VERTEILTE SYSTEME

Kapitel 1: Einleitung

Definition: Verteiltes System	Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes zusammenhängendes (kohärentes) System erscheinen.
Hauptziel	Hauptziel eines verteilten Systems ist es, den Zugriff auf entfernte Ressourcen einfach zu machen, sodass diese mit anderen Benutzern kontrolliert und effizient gemeinsam genutzt werden können.
Ziele / Vorteile	Der Aufbau eines verteilten Systems ist dann sinnvoll, wenn: 1. Ein verteiltes System **Ressourcen leicht zugreifbar** macht. 2. Ein verteiltes System die Tatsache verbirgt, dass **Ressourcen über ein Netzwerk verteilt** sind. 3. Es **offen** ist. (Standardisierte Schnittstellen) 4. Es **skalierbar** ist. (Grösse, Geografie, Administration) -> Leistungsprobleme
Nachteile	Komplexität, weniger Leistung, weniger Sicherheit
Verteilungs-transparenz	Ein verteiltes System, das in der Lage ist, sich Benutzern und Anwendungen so darzustellen, als sei es nur ein einziges Computersystem, wird als **transparent** bezeichnet
	Verteilte Systeme versuchen häufig die Verteilung von Prozessen, Daten und der Steuerung zu verbergen Formen: Zugriff, Ort, Migration, Relokation, Replikation, Nebenläufigkeit, Fehler
Middleware	Um heterogene Computer und Netzwerke zu unterstützen und gleichzeitig das Erscheinungsbild eines einzigen Systems anzubieten, werden verteilte Systeme häufig mithilfe einer Verteilten Systemschicht (**Middleware**) angeordnet. Diese Softwareschicht liegt zwischen Anwendung und Betriebssystem.
Skalierungstechniken	**Verbergen der Latenzzeiten**: Lieber sinnvolle Arbeiten als auf weit entfernte Antworten zu warten. Verteilung: Komponenten in kleinere Teile zerlegen und diese verteilen. (DNS, WWW) Replikation: damit nahe liegende Kopien benutzt werden können. Sonderform: Caching

Arten & Klassen	Verteilte Computersysteme
	• **Cluster Computer** (Beowulf-Cluster), homogen, einfach Computer
	• **Grid Computer**, heterogen, für die Zusammenarbeit aus unterschiedlichen Domänen
	Verteilte Informationssysteme
	• **Systeme zur Transaktionsverarbeitung** (Bündeln von Anforderungen -> Transaktion), ACID
	• **Systeme zur Integration von Unternehmensanwendungen** (Java EE), direkte Kommunikation:
	  ○ RPC: Aufruf einer lokalen Prozedur
	  ○ RMI: Entfernter Methodenaufruf, Aufruf von entfernten Objekten
	  ○ MOM: Warteschlangensysteme, um enge Kopplung zu vermeiden
	Verteilte pervasive (=umsichgreifende) Systeme
	• **Haus- und Multimedia-Systeme**
	• **Sensornetze**, (z.B: im Altersheim Kippsensor)

Kapitel 2: Architekturen

Definition: Architektur	Eine **Softwarearchitektur** definiert die logische Anordnung der Komponenten und deren Zusammenarbeit. Wie arbeiten die Komponenten zusammen? Wie sind sie strukturiert?
	Eine **Systemarchitektur** definiert wo die Komponenten auf den verschiedenen Computern platziert sind. (Physikalische Realisierung, Instanziierung einer Softwarearchitektur).
Architekturstile	• Geschichtete Architekturen (Internetstapel, N-Tier Web-Applikationen) • Objektbasierte Architekturen (loser Verbund von Objekten, RPC, CORBA, RMI) • Datenzentrierte Architekturen (gemeinsamer Datenraum = Repository) • Ereignisbasierte Architekturen (Ereignisbus, à la Middleware) • Dienstbasierte Architekturen (nicht behandelt)

Systemarchitekturen	• Zentralisierte Architekturen
	  ○ Client/Server Modell (Thin-Client / Rich-Client)
	• Dezentralisierte Architekturen
	  ○ Peer-2-Peer Architekturen (Strukturiert / Unstrukturiert)
	• Hybridarchitekturen (z.B. BitTorrent)

