber**

Subworkflow 7

- **T71: Leitwegliste auf Vollständigkeit prüfen**
- **T72: Zahlungstransfer vorbereiten**
- **T731: Führe Transfer für Land x1 (z.B. USA) und Bank y1 aus**
  - **T732: Führe Transfer für Land x2 und Bank y2 aus**
  - **T733: Führe Transfer für Land x3 und Bank y3 aus**
- **T74: Buchungsaufträge erstellen**

Subworkflow 8

- **T81: Ergebnisauswertung des Zahlungsverkehrs**
- **T82 Positionen fiskalisch verbuchen**

20.4.8 Service Oriented Architecture

Eine Service Orientierte Architektur (SOA) ist der vielfach angedachte Lösungsansatz für die beschriebenen Probleme.

SOA ist ein Konzept, keine Technik!

Das Ziel einer SOA ist eine an Geschäftsprozessen ausgerichtete IT Infrastruktur

Es existiert noch keine allgemein akzeptierte Definition. Zentrale Aspekte sind:

- **Interoperabilität**
- **lose Kopplung von Anwendungen**
- **Web Services, SOAP und WSDL werden als geeignete Werkzeuge angesehen.**

SOA ist kein neuer Standard, sondern verknüpft zahlreiche vorhandene Standards und Technologien zu einer Architektur. Abb. 20.4.5 zeigt beispielsweise die Elemente, die für eine Service Oriented Architecture eingesetzt werden.

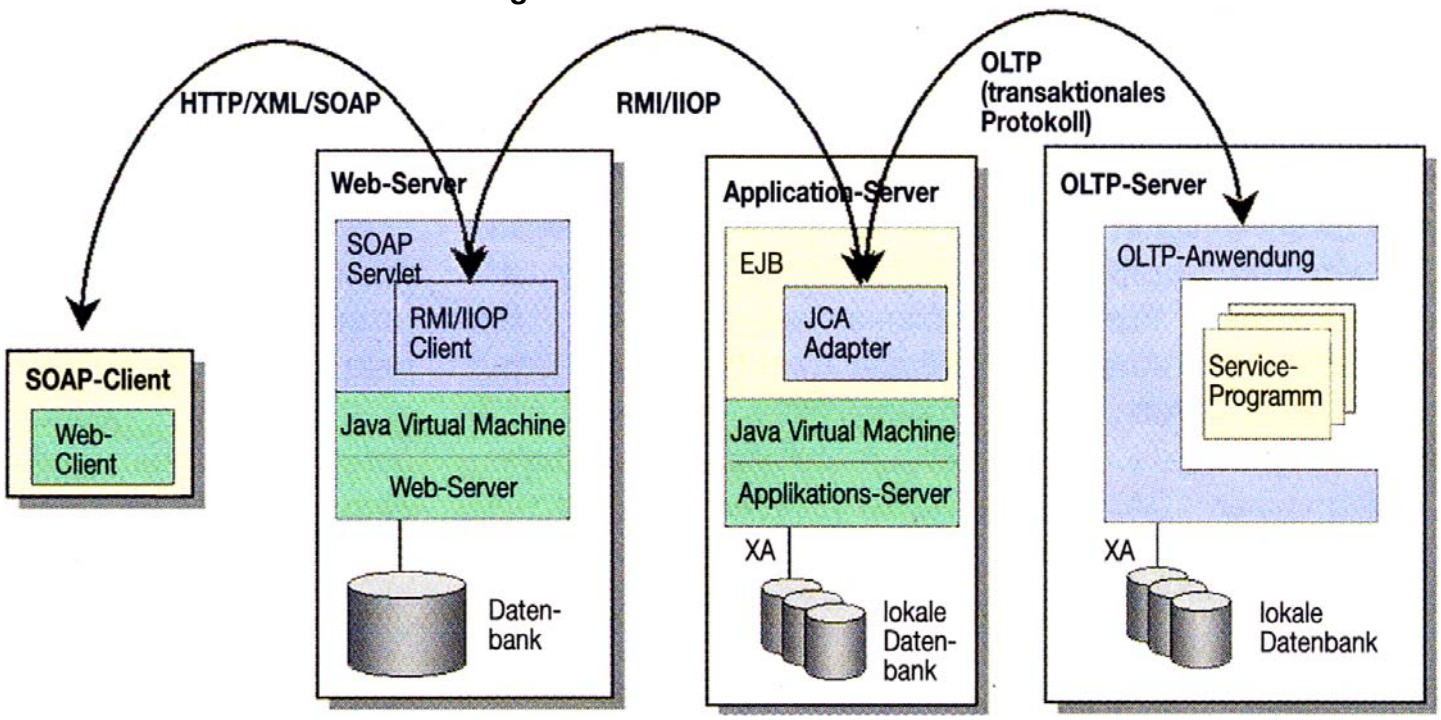


Abb. 20.4.5
Implementierungsbeispiel

In Abb. 20.4.5 ist die Integration von Host-Systemen für das Online Transaction Processing (OLTP) mit Hilfe der Java Enterprise Edition (JEE) dargestellt. Ein Online Transaction Processing (OLTP) Server ist z.B. CICS.

Die X/Open XA Interface ist in Abschnitt 19.1 dargestellt.

Die Service Oriented Architecture lässt sich am einfachsten anhand der von ihr spezifizierten Rollen charakterisieren.

Die Rollen werden von den Komponenten, die diese Architektur implementieren, eingenommen. Die zentrale Rolle spielt hierbei der Dienstanbieter (Service-Provider), der Dienste (Services) anhand von Schnittstellen zur Verfügung stellt, die er selbst implementiert. Die Schnittstellen werden von Dienstnutzern (Service Requester) in Anspruch genommen.

