



Grundlagen: Rechnernetze und Verteilte Systeme

Kapitel 9: Verteilte Systeme

Prof. Dr.-Ing. Georg Carle
Lehrstuhl für Netzarchitekturen und Netzdienste
Technische Universität München
carle@net.in.tum.de
http://www.net.in.tum.de



Acknowledgements:
Prof. Dr. Wolfgang Küchlin, Univ. Tübingen



Technische Universität München



Gliederung - Kapitel 9: Verteilte Systeme

- Kapitel 9
- 9.1 Grundlagen
- 9.2 Middleware
- 9.3 RPC
- 9.4 RMI
- 9.5 Service Oriented Architectures
- 9.6 Corba
- 9.7 Web-Anwendungen
- 9.8 HTML und XML
- 9.9 Web Services



Übersicht

1. Einführung und Motivation
 - Bedeutung, Beispiele
2. Begriffswelt und Standards
 - Dienst, Protokoll, Standardisierung
3. Direktverbindungsnetze
 - Fehlererkennung, Protokolle
 - Ethernet
4. Vermittlung
 - Vermittlungsprinzipien
 - Wegwahlverfahren
5. Internet-Protokolle
 - IP, ARP, DHCP, ICMP
 - Routing-Protokolle
6. Transportprotokolle
 - UDP, TCP
7. Verkehrssteuerung
 - Kriterien, Mechanismen
 - Verkehrssteuerung im Internet
8. Anwendungsorientierte Protokolle und Mechanismen
 - Netzmanagement
 - DNS, SMTP, HTTP
9. **Verteilte Systeme**
 - **Middleware**
 - **RPC, RMI**
 - **Web Services**
10. Netzsicherheit
 - Kryptographische Mechanismen und Dienste
 - Protokolle mit sicheren Diensten: IPSec etc.
 - Firewalls, Intrusion Detection
11. Nachrichtentechnik
 - Daten, Signal, Medien, Physik
12. Bitübertragungsschicht
 - Codierung
 - Modems



9.1 Definition eines Verteilten Systems

- "Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen."

[Tanenbaum, van Steen: Verteilte Systeme, Pearson Studium, 2. Auflage, 2008]

- "Ein System, mit dem man nicht arbeiten kann, weil ein Rechner ausgefallen ist, von dem man noch nie etwas gehört hat."

[Leslie Lamport]

Verteilte Systeme

Eigenschaften Verteilter Systeme

- ❑ Unterschiede zwischen den verschiedenen Computern werden verborgen
- ❑ Benutzer und Anwendungen können auf konsistente und einheitliche Weise mit dem verteilten System zusammenarbeiten
- ❑ Verteilte Systeme sollen erweiterbar und skalierbar sein

Beispiele

- ❑ Netzwerk aus Workstations mit gemeinsamen Dateidiensten und gemeinsamer Benutzerverwaltung
- ❑ Informationssystem für Arbeitsabläufe
- ❑ World Wide Web

9.2 Middleware

- ❑ Ziele einer Middleware
 - Einführung einer **zusätzlichen Schicht** zwischen Betriebssystem und Anwendung, um **höhere Abstraktionsebene** zur Unterstützung verteilter Anwendungen zu erhalten.
 - Lokale Ressourcen einzelner Knoten sollen weiterhin vom (lokalen) Betriebssystem verwaltet werden.
- ❑ Mögliche Modelle (Paradigmen) für Middleware
 - Modell „**Datei**“: Behandlung aller lokaler und entfernter Ressourcen als Datei (Beispiele: Unix, Plan9)
 - Modell „**Prozeduraufruf**“: lokaler und entfernter Prozeduraufruf (Beispiel: RPC)
 - Modell „**verteilte Objekte**“: Objekte können lokal oder auf transparente Weise entfernt aufgerufen werden (Beispiel: RMI)
 - Schnittstelle besteht aus den Methoden, die das Objekt implementiert.
 - Verteiltes System kann so realisiert werden, dass sich ein Objekt auf einer Maschine befindet, seine Schnittstelle aber auf vielen anderen Maschinen bereitgestellt wird
 - Modell „**verteilte Dokumente**“: Dokumente mit verweisen, wobei Ort des Dokuments transparent ist (Beispiel: WWW)
 - Modell „**Nachrichtenorientierte Middleware**“: Nachrichtenwarteschlangensysteme für persistente asynchrone Kommunikation

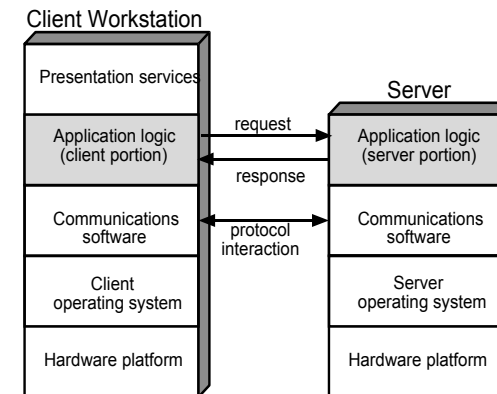
Ziele für Verteilte Systeme

- ❑ Benutzer und Ressourcen verbinden
 - Den Benutzern ermöglichen, auf entfernte Ressourcen zuzugreifen
 - Unterstützung für kontrollierte gemeinsame Benutzung
- ❑ Transparenz
 - Zugriff – verbirgt Unterschiede in der Datendarstellung
 - Position – verbirgt Ort der Ressource
 - Migration – verbirgt Möglichkeit, Ressource an anderen Ort zu verschieben
 - Relokation – verbirgt Verschiebung von Ressource während Nutzung
 - Replikation – verbirgt, dass eine Ressource repliziert ist
 - Nebenläufigkeit – verbirgt gleichzeitige Nutzung konkurrierender Benutzer
 - Fehler – verbirgt Ausfall und Wiederherstellung einer Ressource
 - Persistenz – verbirgt Speicherart (Hauptspeicher oder Festplatte)

⇒ Vor- und Nachteile von Transparenz
- ❑ Offenheit
 - Vollständige Schnittstellenspezifikation (⇒ Schnittstellendefinitionssprache IDL – Interface Description Language):
Festlegen von Namen der verfügbaren Funktionen, Typen der Übergabeparameter und Rückgabewerte
- ❑ Skalierbarkeit

Die Rolle von Middleware

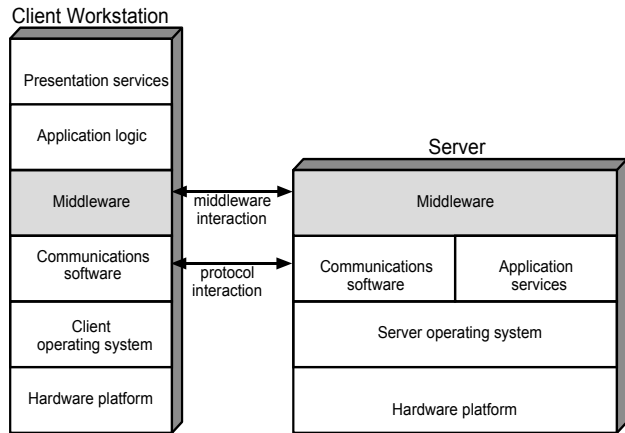
- ❑ Client/Server-Kommunikation **ohne** Middleware



Quelle: „Operating Systems“, Stallings, Abb.13-7

Die Rolle von Middleware

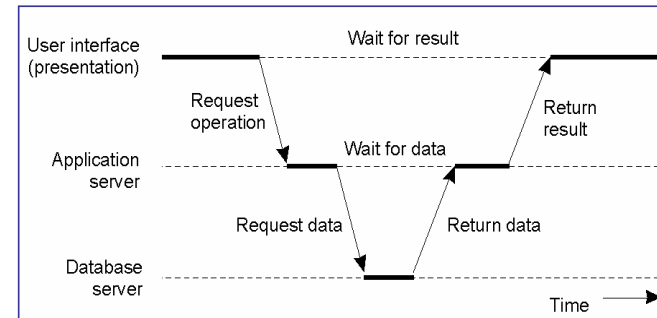
- Client/Server-Kommunikation mit Middleware