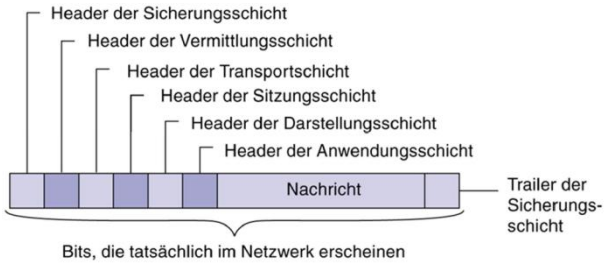
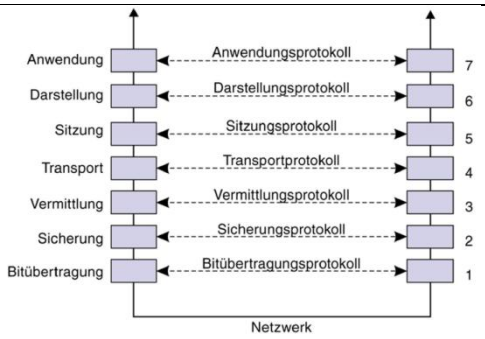
Interzeptor	Softwarekonstrukt, das den gewöhnlichen Programmfluss unterbricht und es anderem Code gestattet ausgeführt zu werden.
Adaptive Software	Middleware ist verantwortlich, um auf Veränderungen zu reagieren (nicht die Anwendungen) 1. Trennung der Belange 2. Reflektion 3. Komponentenbasiertes Design
Selbstverwaltende Systeme	Selbstverwaltenden Systeme lassen sich mit einer Rückkopplungsschleife gestalten und bestehen aus: • Überwachungskomponente(Verhalten messen) • Analysekomponente (Anpassung feststellen) • Instrumente zur Verhaltensänderung