Web-Services sind derzeitig die bei weitem der bevorzugte Technologie, um eine SOA zu erstellen und einzuführen. Es können aber an Stelle von Web-Services auch andere Standardtechniken wie Corba, J2EE oder Dotnet benutzt werden; auch eine eigenentwickelte Technik ist möglich. Bei SOA dreht sich alles um das Aufteilen und Verwalten der Services sowie der Prozesse und Orchestrierungen, die auf diesen Services aufsetzen. Welche Technik eingesetzt wird, sollte davon abhängen was gebraucht wird.

Bei der Realisierung einer Service Oriented Architecture stellen Web Services den Service Provider dar. Sie stellen eine Menge von Diensten beliebigen Anwendungen (den Service Requestern) zur Verfügung. Die Dienste sind in der Regel mittels der Web Service Description Language (WSDL) beschrieben. Die Service Requester nehmen die Anwendungen unter Verwendung XML basierter Protokolle,z.B.dem Simple Object Access Protocol (SOAP), in Anspruch.

Eine SOA ist von Natur aus keineswegs hochskalierbar. Eine SOA ist ein Konzept, dessen Skalierbarkeit von der verwendeten Technik und Architektur abhängt. Werden Services zu fein granular entworfen, bereitet die Skalierbarkeit der Lösung vermutlich Probleme. Anwender müssen daher zuerst die SOA richtig entwerfen, die Eigenschaften ihrer Bestandteile verstehen und die passende Technik und Entwicklungsplattform finden.

20.4.9 Erfassung des Wissens um einen Geschäftsprozess

Für einen Geschäftsprozess, z.B. die monatliche Überweisung der Betriebsrente an 100 000 Daimler Mitarbeiter, ist typischerweise ein Team von Mitarbeitern der Fa. Daimler AG zuständig. Das Wissen um die Durchführung des Geschäftsprozesses sitzt in den Köpfen der Mitarbeiter (das haben wir schon immer so gemacht). Typischerweise existiert in einem Großunternehmen niemand, der alle Geschäftsprozesse auch nur näherungsweise versteht. In der Regel sind die Geschäftsprozesse in irgend einer Form informell dokumentiert, z.B. in der Form einer Loseblattsammlung in einer Reihe von Leitz Ordnern. Geschäftsprozesse ändern sich dauernd, z.B. weil ein Mitglied des Vorstandes eine brillante Idee hat, oder weil der Gesetzgeber neue Gesetzte erlässt. Die Loseblattsammlung enthält handschriftliche Änderungsnotizen, ist in der Regel veraltet, fehlerhaft und unvollständig.

Wenn der Abteilungsleiter die Anweisung gibt “Überweisen Sie am letzten Freitag des Monats die Betriebsrenten an alle ehemaligen Mitarbeiter“, dann weis der Sachbearbeiter, der das schon seit vielen Jahren gemacht hat, was er zu tun hat. Er weiß auch, dass der Prozess aus vielen einzelnen Schritten (fachliche Aktivitäten) besteht. Wenn er in Urlaub ist, weis sein Vertreter ebenfalls (hoffentlich), was er zu tun hat.

Fachliche Aktivitäten werden als technische Aktivitäten auf einem Computer durchgeführt, z.B. als Transaktionen. Die Durchführung eines Geschäftsprozesses erfordert die Ausführung vieler technischen Aktivitäten. Es ist denkbar, dies mit Hilfe eines Scriptes, eines Workflows, durchzuführen. Es ist aber erstaunlich festzustellen, wie viele manuelle Interventionen in der Regel auch heute noch erforderlich sind.

Die Automatisierung der Prozessabläufe (fachlichen Aktivitäten) erfordert eine formale Beschreibung des in den Köpfen der Sachbearbeiter vorhandenen Wissens. Letztere mögen sehr kompetent auf ihrem Fachgebiet sein, verfügen aber über keine Programmiersprachen Kompetenz. Das Prozesswissen wird deshalb mittels einer grafischen Darstellung erfasst, die der Sachbearbeiter verstehen kann. Eine derartige grafische Darstellung eines derartigen Prozesses ist in Abb. 20.4.6 wiedergegeben. Der Sachbearbeiter verifiziert die Darstellung mit der Aussage: „Jawohl, das ist mein Prozess“.

Dies ist ein sehr kritischer, weil fehleranfälliger Schritt, der ein sehr sorgfältiges Vorgehen erfordert. Leider ist es bis heute nicht möglich, formal zu verifizieren, dass die grafische Darstellung den Geschäftsprozess tatsächlich korrekt wiedergibt.

Das in Abb. 20.4.6 dargestellte Beispiel wurde mittels der eEPK-Methode (erweiterte Ereignisgesteuerte Prozessketten) des ARIS Systems erstellt. ARIS (Architektur integrierter Informationssysteme) ist ein verbreitetes Produkt der Firma IDS Scheer AG in Saarbrücken. Die IDS Scheer AG ist eine Ausgründung von Prof. Dr. August-Wilhelm Scheer. Institut für Wirtschaftsinformatik, Universität des Saarlandes.

Die Darstellung versucht die Komplexität wiederzugeben.

Die zweite Säule von links ist eine Expansion (Vergrößerung) eines Teils der ersten Säule von links, die dritte Säule eine Expansion eines Teils der der zweiten Säule, usw.