Quelle: „Operating Systems“, Stallings, Abb.13-12

Three-Tier-Modell

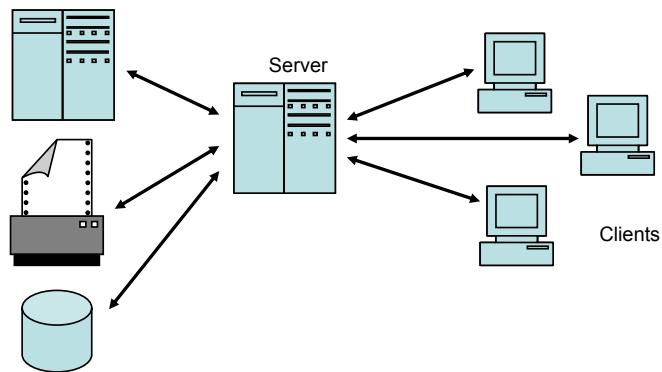
- Erweitertes request-reply Verhalten



Quelle: „Distributed Systems“, Tanenbaum, van Steen, Abb.1-30

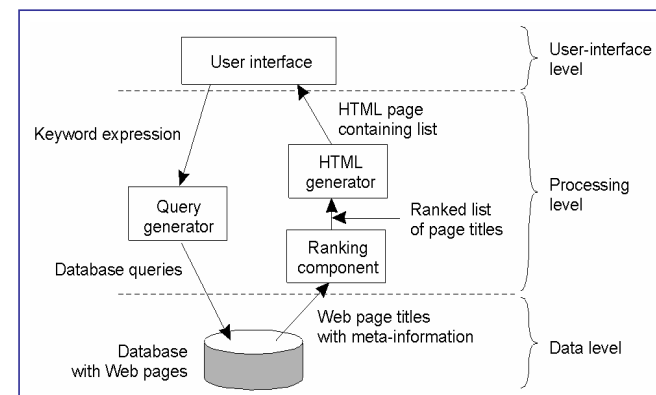
Three-Tier-Modell

- Client/Server-Modell: Two-Tier-Modell
- Server selbst ist auch Client ⇔ Three-Tier-Modell



3+ -Schichten Client/Server-Modell

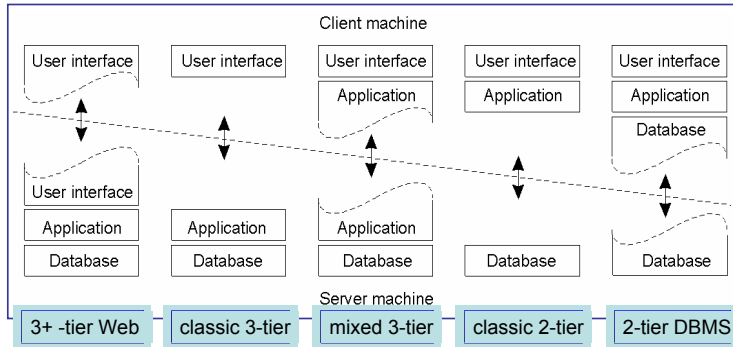
- Zwischenschichten in der Mitte, z.B. Web-Schicht
Beispiel: Suchmaschine



Quelle: „Distributed Systems“, Tanenbaum, van Steen, Abb.1-28

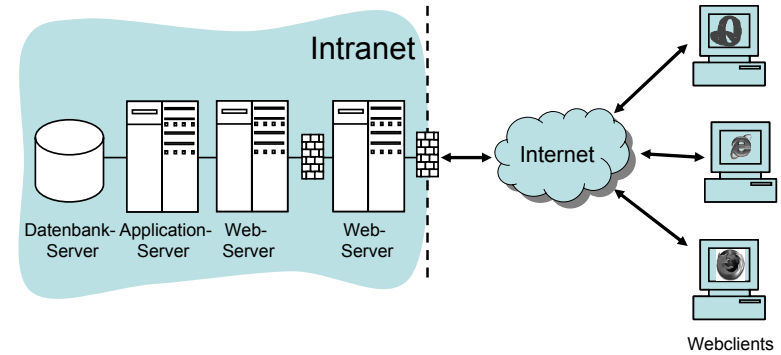
Client/Server-Modell: Aufgabenverteilung

- Wie wird die Anwendung zwischen Server und Client verteilt?
 - Thin Client → Fat Client



Quelle: „Distributed Systems“, Tanenbaum, van Steen, Abb.1-29

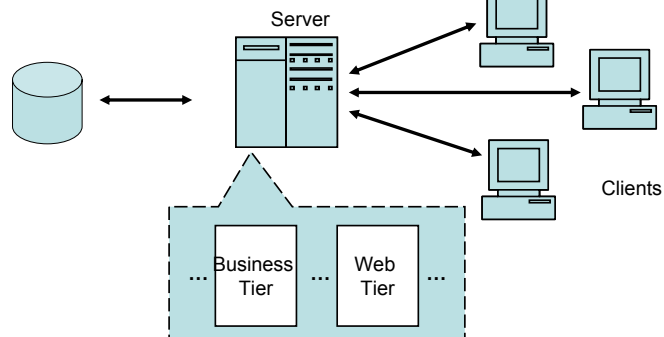
Beispiel: Multi-Tier-Internet-Anwendung



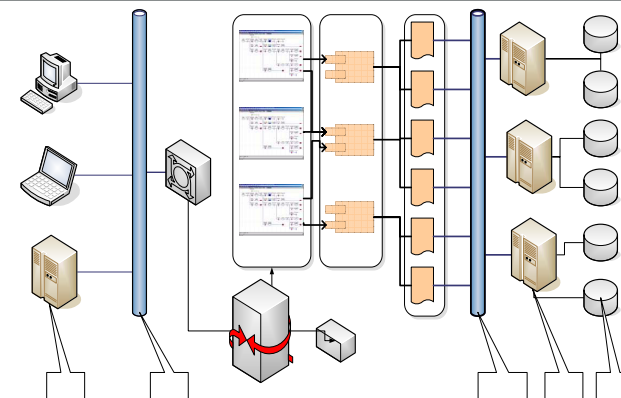
nach „Verteilte Systeme und Anwendungen“, Hammerschall, Abb.13-7

Multitier-Modell

- Einteilung in Komponenten gemäß der Anwendungslogik



Die IBM Referenzarchitektur



- Horizontale Verteilung: ein Server wird physisch in logisch äquivalente Teile unterteilt
- Vertikale Verteilung: logisch unterschiedliche Komponenten auf unterschiedlichen Maschinen



Die IBM Referenzarchitektur

- Client Tier: Vielzahl unterschiedlicher Clients
- Multichannel Integration
 - verschiedene Zugänge: WAP-Handy, Geldautomat, Palmtop, Laptop, 3270
 - Umsetzung der Client-Protokolle und Erzeugung der Client Views
- Application Tier
 - Workflow System implementiert Geschäftsprozesse
 - basierend auf Geschäfts-Komponenten
 - WAS (Websphere Application Server) Process Server implementiert WS-BPEL (Web Services Business Process Execution Language) interpretiert Regeln
 - Geschäftskomponenten integrieren einzelne Dienste zu Business Funktionen (z.B. Authentifizierungs-Dienst benötigt für Auszahlung)
 - Einzeldienste konventionell oder als Web-Services
- EAI (Enterprise Application Integration) Schicht
 - bindet konventionelle Systeme ein
 - verbindet (klassische) Anwendungen (high volume application mediation)
- Host / Datenbanken
 - wie bisher



Remote Procedure Call (RPC)

- Bei bekannten Datentypen automatische Erzeugung von Code
 - Erzeugung von zwei Stummel (*stub*) Prozeduren für Ein- / Auspacken, Versenden und Empfangen
- Ablauf:
 - Client ruft Client-stub auf, der den Namen der fernen Prozedur trägt
 - Client-stub benachrichtigt Server-stub (und blockiert)
 - Server-stub ruft eigentliche Prozedur auf und schickt Ergebnis zurück
 - Client-stub wird deblockiert, Ergebnis wird ausgepackt und der Client-stub terminiert mit fernem Ergebnis als Ergebnis seines Aufrufs
- Daten müssen in einer Standardrepräsentation verschickt werden
 - Client und Server können auf verschiedenen Architekturen laufen