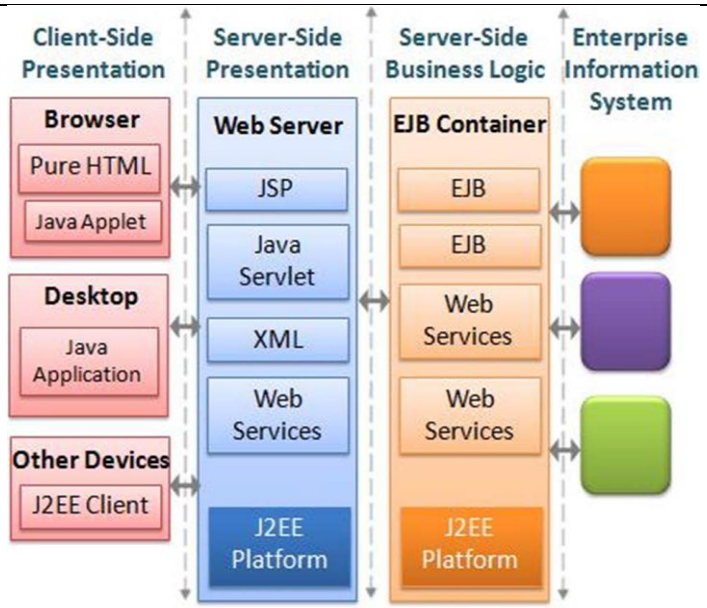
Kapitel 3: Prozesse

Threads	**Problem**: Ein Prozess mit nur einem Thread wird vollständig blockiert, sobald ein blockierender Systemaufruf ausgeführt wird. **Lösung**: Threads sind in verteilten Systemen nützlich, um die CPU auch während der Durchführung einer blockierenden I/O-Operation weiter nutzen zu können. **Konsequenz**: Threads ermöglichen hochgradig effiziente Server, die mehrere Aufgaben parallel ausführen. Jedoch ist die Programmierung etwas komplizierter. **Implementierung:** • Benutzerthread (günstig erstellt/zerstört, Blockade blockiert ganzer Prozess) • Kernelthread (muss vom Kernel ausgeführt werden, nicht günstiger als ein Prozess) • Lightweight-Prozess (LWPs) Mischform (billig, ohne eingreifen des Kernels, verborgen)
Definition Virtualisierung	Virtualisierung bezeichnet die Erzeugung von virtuellen (d.h. nicht physikalischen) Dingen wie einer emulierten Hardware, eines Betriebssystems, Datenspeichers oder Netzwerkressource.
	Einführung einer Abstraktionsschicht zwischen Anwender und Ressource, um andere physische Gegebenheiten vorzugeben, als tatsächlich vorhanden.
	Ressourcenvirtualisierung erlaubt das Zusammenfügen mehrerer (heterogener) Hardwareressourcen zu einer homogenen Umgebung.
Vorteile von Virtualisierungen	Virtualisierung ist ein wichtiger Mechanismus in verteilten Systemen, weil sie einen hohen Grad an Portabilität und Flexibilität erlaubt. Anwendungen laufen unabhängig von der zugrundeliegenden Hardware / Plattform. • Alte Software am Leben erhalten • Plattformunterschiede reduzieren • Einfache Verwaltung virtualisierter Umgebungen
Arten von Virtualisierungen	• Virtuelle Prozessmaschinen (z.B. Java, Wine, etc.) • Virtual Machine Monitore, VMM (z.B. VMWare, Xen, etc.)
Clients	**Aufgabe**: Benutzern eine Möglichkeit zur Interaktion mit einem entfernten Server zu bieten **Thin-Client** beliebter da: allgegenwärtige Internetanbindung, erleichterte Systemverwaltung
Serverkonstruktion	• Iterativ – schickt Antwort selbst zurück • Parallel (nebenläufig) – übergibt sie einem separatem Thread, der antwortet • Endlicher Zustandsautomat • Zustandslos (stateless) – keine Information über den Status seiner Clients • Zustandsbehaftet (statefull) – kennt Status seiner Clients • Soft state (Zustandsinformation für begrenzte Zeit)
Definition: Servercluster	Servercluster = Ansammlung von Computern, die über ein Netzwerk verbunden sind, wobei jeder Rechner einen oder mehrere Server ausführt.
	Server kann man zu einem Cluster gliedern, die Details des Clusters werden verborgen, ein einzelner Eintrittspunkt nimmt Anforderungen entgegen und verteilt sie an die Server im Cluster
Codemigration in VS	**Definition:** Codemigration ist das Verschieben von Code von einem Computer auf einen anderen. **Gründe:** • Leistungssteigerung (Berechnungen werden vom Server auf den Client verlagert, um den Client eine lokale Verarbeitung durchführen zu lassen • Erhöhte Flexibilität (Clients können die Software zur Kommunikation mit einem Server auch dynamisch laden **Probleme**: Probleme bei der Codemigration treffen aufgrund von lokalen Ressourcen auf.

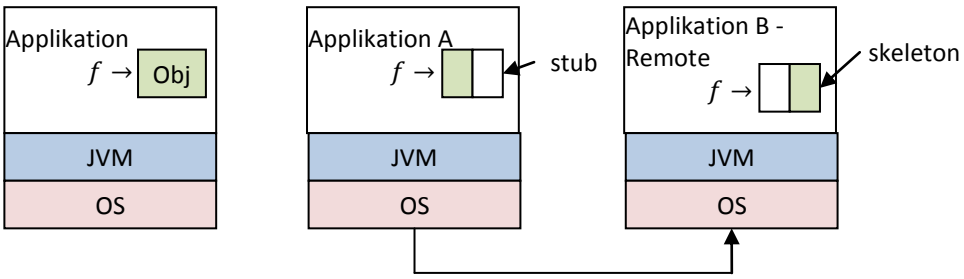
Kapitel 4: Kommunikation

Kommunikations-modelle	Entfernter Prozeduraufruf (Remote Procedure Call, RPC) • Ideal für Client/Server-Anwendungen Nachrichtenorientierte Middleware (Message-Oriented Middleware, MOM) • Wie e-Mail Data Streaming für multimediale verteilte Systeme • Zeitliche Steuerung unterliegt Einschränkungen