Wenn diese Beschreibung in digitaler Form innerhalb eines Computers vorliegt, ist es kein Problem, die Beschreibung in ein beliebiges anderes Format zu konvertieren.

Probleme:

- **Wie können Sie sicherstellen, dass diese digitale Beschreibung eines Geschäftsprozesses fehlerfrei ist ?**
- **Wie können Sie die häufigen Änderungen der Geschäftsprozesse eindeutig nachvollziehen ?**

Das in Abb. 20.4.6 dargestellte Beispiel ist ein Ausschnitt einer eEPK Beschreibung eines konkreten, aber sehr einfachen Geschäftsprozesses. Die Verifikation der korrekten Wiedergabe besteht in der visuellen Betrachtung des Sachbearbeiters: „Jawohl, das ist mein Prozess“. Selbst in diesem sehr einfachen Beispiel ist es nicht unwahrscheinlich, dass sich Fehler eingeschlichen haben. Die Verifikation einer eEPK Beschreibung von Hunderten sehr viel komplexerer Geschäftsprozesse ist ein bisher ungelöstes Problem.

Die Einführung einer SOA in den Unternehmen wird u.A. aus diesem Grund nur sehr langsam und schrittweise erfolgen.

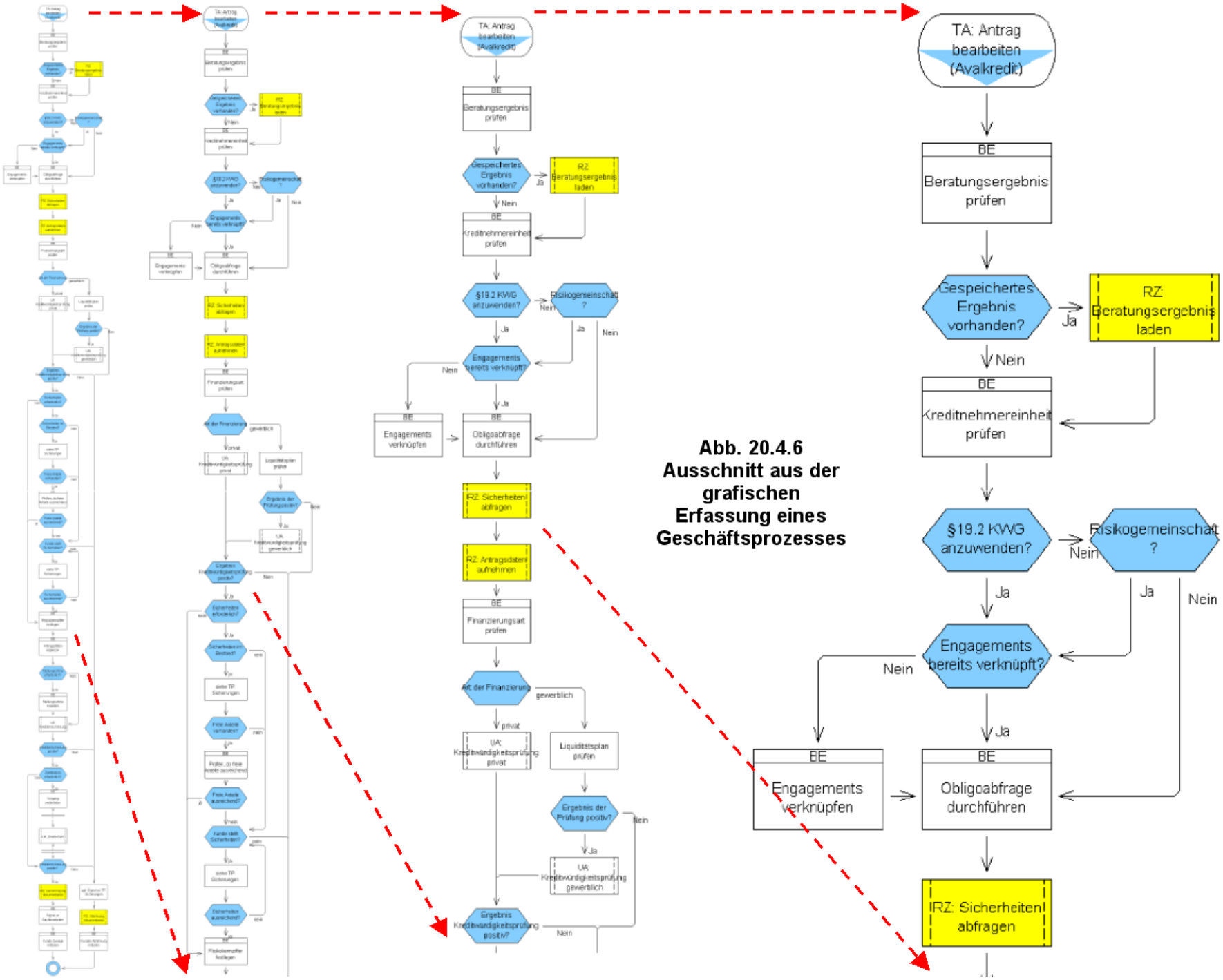


Abb. 20.4.6
Ausschnitt aus der grafischen Erfassung eines Geschäftsprozesses

20.4.11 Business Process Execution Language

Die grafische eEPK Darstellung hat den Vorteil, dass aus ihr automatisch eine Beschreibung in einer ausführbaren Computersprache erzeugt werden kann. Hierfür existieren mehrere Sprachen, die mehr oder weniger alle den gleichen Funktionsumfang haben.