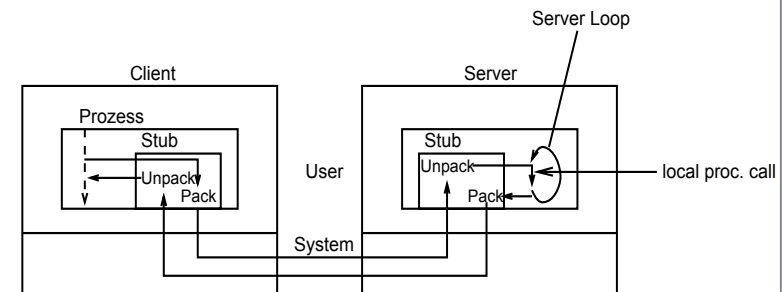


9.3 Remote Procedure Call (RPC)

- Entfernter Funktionsaufruf
 - Client kann Funktion des Servers direkt aufrufen
 - ohne *explizites* Verschicken von Nachrichten auf Programmiererebene
 - Implizit durch Verarbeitung im (automatisch generierten) Stummel (Stub):
 - Beim Aufruf: Prozedurname und Parameter werden in Nachricht verpackt (*marshalling*)
 - Nachricht wird an Server geschickt
 - Beim Server: Nachricht wird ausgepackt (*de-marshalling*) und der entsprechende Aufruf wird ausgeführt
 - Ergebnis wird wieder in Nachricht verpackt und zurückgeschickt
- Zu lösende Probleme
 - Unterschiedliche lokale Darstellungen
 - Definition der Schnittstelle
 - Ausfall von Client bzw. Server



Schema des RPC





Remote Procedure Call (RPC)

- *parameter marshalling* (aufreihen, zusammenstellen)
 - Verpacken von Parametern einschließlich Konversion in Standarddarstellung
 - Code kann automatisch generiert werden
- *unmarshalling* (oder demarshalling)
 - Auspacken einschließlich Konversion in lokale Darstellung
- Server-Loop
 - Code kann ebenfalls automatisch generiert werden
 - Server ruft in Schleife den Server-stub immer wieder auf und bearbeitet so einen Auftrag nach dem andern



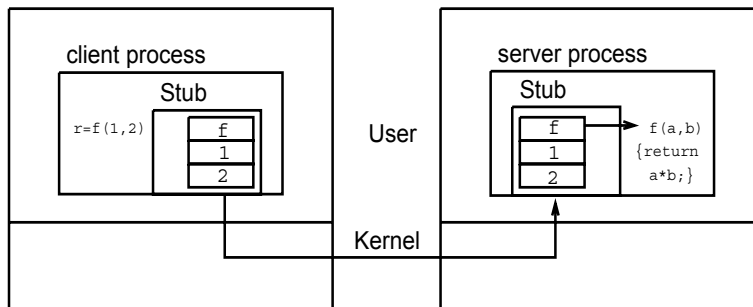
Remote Procedure Call (RPC)

- Grund-Annahme: Heterogenität
 - Client und Server können grundverschieden sein
 - verschiedene Prozessoren
 - verschiedene Betriebssysteme
 - verschiedene Programmiersprachen
 - Keine Codeübertragung
 - keine Übertragung von Objekten als Parameter
 - Daten müssen in einer Standardrepräsentation verschickt werden
 - XDR - eXternal Data Representation
 - Schnittstellendefinition
 - ⇒ Sprach-unabhängige Datendefinitionssprache (IDL)
 - Für jede Programmiersprache erzeugt ein IDL-Compiler Sprach-spezifische Stubs



Schema des Parameter Marshalling

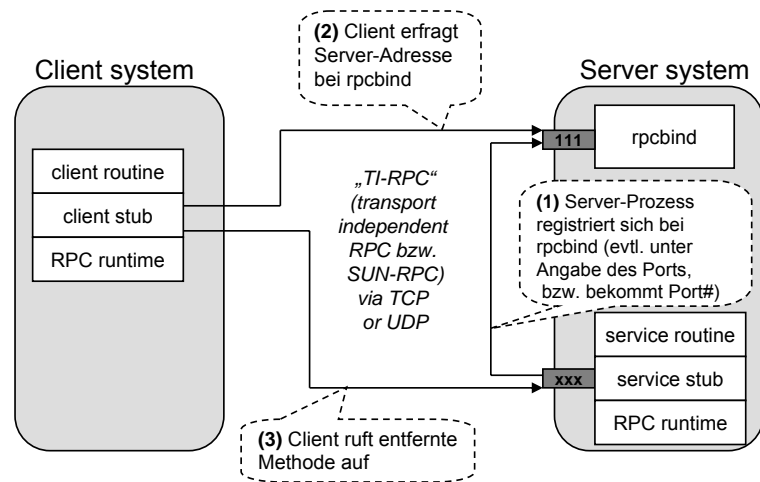
- Beispiel: Multiplikation zweier Zahlen über RPC



RPC – Dynamisches Binden

- Durch die RPC-Methode sind Aufrufer und gerufene Prozedur entkoppelt:
 - nicht in einem gemeinsamen Programm vereinigt
 - können zu verschiedenen Zeiten gestartet werden
- ⇒ **dynamisches Binden (dynamic binding)**
 - Beispiel: statisch: `mobject.add(int)`
dynamisch: `invoke(mobject, "add", int)`
 - Beim statischen binden sind Namen und Transportadresse des betreffenden Dienstes zur Übersetzungszeit bekannt.
 - Beim dynamischen Binden werden die Namen und die Transportadressen der verfügbaren Dienste in einem Namens-Server, *rpcbind* genannt, abgespeichert.
 - Programmbeginn: Client-stub kennt Partner-Adresse noch nicht
 - Bei Aufruf von Client-stub: Anfrage an zentralen *Binder* nach Server, der die Prozedur in der benötigten Version zur Verfügung stellt.
 - Server melden sich beim Binder betriebsbereit unter Angabe von Name, Versions-Nr. und Adresse + evtl. id, falls Name nicht eindeutig.
 - Zur Laufzeit: Binder reicht dem Client die Server-Information weiter, und der Client-stub wendet sich danach direkt an den Server.

Transport-Independent RPC



Fehlerbehandlung in RPC-Systemen

Behandlungsmöglichkeiten sind u.a.:

Zu 1: **No Server.** Stub-Prozedur gibt eine Fehlermeldung zurück.

- Fehlerwert, z.B. -1
 - genügt nicht im allgemeinen, da er auch ein legales Resultat sein könnte.
 - *Exceptions*
 - In C: Simulation durch signal handlers, z.B. mit einem neuen Signal `SIGNOSERVER`
 - Signalverarbeitungsroutine könnte darauf hin mit Fehler umgehen
- Nachteile: Transparenz geht verloren, sowie Signale werden nicht in allen Sprachen unterstützt

Fehlerbehandlung in RPC-Systemen

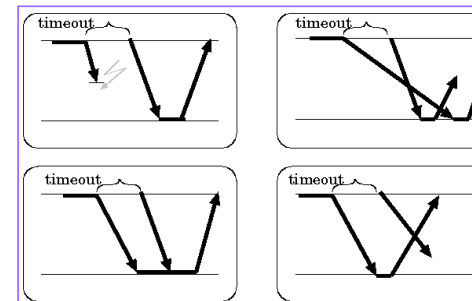
□ Durch die Entkopplung zwischen Klient und Server kann es zu folgenden Fehlern kommen:

1. Der Klient findet den Server nicht.
2. Die Auftragsnachricht Klient/Server geht verloren.
3. Die Antwortnachricht Server/Klient geht verloren.
4. Der Server stürzt nach Auftragserhalt ab.
5. Der Klient stürzt nach Auftragsvergabe ab.

Fehlerbehandlung in RPC-Systemen

Zu 2: **Lost Request.** Sender startet einen Timer und verschickt den Auftrag nach Timeout neu.

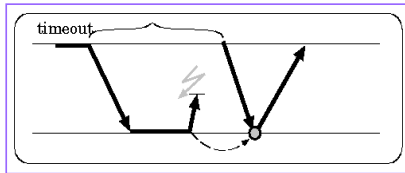
- Kennzeichnung der Aufträge als Original oder Kopie kann verhindern, dass derselbe Auftrag (z.B. Buchung) mehrmals bearbeitet wird.



Fehlerbehandlung in RPC-Systemen

Zu 3: **Lost Reply.**

- Manche Aufträge können problemlos wiederholt werden (Lesen eines Datums), andere nicht (*relative update*; Buchung).
- Buchungsaufträge müssen als Originale und Kopien gekennzeichnet werden.