OSI-Modell	=Open Systems Interconnection 7 Schichten  
Middleware Protokolle	Es gibt zahlreiche anwendungsunabhängige Protokolle (Protokolle für allgemeine Zwecke, die nicht Transportprotokolle sind), sog. Middleware Protokolle, die eine Vielfalt von Diensten unterstützen: • Authentifizierungsprotokolle • Commit-Protokolle • Sperrprotokolle • Middleware Kommunikationsprotokolle ○ RPC, MOM, Streaming Protokolle, Multicast-Protokolle
Klassifikation von Kommunikation	**Persistente Kommunikation:** Nachricht wird so lange von der Middleware gespeichert, wie es dauert, sie an den Empfänger auszuliefern **Flüchtige Kommunikation**: Nachricht wird nur so lange gespeichert, wie die sendende und die empfangende Anwendung ausgeführt werden
	Asynchrone Kommunikation: Der Sender fährt sofort fort, nachdem er seine Nachricht zur Übertragung abgegeben hat **Synchrone Kommunikation**: Der Sender ist gesperrt, bis er weiss, dass seine Anforderung akzeptiert wurde
RPC	**Idee**: Ein entfernter Prozeduraufruf sollte wie ein lokaler Prozeduraufruf aussehen. **Wie**: Der lokale Aufruf der Client- Prozedur wird in einen lokalen Aufruf der Server-Prozedur umgewandelt, sodass weder der Client noch der Server die Zwischenschritte oder das Vorhandensein des Netzwerks bemerken. RPCs sind synchron, d.h. der Client blockiert, bis der Server eine Antwort sendet. **Transparenz**: Alle Einzelheiten der Nachrichtenübergabe bleiben in den Stubs verborgen. **Beispiele**: RMI, SOAP, REST **Nicht geeignet für**: Integration von Datenbanksammlungen, E-Mail, Workflow
RPC-Protokolle	RPC-Protokolle definieren: • Das Format der ausgetauschten Nachrichten • Die Darstellung einfacher Datenstrukturen ○ Ganzzahlen, Zeichen, boolesche Werte, etc.
Dienst eines Servers	Die Schnittstelle zum Server besteht aus einer Sammlung von Prozeduren, die vom Client aufgerufen werden können und vom Server implementiert werden. Schnittstellen werden häufig mit einer Schnittstellenbeschreibungssprache (Interface Definition Language, IDL) erstellt. Es wird nur die Syntax beschrieben, nicht die Semantik. • Beispiele: WSDL, WADL Aus der Schnittstellenbeschreibung lassen sich Client- und Server-Stub erstellen.
RPC Alternativen	Nachrichtenorientierte flüchtige Kommunikation • Sockets • MPI (Message-Passing Interface) Nachrichtenorientierte persistente Kommunikation • Warteschlangensysteme (Message-Queuing Systems) • Nachrichtenorientierte Middleware (Message-Oriented Middleware)
Warteschlangen-systeme	Idee der Warteschlangensysteme: Anwendungen kommunizieren, indem sie Nachrichten in bestimmte Warteschlangen einfügen. Nachrichten werden über eine Folge von Kommunikationsservern weitergeleitet und am Ziel ausgeliefert, auch wenn dies beim Aussenden der Nachricht ausgeschaltet war. Warteschlangensysteme ermöglichen eine zeitlich lose gekoppelte Kommunikation. Sender und Empfänger können vollkommen unabhängig voneinander ausgeführt werden.
Datenstreams	Ein Datenstream ist eine Folge von Dateneinheiten. Ein einfacher Stream besteht aus einer Datenfolge, ein komplexer Stream aus mehreren in Beziehung zueinander stehenden Streams, sog. Substreams. • Beispiel: Übertragung eines Films, Videostream + zwei Audiostreams Bei der Kommunikation mit Streams ist es entscheidend, ob zwei aufeinanderfolgende Dateneinheiten eine zeitliche Beziehung haben oder nicht. Komplexe Streams erfordern die Synchronisierung der Substreams.