Die Business Process Execution Language, kurz BPEL, ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren einzelne Aktivitäten durch Webservices implementiert werden können. Ursprüngliche Bezeichnung: Business Process Execution Language for Web Services (BPEL4WS)

Die im Jahr 2002 von IBM, BEA und Microsoft eingeführte Sprache wird dabei zur Beschreibung von sogenannten Webservice-Orchestrierungen verwendet. Die Beschreibung selbst wird ebenfalls in Form eines Webservice bereitgestellt und kann als ein solcher verwendet werden.

Durch die Abstraktion mittels BPEL kann die Schnittstelle eines Webservice, der die an einem Prozess beteiligten Web Services steuert, beschrieben werden – beispielsweise in welcher Reihenfolge Nachrichten eintreffen müssen

Ausführbare BPEL-Prozesse können auf einer Workflow Maschine zum Einsatz gebracht werden (engl. deployed). Abstrakte Prozesse dienen der Beschreibung des Verhaltens des Prozesses („behavioral interface“). Sie werden als Sicht auf einen ausführbaren Prozess verwendet und dienen dazu, das interne Verhalten des Prozesses zu verbergen.

Das Ideal ist, aus einer BPEL Beschreibung eines Geschäftsprozesses den Code für ein SOA Workflow Programm automatisch zu generieren.

Hierfür existiert ein Beispiel, implementiert mittels einer Dissertation und mehrerer Diplomarbeiten, durchgeführt in enger Kooperation mit dem Werk Sindelfingen der Daimler AG:

Michael Herrmann: Service-orientierte Architektur (SOA), Identifizierung äquivalenter Services in Form semantischer semiautomatischer Unterstützung des EMEO-Layers.
Dissertation Institut für Informatik, Universität Leipzig, 2008.
http://cedix.de/DiplArb/MichHerr.pdf

Oliver Dalferth: Exemplary Implementation of a Purchase Requisition Process according to the Principles of a Service-Oriented Architecture. Diplomarbeit Eberhard-Karls-Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, August 2007,
http://cedix.de/DiplArb/Dalferth.pdf

20.4.12 Facit

SOA ist derzeitig eine Vision, gekennzeichnet durch viel Hype und wenig Realität. Dennoch, es ist die Lösung der Zukunft.

Der Weg dorthin ist lang, die Probleme sind teilweise ungelöst und die Schwierigkeiten einer Implementierung enorm. Zu den existierenden Software Produkten gehören heute vor allem

- **IBM WebSphere Product Family, einschl. des WebSphere Application Servers(WAS)**
- **Oracle Fusion Product Family, einschl. des WebLogic Application Servers**
- **Software AG webMethods Product Family, einschl. des ARIS Platform.**

All dies sind jedoch nur Teillösungen, unvollständig und verbesserungsfähig. Über die zu implementierende Software Architektur besteht wenig Einigkeit.

Die „Enterprise Service Bus“ (ESB) SOA Implementierung durch WebSphere ist eine Middleware Software Architektur, die fundamentale Services für komplexere Systeme zur Verfügung stellt. ESB ist eine flexible Infrastruktur zur Integration von Applikationen und Services. Es nutzt Java und J2EE/JEE Funktionalität sowie Web Services zur Kommunikation zwischen den Komponenten. Ein ESB bietet die Konnektivität für die Implementierung einer serviceorientierten Architektur (SOA). Die Basis sind prinzipiell XML-basierte Daten. WebSphere ESB stellt Funktionen zur Verfügung, die die Transformation von unterschiedlichen Datenformaten, so unter anderem auch XML und Java, in die benötigten Zielformate unterstützen. Zudem sind das Routing von Daten zwischen Services und das entsprechend konfigurierte Erzeugen und Verarbeiten von Business Events Funktionen des ESB.

WebSphere ESB wird derzeitig von vielen Fachleuten als ein erfolgversprechender Ansatz für die Implementierung einer SOA Architektur angesehen.

Schlusswort

und das war es.

Haben Sie sich wirklich durch alle Teilgebiete dieses Lehrtextes durchgearbeitet ? Wenn ja, verstehen Sie nun sicherlich, warum Enterprise Computing ein sehr wichtiges Teilgebiet in der Informatik darstellt.

Fachgebiete wie Web 3.0 (Facebook), Tablet Betriebssysteme oder BioInformatik sind sicher sehr wichtige Teilgebiete der Informatik, aber für Enterprise Computing trifft dies ebenfalls zu.

Herzlichen Glückwunsch. Wir wünschen Ihnen , dass sie das erworbene Wissen erfolgreich für die Verbesserung der Enterprise Computing Infrastruktur in Ihrer Organisation einsetzen können.

20.5 Weiterführende Information

Eine gute Beschreibung der Webservices ist in dieser Presentation enthalten:
http://www-01.ibm.com/support/docview.wss?uid=swg27010693&aid=1
oder
http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/Webservice01.pdf

Eine weitere Beschreibung finden Sie in Kapitel 18 des (kostenlosen) Lehrbuches „ Java ist auch eine Insel
http://openbook.galileodesign.de/javainsel5/javainsel18_000.htm

Eine detaillierte Beschreibung finden Sie im WebSphere Version 6 Web Services Handbook:
http://www.redbooks.ibm.com/abstracts/sg246461.html

Heute ist SOA reich an Hype und arm an konkreten Implementierungen. Nicht überraschend existieren zahlreiche Videos zum Thema SOA, z.B.