Fehlerbehandlung in RPC-Systemen

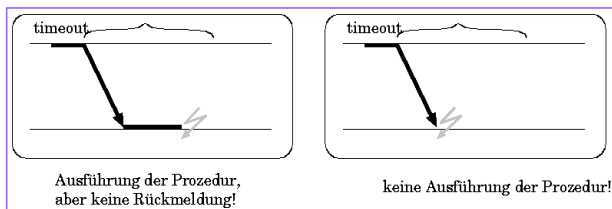
Drei Konzepte der RPC-Abwicklung:

- At least once semantics.** Das RPC-System wiederholt den Auftrag so lange, bis er quittiert wurde.
 - At most once semantics.** Das RPC-System bricht nach Timeout ab mit Fehlermeldung.
 - Keine Garantie.** Das RPC-System gibt irgendwann auf. Der Auftrag kann nicht oder auch mehrmals bearbeitet worden sein.
- Ideal wäre: *exactly once semantics*. Dies ist im Allgemeinen aber nicht zu realisieren.

Fehlerbehandlung in RPC-Systemen

Zu 4: **Server Crashes.**

- Abstürze des Servers vor Auftragseingang fallen in Kategorie 1 (no server)
- Späterer Absturz **vor** Auftragsbearbeitung: Wiederholung des Auftrags (nach reboot)
- Späterer Absturz **nach** Auftragsbearbeitung: Wiederholung nicht unbedingt möglich
- Problem: Klient kennt die Absturzursache nicht.



Fehlerbehandlung in RPC-Systemen

Zu 5: **Client Crashes.**

- Klient-Absturz vor Auftragsvergabe oder nach Auftragsbestätigung hier irrelevant.
- Absturz während der Auftragsverarbeitung führt zu einer Waise (Prozess ohne Eltern; *orphan*).
- Probleme:
 - Verbrauch von CPU-Ressourcen
 - Blockierung anderer Prozesse
 - Fehlerhafte Zuordnung einer Antwort beim Client

Fehlerbehandlung in RPC-Systemen

- Methoden, um Waisen aus dem System zu entfernen:
- 1. Löschung (Extermination)
 - Notieren jeder Auftragsvergabe beim Klienten auf sicherem Medium
 - Nach reboot werden (unquittierte) offene Aufträge storniert (Löschung, Extermination)
 - Problem: großer Aufwand, sowie keine Erfolgsgarantie (z.B. bei fehlender Konnektivität zum Server)
 - Schlussfolgerung: keine relevante Methode
- 2. Reinkarnation (Reincarnation)
 - Zeit wird in Epochen eingeteilt
 - Jeder Client-reboot startet neue Epoche
 - Prozesse der alten Epoche werden auf dem Server beendet
 - Überlebt doch einer (z.B. durch verlorene Epochen-Meldung), so tragen seine Resultate veralteten Epochen-Stempel

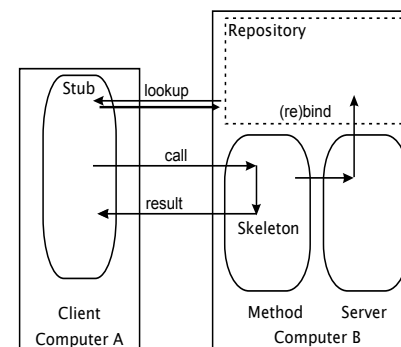
9.4 Java Remote Method Invocation (RMI)

- Entfernte Prozeduren → entfernte Methoden
- RMI = RPC in Objektsystemen
 - Parameter können Objekte sein (Erschwernis)
 - Umgang mit Zeigern und mit Code
 - Objekte in Bytestrom übertragen (serialisieren)
 - verzeigerte Objekte (Listen, Bäume, Graphen): Objektgraphen
 - Alle Information in der Klassendefinition gekapselt (Erleichterung)
- Java RMI ist eine Java-spezifische Realisierung des RPC
 - benutzt Java Objekt-Serialisierungsprotokoll
- Common Object Request Broker Architecture (CORBA)
 - Kapselung von Objekten verschiedener Programmiersprachen

Fehlerbehandlung in RPC-Systemen

- 3. Freundliche Reinkarnation (gentle Reincarnation)
 - Server fragen bei Start einer neuen Epoche nach, ob Eltern der Aufträge noch leben
 - Nur wenn sich Eltern nicht melden, werden die Aufträge beendet
- 4. Verfall (Expiration)
 - Auftragsbearbeitung wird mit Timeouts versehen
 - Nach Timeout müssen sich Eltern melden und dadurch den Timer neu starten
 - ansonsten wird der Prozess beendet
 - Problem: richtigen Wert für Timeout festlegen (RPCs haben stark variierende Anforderungen)

RMI – Schema



- Repository: Verzeichnis der aufrufbaren Objekte
- Server registriert mit bind() bzw. rebind() ein Objekt beim Repository
- Clients fragen mit lookup() beim Repository nach einem Objekt und machen es lokal zugänglich
- Der Client bindet sich zum verteilten Objekt, durch Laden einer Objektschnittstelle (Proxy) in den Adressraum des Clients
- Beim Aufruf im Client wird ein 'Stub'-Objekt aktiviert, das die Eingabeparameter an 'Skeleton'-Objekt übergibt, welches die gewünschte Methode aufruft.
- Returnwert wird nach Terminierung der Methode an das Stub-Objekt zurückgeschickt, das den Wert an das aufrufende Objekt übergibt.
- Der Einsatz von Stub und Skeleton erfolgt transparent



Aufgabe von verteilten RPC-Systemen

- Verteilte RPC Systeme zielen auf die Lösung der Kommunikationsprobleme, die auf der **Heterogenität** der beteiligten Systeme beruhen:
 - Unterschiedliche Programmiersprachen
 - Unterschiedliche Rechner
 - Unterschiedliche Betriebssysteme
 - Unterschiedliche Datenrepräsentation
 - Unterschiedlicher Maschineninstruktionssatz



Probleme bei RPC

- Einschränkung hinsichtlich der übertragbaren Daten
 - Nur einfache Datentypen, die in allen unterstützten Programmiersprachen repräsentierbar sind, und
 - Referenzen auf entfernte Objekte, sowie
 - Komplexe Datentypen, die sich aus den zuvor genannten zusammensetzen
- Komplexität bei Typanpassung verbleibt beim Programmierer
- Life-Cycle-Management wird dem Programmierer auferlegt
 - ⇒ Gefahr von Fehlern
- Sender und Empfänger müssen die übertragbaren Datentypen zum Zeitpunkt der Kompilierung bereits kennen.
 - ⇒ Keine Unterstützung von Polymorphie
(Polymorphie: Bezeichner kann je nach Kontext einen unterschiedlichen Datentypen annehmen)



Lösungsansatz bei RPC-Systemen

- Stub und Skeleton bilden Aufrufchnittstelle und erledigen den eigentlichen Datenaustausch
- Marshaling, Unmarshaling zur (De-)Serialisierung
- Gemeinsame IDL (Interface Description Language) ermöglicht Kommunikation auch zwischen unterschiedlichen Programmiersprachen
 - Für jede beteiligte Sprache ein IDL-Compiler
- Unterstützung entfernter Referenzen für Objekte



Lösung von Java RMI

- Heterogenität stellt kein Problem dar, da **Homogenität** durch die Java JVM gewährleistet ist.
 - Externe IDL nicht erforderlich (stattdessen Java Interfaces)
 - Keine Einschränkung hinsichtlich der übertragbaren Datenstrukturen.
 - Java Objekt-Serialisierung ermöglicht exakte Typenprüfung
 - Dynamic Code Loading: Der den Kommunikationsfluss steuernde Programmcode kann auch erst während der Programmausführung zur Verfügung gestellt werden.
 - Java Objekt-Serialisierung und Dynamic Code Loading ermöglichen den Einsatz von Polymorphie und aller darauf aufbauenden Programmiermuster
 - Network Garbage Collection wird unterstützt



Parameterübergabe und Rückgabewerte

- Java allgemein:
 - Parameter (und Rückgabewerte) werden jeweils kopiert: call by value.