Datenstreams übertragen	Asynchron • Beispiel: Datei als Datenstrom Synchron • Max. Ende-zu-Ende-Verzögerung ist definiert • Beispiel: Sensordaten auslesen Isochron • Max. und min. Ende-zu-Ende Verzögerungen sind definiert • Beispiel: verteilte Multimediasysteme
Jitter	Als Jitter bezeichnet man das zeitliche Taktzittern bei der Übertragung von Digitalsignalen, eine leichte Genauigkeitsschwankung im Übertragungstakt. Vermeidung mit Puffer
Synchronisierung von Streams	Das Synchronisieren von Streams bedeutet, dass die zeitlichen Beziehungen zwischen den Streams aufrechterhalten werden. Synchronisierung: • Durch die Anwendung • Durch die Middleware-Schicht
3-Tier Web Application	

Sonstiges

RMI	=Remote Method Invocation ist JAVA-basiert, veraltet, da bei bestehender Verbindung über zufällig gewählten Port geht. 
CORBA	ist plattformunabhängig kommuniziert über Port 80 (HTTP)
Übersicht	

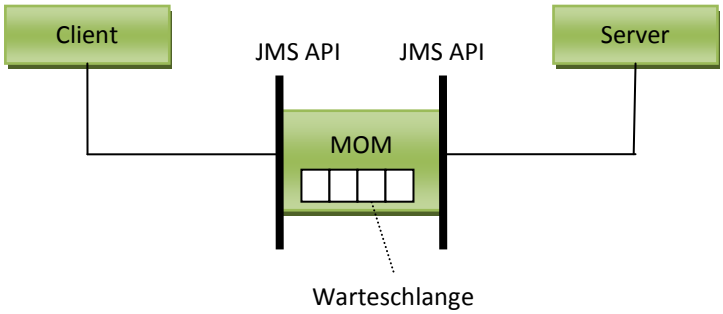
	synchron	**asynchron**
flüchtig	RPC	asRPC
persistent	MOM	

1. Objekt trennen 2. f aufrufen (als ob diese lokal sei) 3. f weiterleiten	**Server** Interface Registrieren Objekt anlegen Objekt registrieren	**Client** lookup (wurde das O registriert) f aufrufen

Server-Client Kommunikation

DYOT	**über SOAP** (“big web services”)	**RMI**	**REST** im Jahr 2000 erfunden
do your own thing	ist aus XML-RPC entstanden	ist aus CORBA entstanden	ist aus WWW / HTTP entstanden
	kein JAVA Kommunikation über <xml>	braucht JAVA	www: GET http://ntb.ch/index.html ➔ html REST: GET http://nbt.ch/Converter/CHF/EUR ➔ xml, text, JSON,
			Ideen 1) wir verwenden nun auch noch PUT und DEL 2) Hyperlinks um auf Kundendaten zu verlinken

MOM (Message oriented middleware)



JMS = Java Message Service
Interface um zu definieren, was die MOM alles kann

Eigenschaften
- zeitliche Entkopplung von Server und Client
- meist XML

Vorteile
- es müssen nicht beide gleichzeitig online sein
- ergibt sehr lose Kopplung