http://www.youtube.com/watch?v=sbd_1G8Kgjs

http://www.youtube.com/watch?v=jL1oVENiYT8&feature=fvwp&NR=1

http://www.youtube.com/watch?v=u76jUBPmS1I&feature=endscreen&NR=1

http://www.youtube.com/watch?v=uW8dnVuMZRM&feature=endscreen&NR=1

http://www.youtube.com/watch?v=jl5oFtNwJ9Q&feature=endscreen&NR=1

Um so wichtiger sind Erfahrungsberichte aus der Wirtschaft und öffentlichen Verwaltung. Eine Zusammenfassung finden Sie unter

http://www-01.ibm.com/software/solutions/soa/soa_videos.html

http://www.youtube.com/watch?v=aF5WAqMyFQw

Literatur zum Thema Enterprise Service Bus :

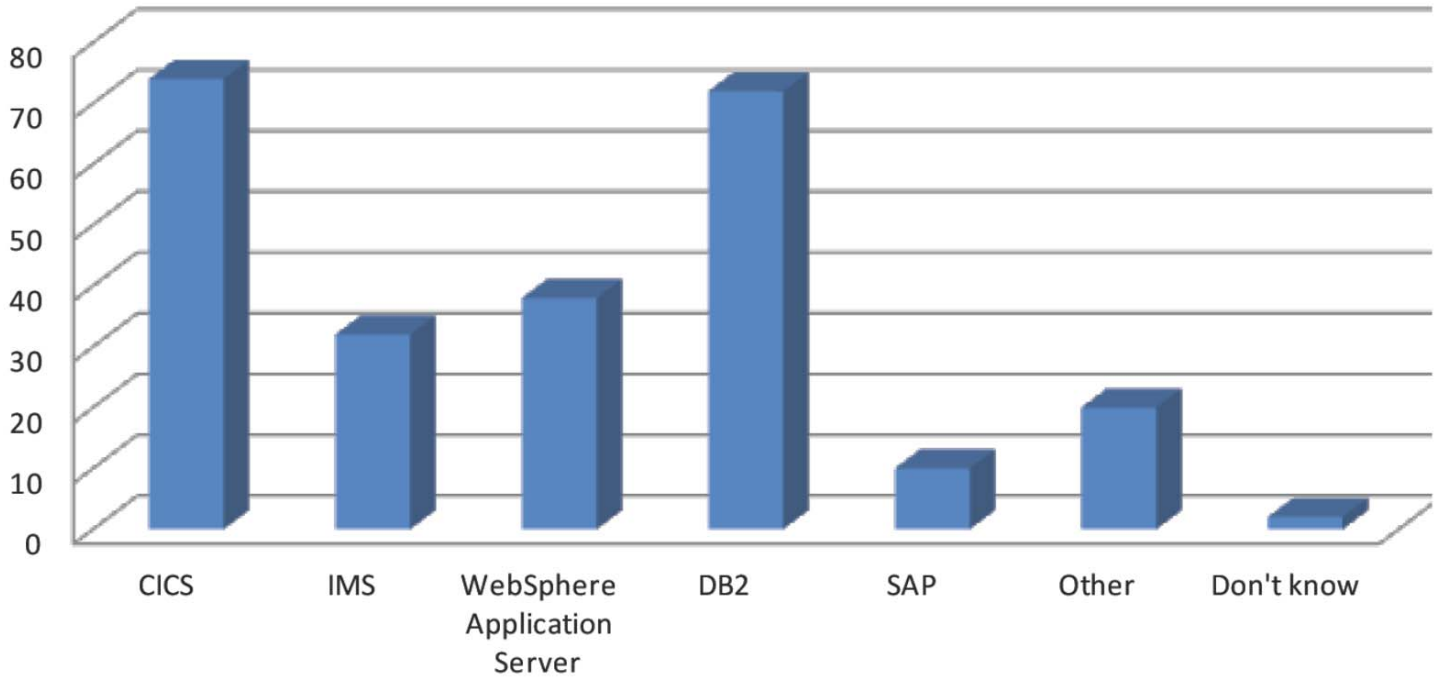
http://www.ibm.com/developerworks/websphere/techjournal/0501_reinitz/0501_reinitz.html

http://www.ibm.com/developerworks/websphere/techjournal/0502_reinitz/0502_reinitz.html

http://www.redbooks.ibm.com/abstracts/sg246346.html.

http://www.redbooks.ibm.com/redbooks/pdfs/sg247538.pdf

Die häufigste z/OS Middleware



Die hier dargestellte Grafik aus dem Arcati Yearbook 2013 zeigt auf, welche z/OS Middleware derzeitig wie häufig für Web Services enabled ist. CICS und DB2 sind die wichtigsten Subsysteme.

IMS fällt zurück, behauptet aber seine Position. WebSphere ist wichtig, weil die Mehrzahl der Instanzen nicht unter z/OS laufen, sondern als Distributed Systems auf z/OS Subsysteme zugreifen.