Hinweis: Objektvariablen bilden dabei keine Ausnahme. Da es sich bei ihnen jedoch um Referenzen handelt, entspricht ihre call by value-Übergabe der Semantik von call by reference. Änderungen durch die aufgerufene Methode entfalten also Wirkung.
- Java RMI:
 - Lokale Datentypen und lokale Objekte werden kopiert: call by value
 - Entfernte Objekte durch Kopie der Objektreferenz (des entsprechenden proxies/stubs): call by reference

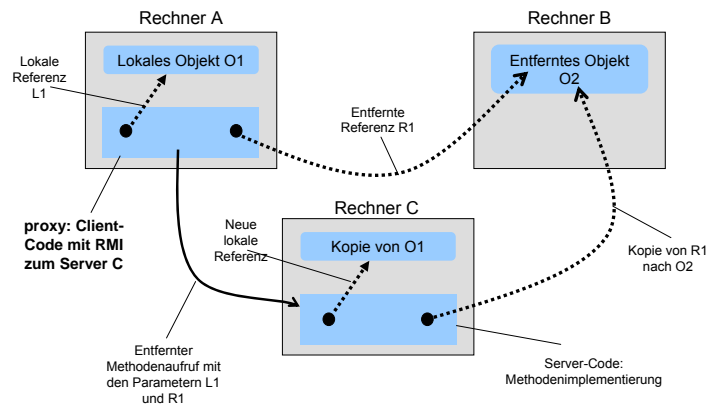


Gliederung - Kapitel 9: Verteilte Systeme

- Kapitel 9 - Teil 1
 - 9.1 Grundlagen
 - 9.2 Middleware
 - 9.3 RPC
 - 9.4 RMI
- Kapitel 9 - Teil 2
 - 9.5 Service Oriented Architectures
 - 9.6 Corba
 - 9.7 Web-Anwendungen
 - 9.8 HTML und XML
 - 9.9 Web Services



Veranschaulichung der Parameterübergabe



Quelle: Tanenbaum, Distributed Systems, 2. Aufl., Abb. 10-8

- Lokale Objekte werden als Kopie übergeben
- Entfernte Objekte werden als Referenz übergeben (und bleiben entfernt)



Inhalte von Kapitel 9, Teil 2

- Service-Orientierte Architekturen
- Corba
- Web-Technologien
 - Java Server Pages
 - Java Servlets
- Sprache XML
 - XML Tags
 - Name Spaces
 - XML-Schemata
 - Validierung von XML-Dokumenten
 - Werkzeugunterstützung für XML
 - Transformation in andere XML-Formate, oder andere Sprachen
- Web Services
 - Schichtenarchitektur
 - Simple Object Access Protocol - SOAP: Mechanismus zur Repräsentation/zum Austausch von Daten
 - Web Services Description Language - WSDL
 - Simple API for XML Parsing - SAX
 - Universal Description and Integration UDDI



9.5 Definition Service Oriented Architectures

- SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird.
[Organization for the Advancement of Structured Information Standards (OASIS) , 2006] c.f. oasis-open.org
- Ein Dienst in einer Service-Orientierten Architektur hat (idealerweise) folgende Eigenschaften
 - Dienst ist in sich abgeschlossen und kann eigenständig genutzt werden.
 - Dienst ist über ein Netzwerk verfügbar.
 - Dienst hat eine veröffentlichte Schnittstelle. Für die Nutzung reicht es, die Schnittstelle zu kennen. Kenntnisse über die Details der Implementierung sind hingegen nicht erforderlich.
 - Dienst ist plattformunabhängig, d.h. Anbieter und Nutzer eines Dienstes können in unterschiedlichen Programmiersprachen auf verschiedenen Plattformen realisiert sein.
 - Dienst ist in einem Verzeichnis registriert.
 - Dienst ist dynamisch gebunden, d.h. bei der Erstellung einer Anwendung, die einen Dienst nutzt, muss der Dienst nicht vorhanden sein. Er wird erst bei der Ausführung lokalisiert und eingebunden.



CORBA

- Merkmale von CORBA
 - Objektorientierung
 - Sprachunabhängigkeit
 - Kommunikationsmechanismen für verteilte Systeme
 - Allgemeines Konzept von kommunizierenden Objekten
 - Plattformunabhängigkeit
 - Herstellerunabhängigkeit
 - Anbindung anderer Komponentensysteme
- CORBA geht daher nach folgendem Prinzip vor
 - Die Schnittstellen werden von der Implementierung streng getrennt.
- Durch separate Definition der Schnittstellen kann die Kommunikation unabhängig von der jeweiligen Implementierung betrachtet werden.
 - IDL (Interface Definition Language): Neutrale Spezifikationsprache beschreibt die Schnittstellen der beteiligten Objekte
 - ORB (Object Request Broker): Kommunikation zwischen den Objekten



9.6 CORBA

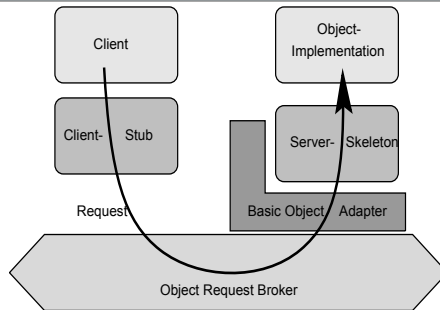
- Common Object Request Broker Architecture
- Allgemeiner Architektur-Standard für Entwicklung von Client/Server Anwendungen
- Verschiedene konkrete Implementierungen: ORBs.
- Definiert von der Object Management Group (OMG)
 - Zusammenschluss von über 750 Unternehmen, Software-Entwicklern und Anwendern.
 - 1989 gegründet.
- Allgemeine Kommunikationsinfrastruktur zwischen verteilten Objekten, wobei für den Entwickler die Kommunikation weitestgehend transparent ist.



Merkmale von CORBA

- Kommunikationsmechanismen
 - Kommunikation zwischen den Objekten ist völlig transparent
 - Der ORB verdeckt
 - Auffinden des Zielobjekts
 - Übertragung der Daten
 - Etwaige Konvertierungen zwischen Datenformaten.
 - Kommunikation selbst ist plattform- und sprachunabhängig.
 - Seit CORBA 2.0 arbeiten auch Object Request Broker verschiedener Hersteller zusammen (durch das Internet Inter-ORB Protocol - IIOP).
- Allgemeines Konzept von kommunizierenden Objekten
 - Die bisherige starre Einteilung in Client und Server entfällt, in CORBA existieren gleichberechtigte Objekte.
 - Objekt A kann Server für ein Objekt B sein, während B gleichzeitig Serverfunktionalität für ein Objekt C anbietet.
 - Festlegung: Server ist das Objekt, das ein IDL-Interface implementiert.

Grundprinzip von CORBA



- Mit Hilfe der IDL wird ein Interface definiert.
- IDL-Compiler erzeugt aus dieser Schnittstellenbeschreibung Source Code in der gewünschten Sprache für den Client-**Stub** und für den Server-**Skeleton**.
- Server wird implementiert und ist über das Skeleton für andere Objekte zugänglich. Über den *Basic Object Adapter (BOA)* meldet sich der Server beim ORB an und ist jetzt bereit, Aufrufe anderer Objekte zu empfangen.
- Der Client kann nun über den Stub auf den Server zugreifen. Dieser Zugriff läuft über den ORB.

Object Request Broker – Grundfunktionalität

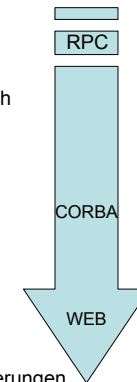
- Statische und dynamische Aufrufe von Methoden
 - Statischer Aufruf: sichere Typprüfung
 - Dynamische Aufrufe: höhere Flexibilität
- Kommunikation auf Hochsprachen-Niveau
 - Es ist für Entwickler nicht notwendig, Parameter „einpacken“ (marshaling) oder Befehle in einer Kurzform zu übertragen
- Selbstbeschreibendes System
- Jeder ORB besitzt eine Datenbank, das sog. Interface Repository
 - Enthält Meta-Informationen über die bekannten Interfaces
 - Daten werden automatisch verwaltet und gepflegt, i.a. durch den IDL-Compiler
- Transparenz zwischen lokalen und entfernten Aufrufen
- Umwandlungen zwischen Datenformaten (z.B. little endian/big endian, usw.) werden vom ORB durchgeführt
- Kompatibilität bzgl. Art und Größe der Typen durch IDL gewährleistet