Nachteile
- wenn MOM ausfällt, funktioniert nichts mehr

ähnliche Systeme: Mailsystem

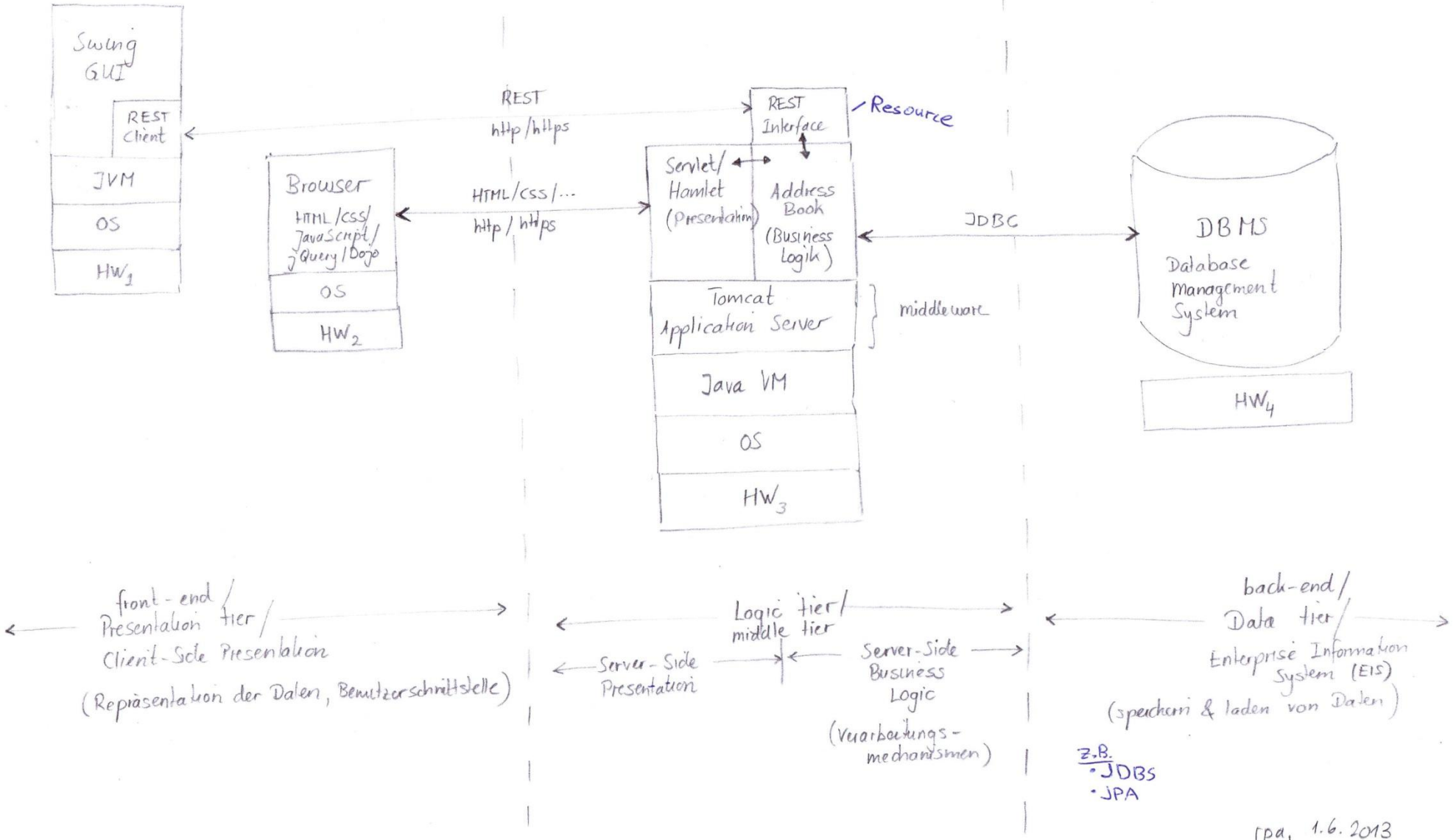
Kapitel 12:

Grundsystem	Client greift über Browser auf ein Webserver zu, der das Dokument holt und zurückschickt.	
MIME-Typen	Text	HTTP: Hypertext Transfer Protocol; XML Extensible Markup Language
	Image	GIF, JPG, ...
Webservercluster	Front-End bearbeitet alle ankommenden Anforderungen und ausgehenden Antworten. Schickt diese an diverse Webserver (über LAN)	
HTTP Methoden	Head: Anforderung der Rückgabe eines Dokumentenkopfes Get: Anforderung der Rückgabe eines Dokumentes zum Client Put: Anforderung der Speicherung eines Dokumentes Post: Verfügbarmachen von Daten, die einem Dokument hinzugefügt werden sollen Delete: Anforderung der Löschung eines Dokumentes	
SOAP	=Simple Object Access Protocol Standard für die Kommunikation mit Webdiensten SOAP-Kommunikation wird über HTTP implementiert Annahme: Beteiligte wissen wenig voneinander Besteht aus Rumpf und Kopf (optional) enthält keine Empfängeradresse sondern wird über HTTP (üblicher) oder SMTP definiert	
URIs	http	HTTP
	mailto	E-Mail
	ftp	FTP
	file	Lokale Datei
	data	Eingefügte Daten
	telnet	Anmeldung übers Netzwerk
	tel	Telefon
	modem	Modem
Leistungs-steigerungen	durch Chaches und durch Replikationen	

3-tier application: AddressBook Web-Application

vertikale Verteilung

tier = Schicht



rpa, 1.6.2013