21. Abkürzungen - Akronyme

ABEND	**Abnormal End**
ADS	**Application Data Structure (CICS).**
AIX	**Advanced Interactive Executive (the IBM Unix based operating System)**
ALU	**Arithmetic Logic Unit**
AMS	**Access Method Services**
AOR	**Application owning Region (CICS)**
API	**Application Programming Interface**
APPC	**Advanced Program-to Profram Communication (SNA)**
APPN	**Application to Application Node (SNA)**
ARM	**Automatic Restart Manager (Sysplex)**
ASCII	**American Standard Code for Information Interchange**
ASM	**System z Assembler Language**
ATM	**Automatic Teller Machine**
ATM	**Asynchronous Tramsfer Mode**
B2B	**Business to Business**
BCP	**Base Control Program**
BCR	**Byte Count Register**
BCS	**Basic Catalog Structure**
BI	**Business Intelligence**
BLI	**Business Logic Interface (CICS)**
BM	**Buffer Manager (DB2)**
BMS	**Basic Mapping Support (CICS zeichenorientierte Präsentationslogik)**
BP	**Buffer Pool (Datenbanken)**
BPEL	**Business Process Execution Language**
CA	**Control Area**
CA	**Computer Associates**
CB	**Component Broker**
CCI	**Common Client Interface**
CCF	**Common Connector Framework**
CCU	**Cluster Control Unit (z.B. für IBM 3174)**
CCW	**Channel Command Word (S/360 und System z Ein-/Ausgabe Architektur)**
CEC	**Maschine mit Prozessoren**
CEDE	**Channel End Device End**
CF	**Coupling Facility**
CFCC	**Coupling Facility Control Code**
CFRM	**Coupling Facility Resource Manager**
CHPID	**Channel Path Identifier**
CI	**Control Interval**
CICS	**Customer Information Control System**
CICSPlex	**Collection of related CICS Regions**
CICS TS	**CICS Transaction Server**
CIDF	**Control Interval Definition Field**
CKD	**Count Key Data**
CMS	**Conversational Monitor System**
CO	**Cast Out (DB2)**

COBOL	**Common Business Oriented Language**
COM	**Component Object Model (Microsoft)**
CORBA	**Common Object Request Broker Architecture**
COS	**CORBA Common Object Services**
CP	**Central Processor, also called a CPU**
CPC	**Central Processor Complex, also called a “System", frequently an SMP**
CPI-C	**Common Programming Interface for Communication (SNA RPC)**
CMPL	**Current Multiprogramming Level**
CPU	**Central Processing Unit**
CRM	**Customer Relationship Management**
CRUD	**Create, Read, Update, Delete**
CS	**Central Storage**
CSD	**CICS System Definition**
CSS	**Channel Subsystem**
CTC	**Channel to Channel**
CUA	**Common User Access**
CUI	**Character User Interface**
CWI	**CICS Web Interface**
DA	**Device Adapter**
DASD	**Direct Access Storage Device, IBM terminology for disk drive**
DB	**Data Base**
DB2	**Data Base 2 , the major multiple platform IBM relational data base product**
DBCTL	**Database Control Subsystem**
DBMS	**Database Management System**
DBRC	**Database Recovery Control (IMS)**
DBRM	**Data Base Request Module (DB2)**
DFSMS	**Data Facility / System Managed Storage (z/OS)**
DCA	**Distributed Converter Assembly**
DCE	**Distributed Computing Environment (Open System Foundation)**
DCM	**Dynamic Channel Path Management (IRD)**
DCOM	**Distributed Component Object Model (Microsoft Corporation)**
DDL	**Data Definition Language**
DFSMS	**Data Facility Storage Management Subsystem)**
DHTML	**Dynamic HTML (provides support for SHTML)**
DIMM	**Dual Inline Memory Module**
DLI	**Data Language/1**
DMA	**Direct Memory Access**
DOM	**Document Object Model (XML)**
DPL	**Distributed Program Link (CICS)**
DPS	**Dynamic Path Selection**
DRAM	**Dynamic Random Access Memory**
DTB	**Dynamic Transaction Backout**
DTD	**Document Type Definition (XML)**
DTP	**Distributed Transaction Programming (CICS ISC)**
E/A	**Ein-/Ausgabe**
EBCDIC	**Extended Binary Coded Decimals Interchange Code**
ECI	**Extemal Call Interface**
ECKD	**Extented Count Key Data**
EHL	**Explicit Hierarchical Locking (DB2)**
EJB	**Enterprise Java Beans**
EIS	**Enterprise Information System**
EPI	**External Programming Interface**
ERP	**Enterprise Resource Planning**
ESA	**Enterprise System Architecture (S/390)**
ESDS	**Entry Sequence Dataset (VSAM sequential file)**

ETL	**Extraktion, Transformation, Laden**
ETR	**Extemal Time Reference (Sysplex Timer)**
ETS	**External Time SourceEXCI Extemal CICS Interface (z/OS cross-memory interface)**
FC	**Fibre Channel**
FC-AL	**Fibre Channel Arbitrated Loop**
FCB	**Fabric Book Connectivity (MCM Interconnect)**
FCP	**Fibre Channel (FICON) Protocol**
FCS	**Fibre Channel Standard**
FCT	**File Control Table**
FC-SW	**Fibre Channel Switched Fabric**
FICON	**Fibre Channel Connection**
FIFO	**First In First Out**
FOR	**File owning Region (CICS)**
FSP	**Flexible Support Processor**
FTE	**Full-Time Equivalent**
GDPS	**Geographically Dispersed Parallel Sysplex**
GLM	**Global Lock Manager (DB2, CF)**
GRS	**Global Resource Serialization**
GUI	**Graphical User Interface**
HA	**High Availability**
HA	**Host Adapter**
HACL	**Host Access Class Library (CICS screen scraping)**
HATS	**Host Access Transformation Services**
HCA	**Host Channel Adapter**
HCD	**Hardware Control Definition**
HCM	**Hardware Configuration Manager**
HFS	**Hierarchical File System (Unix System Services)**
HMC	**Hardware Management Console**
HPJ	**High-Perförmance Java Compiler**
HPO	**High Performance Option**
HP-UX	**Hewlett Packard Unix Operating System**
HSA	**Hardware Storage Area**
HSM	**Hierarchical Storage Management**
HTTP	**HyperText Transfer Protocol**
ICAPI	**nternet Connection Application Programming Interface (z/OS Web Server)**
ICB	**Integrated Cluster Bus (z/OS hardware)**
ICF	**Integrated Coupling Facility**
ICSS	**Internet Connection Secure Server**
IDAA	**IBM DB2 Analytics Accelerator for z/OS**
IDE	**Integrated Development Environment**
IDL	**Interface Definition Language**
IEDN	**Intraensemble Data Network**
IEF	**Interpretive Execution Facility**
IFL	**Integrated Facility for Linux**
IIOP	**Internet Inter-ORB Protocol**
IMAP	**Internet Message Access Protocol**
IML	**Initial Microcode Load**
IMS	**Information Management System, non-relational Data Base System**
IMS/DB	**Information Management System Database Manager**
IMS/TM	**Information Management System Transaction Manager**
INIT	**Initiator**
INMN	**Intranode Management Network**
IND	**Interactive Network Dispatcher (WebSphere)**
IOCP	**/O Configuration Program**
IOCDS	**IO Configuration Data Set**