Grundprinzip von CORBA

- Ablauf entfernter Methodenaufruf:
 - Object Request Broker fängt Aufruf ab und lokalisiert das Zielobjekt
 - Übergabeparameter werden verpackt und an den Server geschickt.
 - Dort werden die Parameter wieder entpackt und die Methode auf dem Server ausgeführt.
 - Resultat wird verpackt und an den Aufrufer zurückgesendet.
 - Gesamter Vorgang wird vom ORB verdeckt.
- Client benötigt für entfernten Methodenaufruf eine *Referenz* auf das entfernte Objekt: *Object-Reference*
 - eindeutige ID eines bestimmten Objekts
- Angesprochen wird die Server-Komponente über die automatisch generierte "Stub-Klasse" ⇒ hohe Typsicherheit

Gründe für den Niedergang von CORBA

- Technische Gründe
 - Komplexität von CORBA
 - Unterstützung von C++ fehlerträchtig
 - Sicherheitsmängel
 - unverschlüsselter Datenaustausch
 - pro Dienst ein offener Port in Firewall erforderlich
 - Viel Redundanz, keine Kompression
 - Keine Thread-Unterstützung
 - Fehlende Unterstützung von C#, .NET, DCOM
 - Fehlende Integration des Web
 - Aufkommen von XML, SOAP
 - ⇒ E-business Lösungen generell ohne CORBA
- Soziale Gründe / Verfahrensfehler
 - Zu viele Köche im Standardisierungsprozess
 - Z.T. Fehlende Referenzimplementierungen
 - Ungetestete Innovationen in Standards
 - Hohe Lizenzgebühren für kommerzielle Implementierungen
 - open-source Implementierungen zu spät
 - Mangel an erfahrenen Entwicklern



Quelle: Michi Henning, ACM Queue.
http://www.acm.org/acmqueue/digital/Queuevol5no4_May2007.pdf



9.7 Web Services: Grundidee und Definitionen

- HTML (HyperText Markup Language) als Beschreibungssprache zur Darstellung von Dokumenten im World Wide Web ist schlecht geeignet zur Maschine-Maschine-Kommunikation
- Grundidee für Web Services: Einsatz Web-basierter Service-Orientierter Architektur
 - Darstellung der Informationen mit einer dafür besser geeigneten Sprache ⇒ XML
 - Verwendung von Elementen der Web-Architektur, u.a. Beibehaltung von HTTP zum Transport dieser Daten



9.8 HTML und XML: Markup

- Markup: Informationen, die einem Dokument beigefügt werden, selbst aber nicht unmittelbar dargestellt werden.
- Bsp.: HTML-Markup:
 - <h2>Markup zur Textformatierung</h2>
 - Unter <i>Markup</i> versteht man Informationen, die einem Textdokument beigefügt sind.



Web Services - Gründe für das Interesse

Vorzüge

- Web Services basieren auf offenen Protokollen bzw. Spezifikationen
 - Beschreibung der Schnittstelle: WSDL (Web Service Description Language)
 - Kommunikation: SOAP (Simple Object Access Protocol)
 - Finden von Diensten: UDDI (Universal Description Discovery and Integration)
 - Spezifikation über XML-Grammatiken
- Heterogene Plattformen (J2EE, .Net etc.) werden unterstützt
- Grundlage zur Realisierung der Service-orientierten Architektur mittels WSDL (Web Services Description Language)
- Web Services werden von "großen Organisationen" (IBM, Sun Microsystems, SAP, Microsoft, ...) unterstützt

Mögliche Nachteile

- Leistungsfähigkeit von SOAP / XML ggf. geringer anderer Leistungsfähigkeit anderer Middleware-Lösungen



HTML – Hypertext Markup Language

- HTML definiert primär Layout eines Web-Dokuments
 - nur sekundär auch Struktur
- Auszeichnungssprache
 - paarweise öffnende und schließende Tags
 - Hierarchische Gliederung
- Tags
 - zur Textformatierung
 - <i>kursiv</i>
 - zur Spezifikation von Hypertext-Links
 - W3 Konsortium
 - zum Einbinden von Multimediaobjekten und Applets
 -
 - für Formulare

Web-Anwendungen

- Dynamische Web-Seiten-Erzeugung
 - CGI-Skripte bzw. -Programme
 - Active Server Pages (heute ASP.Net)
 - PHP
 - Java Servlets
 - Java Server Pages

Java Server Pages (JSP)

- JSP: Eine HTML-Seite mit eingebettetem Java-Code, der auf Server ausgeführt wird und Teile der Seite dynamisch generiert.
- Lösung für folgendes Problem: Es ist umständlich, die statischen Teile der HTML-Seite durch das Servlet in print-statements zu generieren.
- Idee: Schreibe statische Teile als HTML in Dokument und bette dynamische Teile durch Programmcode darin ein (spezieller HTML-Kommentar).
- JSP wird von Webserver in Servlet übersetzt

```
<html>
<body>

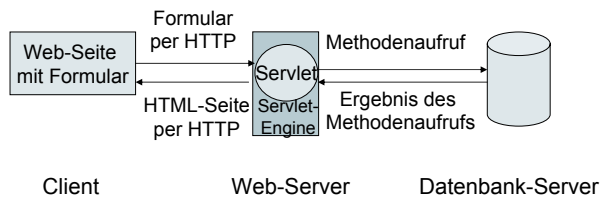
<%-- Kommentar --%>
<% //Java-Code %>

</body>
</html>
```

Skriptlet <% ... %>
Kommentar <% --...--%>
Direktive <% @ ... %>
Ausdruck <% = ... %>
Deklaration <%! ...%>

Servlets

- Problem: HTML Dokumente als Files sind statisch
- Idee: Generiere HTML-Dokumente dynamisch durch Programm
- Servlet: Ein Java-Programm, das auf dem Server als Reaktion auf einen http-request (GET oder POST) gestartet wird
- HTML-Seite wird vom Servlet generiert.
- Beispiel: Servlet in 3-tier-Architektur



SGML – Standardized General Markup Language

- In den 80er Jahren bei IBM entwickelt (Goldfarb)
- ISO Standard 1986
- Tags zur Annotation (mark-up) eines Textes
 - zur Textformatierung
 - <i>kursiv</i>
- Ziel: repräsentiere Industrie-Dokumentationen
 - elektronisch
 - unabhängig von konkreten Text-Satzsystemen
 - bilde auf jeweiliges Medium ab (z.B. Web oder Druck)
 - Industriequalität (Novell-Handbücher: 150.000 Seiten)
- Problem: sehr mächtig, sehr komplex

SGML, HTML, XML

- Bewertung von SGML: Idee prima, aber SGML zu komplex
- HTML: Formatiere Dokumente für das Web
 - Vordefinierte Menge von Tags, Semantik fixiert
 - Werden von Browsern interpretiert
 - bilden *Document Type*, der mittels SGML definiert ist
- XML: definiere Dokumentstruktur allgemein
 - Minimale Sprache, kleines subset von SGML („SGML-“)“)
 - Erweiterbar
 - Abbildung Struktur ⇒ layout separat über style sheets
⇒ auch für Datenserialisierung nützlich

XML – eXtensible Markup Language

- Beschreibung strukturierter Daten (kein Layout)
- Maschinen- und Menschen-lesbar (ASCII-Text)
- wohlgeformte XML-Dokumente
 - paarweise öffnende und schließende Tags
 - HTML erlaubt auch Ausnahmen: `
`
 - keine Überlappung der Tags erlaubt
 - alle Tags kleingeschrieben
 - Attributwerte in doppelten Anführungszeichen
- Inline-Schreibweise mit Attributen
`<autor nachname="Stevens" vorname="Richard"/>`
- gemischte Schreibweise
`<autor geschlecht="männlich">
Stevens, W. Richard
</autor>`

XML – eXtensible Markup Language

- Offener Standard (www.w3.org)
- Strukturdefinition (Grammatik einer Dokumentfamilie)
 - Document Type Definition (DTD)
 - XML-Schema
- Plattform-unabhängig (ASCII)
- Allgemein einsetzbar
- Leichte maschinelle Zugänglichkeit der Information
 - Textbasiert
 - Strukturiert
- Werkzeuge
 - Editoren, Browser, ...
 - Parser
 - APIs
 - Datenbank-Schnittstellen

XML – Beispiel

```
Kopf <?xml version="1.0"?>
Rumpf <literaturverzeichnis>
      <buch>
        <autor>Stevens, W. Richard</autor>
        <titel>UNIX Network Programming</titel>
        <verlag>Prentice Hall</verlag>
        <erscheinungsjahr>1990</erscheinungsjahr>
        <isbn>0-13-949876-1</isbn>
        <stichwort>Netzwerk</stichwort>
        <stichwort>Netzwerk-Programmierung</stichwort>
      </buch>
      <buch>
        ....
      </buch>
    </literaturverzeichnis>
```




XML – Syntax: Namensräume

- XML-Namensräume (XML Name Spaces)
 - Werden benutzt, um in einem Dokument mehrere XML-Sprachen zu mischen
 - Werden durch URIs dargestellt
 - Die entsprechende Adresse muss nicht existieren
 - Wenn eine URL als Namensraum verwendet wird, wird unter dieser Adresse ggf. zusätzliche Informationen zu der XML-Sprache angeboten, z. B. eine Dokumenttypdefinition (DTD) oder ein XML-Schema.
- Ziel: Vermeidung von Mehrdeutigkeiten bei Tags
- Definition mittels `xmlns`-Attribut oder dem `xmlns:`-Präfix
- Wert des Attributs ist Name des Namespaces
- Beispiel:


```
<?xml version="1.0"?>
<da:literaturverzeichnis
  xmlns:da="http://www.in.tum.de/diplArb">
  <da:buch> ... </da:buch>
</da:literaturverzeichnis>
```



XML-Schema

- Sprache zum Definieren von Strukturen für XML-Dokumente.
- Im Gegensatz zu XML-DTDs wird die Struktur in Form eines XML-Dokuments beschrieben.
- Zahlreiche Datentypen werden unterstützt.
- Basis-Datentypen
 - `string`, `date`, `time`, `integer`, `double`, `boolean`
- komplexere Strukturen
 - Einschränkung des Wertebereichs
 - Listen
 - Vereinigung und Kombination verschiedener Typen
 - Vererbung



XML Document Type Definition (DTD)

- Dokumenttypdefinition (Document Type Definition, DTD, auch Schema-Definition oder DOCTYPE)
 - Sprache zur Beschreibung von Dokumenttypen, d.h. zur Beschreibung der Struktur von Dokumenten (Beachte: Für XML-Dokumente existieren auch andere Schema-Sprachen, z.B. XML Schema)
 - Formale Grammatik (DTD-Syntax selber ist kein XML)
 - Spezifiziert Elemente, Attribute, Entitäten
 - Spezifikation von Dokumenttypen erlaubt Validierung von Dokumenten
 - XML-Editor der DTD (Grammatik) kennt kann nur gültige Eingaben zulassen
 - Einbindung im selben Dokument oder ausgelagert in externes Dokument
 - Beispiel:


```
<!ELEMENT publication (proceedings | article)>
<!ELEMENT proceedings ((author)*, title, conference)>
...
<!ELEMENT conference (#PCDATA)>
<!ATTLIST conference
  year CDATA #required>
```



XML-Schema – Beispiel

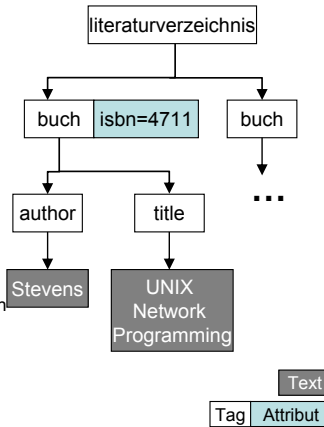
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="literaturverzeichnis">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="buch" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="autor" type="xs:string"
                maxOccurs="unbounded"/>
              <xs:element name="titel" type="xs:string"/>
              <xs:element name="untertitel" type="xs:string"
                minOccurs="0"/>
              <xs:element name="verlag" type="xs:string"/>
              <xs:element name="erscheinungsjahr" type="xs:string"/>
              <xs:element name="isbn" type="xs:string"/>
              <xs:element name="stichwort" type="xs:string"
                maxOccurs="unbounded"/>
              <xs:element name="abstract" type="xs:string"
                minOccurs="0"/>
              <xs:element name="kommentar" type="xs:string"
                minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Vom W3C im XML-Schema
<http://www.w3.org/2001/XMLSchema> vorgegeben:
 - Basistypen `element`, `complexType`, `sequence`, ...
 - Datentypen `string`, `integer`, ...



Werkzeugunterstützung für XML

- APIs zum Parsen von XML-Daten
- SAX
 - *Simple API for XML*
 - Ereignisorientierter Ansatz
 - Dokument wird komplett durchlaufen
 - Beginn / Ende jedes Tags wird über Callback-Methoden mitgeteilt
- DOM
 - *Document Object Model*
 - Erlaubt, gezielt auf einzelne Teile des Dokuments zuzugreifen
 - Baumorientierter Ansatz
 - Applikation bekommt Baum nach Verarbeitung des Dokuments übergeben
 - höherer Speicherbedarf als SAX
- Beide Ansätze in J2SDK enthalten
 - Zusammengefasst in *Java API for XML Processing (JAXP)*



XML Zusammenfassung

- Umfangreicher Tool Support
- Generalized Markup
 - Trennung von Struktur und Darstellung
 - XML Parser ⇒ Information Access
 - CSS, XSL, XSLT ⇒ Darstellung (Layout, Rendering)
 - Sichten ⇒ Flexibilität und Konsistenz
- Document Type Definition (DTD, Schema)
 - Klassenbildung + Validierung
- Persistenz (inkl. XML Datenbanken)
- Erweiterte Link-Fähigkeit
- Multi-Medial, International



XML Standard Familie

Von W3C standardisiert (www.w3.org)

- XML (1998)
- CSS (1998): Cascading Style Sheets (layout: Stylesheet-Sprache für strukturierte Dokumente)
- Namespaces (1999)
- XSLT 1.0 (1999) XSL Transformations (Programmiersprache zur Transformation von XML-Dokumenten)
- XPath 1.0 (1999) Zugriff auf Teile eines Dokuments
- XHTML 1.0 (2000), (Extensible HTML). "A Reformulation of HTML 4 in XML 1.0"
- DOM Level 2, Document Object Model, Core Specification (2000)
- XML Schema (2001) Grammatik-Sprache für XML-Dokumentfamilien
- XLink 1.0 (2001): XML Linking Language (Spez. für Hyperlinks)
- XML Base (2001) (Spezifikation von Datenbank URIs für Dokument-Teile)
- XSL 1.0 (2001) Extensible Stylesheet Language (layout)
- XPointer (2002) : XML Pointer Language (Spezifikation von Pfaden in URIs)
- XQuery 1.0 (2002) XML Query Language (Abfragesprache für Datenbanken)
- XInclude (2002) XML Inclusions



9.9 Definition von Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

*It has an interface described in a machine-processable format (specifically **WSDL**).*

*Other systems interact with the Web service in a manner prescribed by its description using **SOAP** messages, typically conveyed using HTTP with an **XML** serialization in conjunction with other Web-related standards.*

David Booth et al.: *Web Service Architecture*
W3C Working Group Note 11 February 2004
<http://www.w3.org/TR/ws-arch/>



Definition von Web Services

A web service is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP. Using a web service could be as simple as logging into a site or as complex as facilitating a multi-organization business negotiation.

Chappell / Jewell:
„Java Web Services“
O'Reilly, 2002



Web Services

- Vision: Integration verschiedener Business-Komponenten
 - über Abteilungs- und Unternehmensgrenzen hinweg
 - mit Web-basierten Technologien (XML / HTTP)
- Integration durch Schaffung von Standards
 - vor allem: XML
 - im Unterschied zu z.B. CORBA (z.B. XML statt IIOP)
- Merkmale von Web Services
 - XML-basiert ⇒ plattformunabhängig
 - lose Kopplung
 - grobgranular ⇒ mehrere Methoden bilden einen Dienst
 - Unterstützung von entfernten Methodenaufrufen
 - Unterstützung von Dokumentenaustausch