IOR	**Interoperable Object Reference**
IPL	**Initial Program Load**
IPX/SPX	**Intemetwork Packet Exchange/Sequenced Packet Exchange (Novell)**
IRC	**Interregion Communication**
IRD	**Intelligent Resource Director (PR/SM, Work Load Manager)**
IRLM	**Intemal Resource Lock Manager (DB2)**
ISAM	**Indexed Sequential Access Method**
ISC	**CICS Intersystem Communication**
ISPF	**Interactive System Productivity Facility**
ISPF/PDF	**Interactive System Productivity Facility/Program Development Facility**
ITOC	**IMS to TCP/IP OTMA Connection**
IUCV	**Inter User Communication Vehicle**
J2EE	**Java Enterprise Edition**
JAR	**Java Archive**
JCA	**Java Connector Architecture**
JCL	**Job Control Language**
JDBC	**Java Database Connectivity**
JDK	**Java Development Kit**
JEE	**Java Enterprise Edition**
JES2	**Job Entry System 2**
JES3	**Job Enty System 3**
JIDL	**Java Interface Definition Language**
JSP	**Java Server Page**
JIT	**Just in Time Compiler**
JMAPI	**Java Management API**
JMS	**JMAPI Java Management API, Java Message Service**
JNDI	**Java Naming and Directory Interface**
JNI	**Java Native Interface**
JRMP	**Java Remote Message Protocol**
JSP	**Javaserver Pages (WebSphere)**
JTS	**Java Transaction Service**
JVM	**Java Vjrtual Machine**
KVM	**Kernel based Virtual Machine (Hypervisor)**
KSDS	**Keyed Sequence Dataset (VSAM)**
L	**Logon**
LAN	**Local Area Network**
LCSS	**Logical Channel Subsystems**
LDAP	**Lightweight Directory Access Protocol**
LDS	**Linear Dataset**
LE	**Langage Environment**
LIC	**Licensed Intemal Code, a kind of microcode**
LIFO	**Last In First Out**
LLM	**Local Lock Manager (DB2)**
LMB	**Logical Memory Block (LPAR)**
LPAR	**Logical Partition**
LPSW	**Load Program status Word (Maschinenbefehl)**
LSQA	**Local System Queue Area**
LU	**Logical Unit (SNA)**
LUW	**Logical Unit of Work (Transaction Processing)**
LV	**Logical Volume**
MBA	**Memory Bus Adapter (System z Hardware)**
MBQ	**Message based Queuing, sysnonym for MOM**
MCA	**Message Channel Agent**
MCM	**Multi Chip Module**
MDB	**Message Driven Bean**

MIF	**Multiple Image Facility**
MIME	**Multipurpose Internet Mail Extension**
MLC	**Multi Layer Ceramic**
MOM	**Message Oriented Middleware, sysnonym for MBQ**
MPL	**Multiprogramming Level**
MQI	**Message Queue Interface (MQSeries)**
MQOD	**Message Queue Object Descriptor**
MRO	**CICS Multiregion Operation**
MSC	**Multiple Systems Coupling**
MVC	**Model-View Controller Triade**
MVS	**Multiple Virtual Storage**
NAU	**Network Addressable Unit (SNA).**
NCF	**Network Computing Framework (IBM)**
NetBIOS	**Network Basic Input/Output System**
NNCP	**Network-Node Control Point (SNA)**
NSM	**Network- und System Management, Synonym for System Management**
NTP	**Network Time Protocol**
NUMA	**Non Uniform Memory Architecture**
NVS	**Non Volatile Store**
OE	**OS/390 OpenEdition, also known as OS/390 Unix System Services (USS)**
OLTP	**Online Transaction Processing**
OMG	**Object Management Group**
ONC	**Open Network Comuting (SUN)**
OO Cobol	**Object oriented Cobol**
ORB	**Object Request Broker**
OS	**Operating System**
OSA	**Open System Adapter (System z Hardware)**
OSAE	**OSA Express (System z Hardware)**
OSF	**Open System Foundation**
OSI	**Open Systems Interconnection**
OTM	**Object Transaction Monitor**
OTMA	**Outside Transaction Managed Access (IMS)**
OTS	**Object Transaction Service (CORBA)**
PAV	**Parallel Access Volume**
PCB	**Process Control Block**
PDS	**Partitioned Dataset**
PDS/E	**Partitioned Dataset Extended**
PI	**Performance Index (WLM)**
POR	**Power On Reset**
PR/SM	**Processor Resource/System Manager (Virtual Machine Manager, System z Hardware)**
PTF	**Problem Trouble Fix**
PU	**Physical Unit (SNA)**
PU	**Processing Unit (andere Bezeichnung für CPU)**
QDIO	**Queued Direct I/O (Mainframe Hardware, OSA Adapter)**
QMF	**Query Management Facility**
RA	**Resource Adapter (JCA)**
RACF	**Resource Access Control Facility (z(OS).**
RAD	**Rapid Application Development**
RCT	**Region Control Task**
RDB	**Relational Data Base**
RDBMS	**Relational Data Base Management System**
RDF	**Record Definition Field**
RDO	**Resource Definition Online**
RDz	**Rational Developer for System z**