Web Services

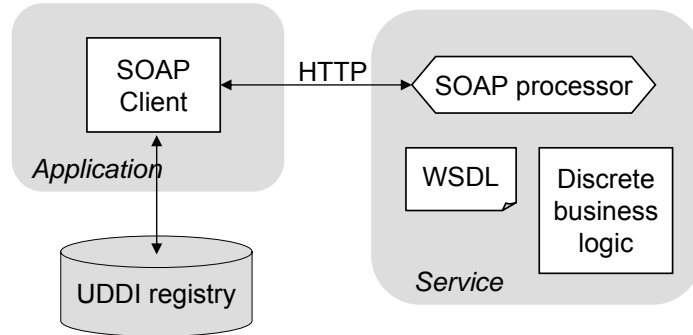
- Komponentenmodell unter Verwendung von Web-Technologien
 - zentral: XML (Serialisierung und Schnittstellenbeschreibung)
- Motivation:
 - Löst ähnliche Probleme wie CORBA
 - Aber offener / allgemeiner als CORBA
- Bisher: große monolithische Informationssysteme
 - Client / Server / Datenbank zu eng verwoben
 - Schlecht dokumentierte Schnittstellen, proprietäre Formate
 - CORBA ORB's nicht überall verfügbar, beschränkt inter-operabel
 - Re-compilation bei kleinsten Änderungen der CORBA IDL



Web Services – Schlüsseltechnologien

- **XML**
 - Universelle Beschreibungssprache
 - Selbst-dokumentierend
 - Robust gegen Änderungen: Empfänger überliest irrelevante Einträge
- **WSDL** (Web Service Description Language)
 - Interface Beschreibung von Diensten (analog CORBA IDL)
- **SOAP** (Simple Object Access Protocol)
 - Kommunikation zwischen Diensten („XML-RPC“)
 - Transportiert XML-serialisierte Werte und Methoden-Aufrufe
- **UDDI** (Universal Description, Discovery and Integration)
 - Suchen von Diensten
 - Weltweiter Verzeichnisdienst für Web Services

Web Services – einfaches Szenario

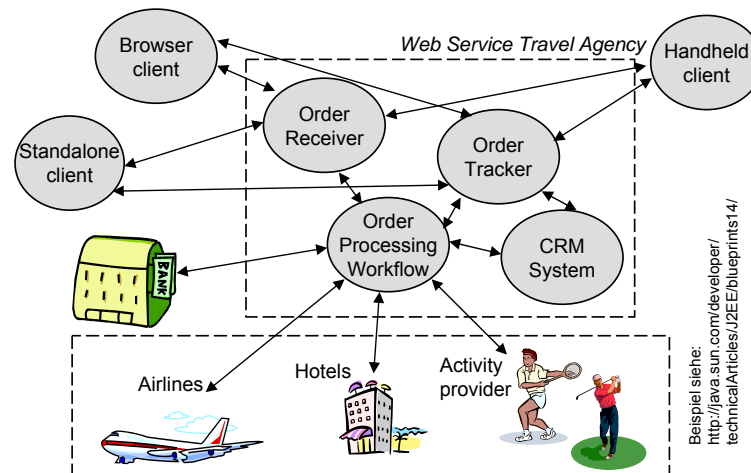


Chappell / Jewel: „Java Web Services“, O'Reilly, 2002, Abb.1-1

XML-RPC

- Idee: Entfernter Methodenaufruf ohne neue Technologien
 - Serialisierung in XML-Dokument
 - Methodenname
 - Parameter
 - Verschicken über HTTP
- Verfügbare Datentypen
 - <int>
 - <double>
 - <string>
 - <boolean>
 - <base64>
 - Bytefolge
 - <dateTime.iso8601>
 - Beispiel: <dateTime.iso8601>20060125T11:15:12</dateTime.iso8601>

Web Services – Szenario



Beispiel siehe:
<http://java.sun.com/developer/technicalArticles/2EE/blueprints14/>

XML-RPC – Prozeduraufruf

```

HOST /xml-rpc.app HTTP/1.1
Content-type: text/xml
Content-length: 255

<?xml version="1.0"?>
<methodCall>
  <methodName>calcMaximum</methodName>
  <params>
    <param>
      <value><int>47</int></value>
    </param>
    <param>
      <value><int>23</int></value>
    </param>
  </params>
</methodCall>
  
```

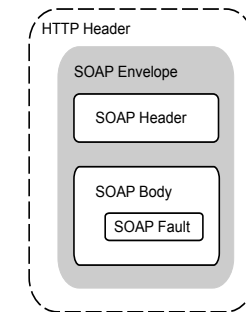
XML-RPC – Prozedurantwort

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 158

<?xml version="1.0" ?>
<methodResponse>
  <params>
    <param>
      <value><int>47</int></value>
    </param>
  </params>
</methodResponse>
```

SOAP

- *Simple Object Access Protocol*
- XML-basiertes Nachrichtenprotokoll
- Arbeitet auf bestehenden Transportprotokollen (HTTP, SMTP)
- Aufbau einer SOAP Nachricht
 - Envelope
 - Header
 - optional
 - für Metainformationen
Body
- kann vollständig im Dokument-Teil eines HTTP-Requests übertragen werden.



XML-RPC – Bewertung

- Vorteil
 - Sehr einfach und schlank
- Nachteile
 - ungenaue Codierung der Datentypen
 - z.B. Probleme mit Datumstyp: keine Zeitzone
 - Aufwändige Codierung binärer Daten (base64)
- Fehlende Metainformation zu den Methoden
 - könnten nur im Request-/Response-Header von HTTP beigefügt werden (⇒ nicht self-contained)
- Weiterentwicklung zu SOAP

SOAP – Nachrichtenformat

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://www.w3.org/2003/05/soap-envelope/">
  <SOAP-ENV:Header>
    <!-- Header-Information -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <!-- Body-Information -->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



WSDL – Web Services Description Language

- Interface Definition Language for Web Services
- Basiert auf XML Schema
- Aufbau:
 - Beschreibung der Datentypen, die in Nachrichten vorkommen
- Abstract Operations
 - Die Operationen, die durch die Nachrichten ausgelöst werden
- Service Bindings
 - Abbildung der Nachrichten auf Transportprotokolle



UDDI

- Anforderungen
 - Veröffentlichen von Web Services
 - Finden von Web Services
- Anbieter von UDDI-Repositories: IBM, SAP und Microsoft
- Benutzung
 - Web-Interface
 - API z.B. JAXR (Java API for XML Registries)
- Beschreibung der Web Services mittels XML-Datenstrukturen
 - Business Entity: Kontakt, Beschreibung, Beziehung zu anderen Geschäftseinheiten, ...
 - Service: Web Service oder andere Dienstleistungen
 - Binding: Technische Beschreibung, Access point URL, Verweis auf Spezifikation
 - ...
- am wenigsten genutzter Standard



WSDL – Web Services Description Language

- Data Type Definitions
 - types – verwendete Datentypen als XML Schema
 - message – Definition der Nachrichten mit Parametern
- Abstract Operations
 - operation – Definition, welchem Dienst (Prozedur, Queue etc) die Nachricht zur Behandlung übergeben werden soll
 - portType – Abstrakter (Service-) Port als Menge von Operationen
 - binding – Abbildung eines Port Type auf einen konkreten Transportmechanismus (Protokoll)
- Service Bindings
 - port – Netzwerkadresse für ein Binding
 - service – Menge von Port Types, die gesamthaft einen logischen Dienst darstellen.

definitions

types

message

portType

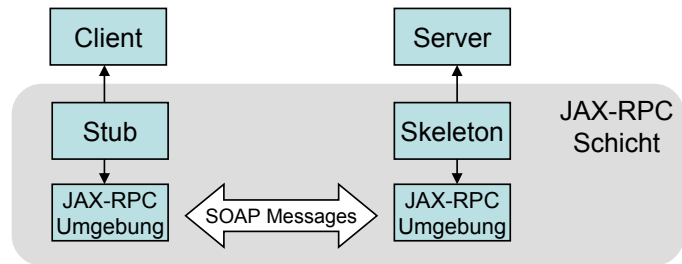
binding

service



Java Web Service Developer Pack

- Im JWSDP u.a. enthalten:
 - Java Architecture for XML Binding (JAXB)
 - Generierung von Java Klassen aus DTDs
 - Java API for XML Processing (JAXP)
 - SAX (Simple API for XML Parsing)
 - ereignisgesteuerter Parser
 - DOM (Document Object Model)
 - Objekt Repräsentation in Form eines Baums
 - Java API for XML-based RPC (JAX-RPC)
 - Methodenaufrufe und Rückgabewerte als SOAP Nachricht
 - Erzeugung von Stubs und Ties (Skeletons) aus WSDL Beschreibung
 - SOAP with Attachments API for Java (SAAJ)
 - Erstellen und Verschicken von SOAP Nachrichten mit Anhängen
 - Asynchroner Nachrichtenaustausch



Thomas Stark: „J2EE“, Abb.12-2
Addison-Wesley, 2005