RECON	**Recovery Control (IMS)**
REXX	**Restructured Extended Executor language**
RISC	**Reduced Instruction Set Computer**
RLS	**Record Level Sharing (CICS, VSAM)**
RMF	**Resource Management Facility**
RMI	**Remote Method Invocation**
ROR	**Ressource owning Region (CICS)**
RP	**Remote Procedure (CICS TCB)**
RPC	**Remote Procedure Call**
RPC	**Remote Procedure Call**
RRDS	**Relative Record Datasets (VSAM byte access mode)**
RRS	**Resource Recovery Services (z/OS)**
RU	**Request Unit (SNA)**
R/W	**Read/Write**
SAC	**Shareable application class loader**
SADR	**Storage Address Register**
SAF	**System Authorization Facility, (z/OS, call interface to RACF)**
SAN	**Storage Area Netzwerk**
SAP	**System Assist Processor**
SAP AG	**Systeme Anwendungen Produkte AG, Walldorf**
SATA	**Serial ATA**
SCLM	**Software Configuration Library Manager**
SCM	**Supply Chain Management**
SCP	**System Control Program**
SCRT	**Sub Capacity Reporting Tool**
SCSI	**Small Computer System Interface (pronounced “scassi)**
SDA	**Shared Data Architecture**
SDA	**Shared Disk Architecture**
SE	**Support Element**
SFM	**Sysplex Failure Management**
SFS	**Structured File Server**
SHTML	**Server-side HyperText Markup Language**
SM	**Systems Management**
SMF	**System Management Facility**
SMIT	**System Management Interface Tool**
SMP	**Symmetric Multi-Processor, multiple CPUs sharing a single z/OS instance**
SMQ	**IMS Shared Message Queue (CF)**
SMS	**System Managed Storage (z/OS)**
SNA	**System Network Architecture**
SOA	**Service Oriented Architecture**
SOAP	**Simple Object Access Protocol (Web Services)**
Solaris	**Sun Unix based operating System**
SP	**Stored Procedure**
SPAS	**Stored Procedure Address Space (DB2)**
SPUFI	**SQL Processing Using File Input**
SQA	**System Queue Area**
SQL	**Structured Query Language**
SQLJ	**Java Structured Query Language**
SRB	**Service Request Block (z/OS)**
SRM	**System Resource Manager**
SNCP	**Single-Node Control Point (SNA)**
SSCP	**System Services Control Point (SNA)**
SSCH	**Start Sub Channel**
SSI	**Server-side Include (HTML)**
STP	**Server Time Protokoll**

SVC	**Supervisor Call**
TP Monitor	**Transaction Processing Monitor, Teleprocessing Monitor**
TCAS	**Terminal Control Address Space**
TCB	**Task Control Block**
TCM	**Thermal Conduction Module**
TCO	**Total Cost of Ownership**
TCP/IP	**Transmission Control Protocol/Intemet Protocol**
TMPL	**Target Multiprogramming Level**
TOR	**Terminal owning Region (CICS)**
TPF	**Transaction Processing Facility**
TRID	**Transaction Identification (Transaktionsmonitor)**
TSAF	**Transparent Services Access Facility**
TSM	**Tivoli Storage Manager**
TSO	**Time Sharing Option(z/OS)**
TXSeries	**Transactional Middleware für Linux und WindowsPlatforms (IBM)**
UCB	**Unit Control Block**
UCW	**United Channel Word**
UDB	**Universal Data Base, DB2 version for Unix and NT platforms**
UDDI	**Universal Discovery, Description, and Integration (Web Services)**
UI Record	**User Interface Record (Rational Developer, CICS MAP equivalent)**
UR	**Unit Record**
URI	**Uniform Resource Identifier**
URL	**Uniform Resource Locator**
URM	**Unified Resource Manager**
USS	**z/OS Unix System Services, also known äs OpenEdition**
VM	**Virtual Machine**
VS	**Virtual System**
VSE	**Virtual Storage Extended, System z Operating System**
VSAM	**Virtual Storage Access Method**
VTAM	**Virtual Telecommunications Access Method**
VTOC	**Volume Table Of Content**
VTS	**Virtual Tape Server**
W3C	**Worid Wide Web Consortium (HTML, XML)**
WAN	**Wide Area Network**
WAS	**WebSpere Web Application Server**
WLM	**Work Load Manager (z/OS, SysPlex)**
WSDL	**Web Services Description Language (Web Services)**
XA	**Extended Architecture (S/370)**
XCF	**Cross-System Communication Facility**
XDR	**Extemal Data Representation (SUN RPC)**
XES	**Cross System Extemsion Services (z/OS access to CF)**
XI	**Cross Invalidate (DB2)**
XML	**Extensible Markup Language**
XRF	**Extended Recovery Facility (OS/390)**
zAAP	**zSeries Application Assist Processor (Java Processor)**
zBX	**zEnterprise Blade Center Extension**
zIIP	**zSeries Integrated Information Processor (DRDA Processor)**

22. Index