TU Braunschweig
Institut für Betriebssysteme
und Rechnerverbund

Verteilte Systeme

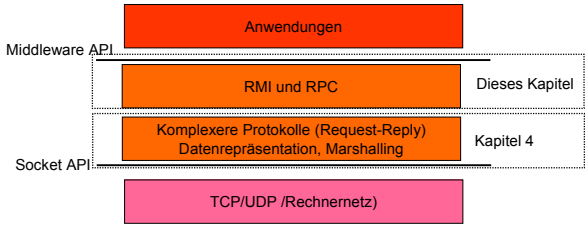
Prof. Dr. Stefan Fischer

Kapitel 5: Middleware

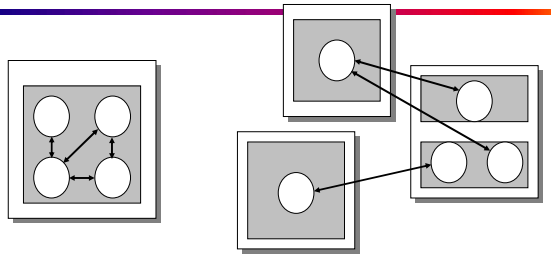
Überblick

- Verteilte Objektsysteme und entfernter Methodenaufruf
- Middleware-Ansätze
  - CORBA
  - Java Remote Method Invocation (RMI)
  - Web Services auf der Basis von XML

Schichten des Kommunikationssystems



Verteilte Objektsysteme



Lokales Objektsystem **Verteiltes Objektsystem**

Rechner **Prozess** **Objekt**

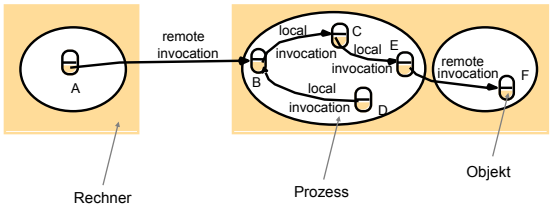
Das Objekt-Modell

- System = Sammlung miteinander interagierender Objekte, von denen jedes aus einer Menge von Daten und einer Menge von Methoden besteht
- Wichtige Begriffe:
  - *Objektreferenz*: die „Adresse“ eines Objekt
  - *Schnittstellen*: Definition der Zugangspunkte eines Objekts; definiert durch die Signatur der Methoden
  - *Aktionen*: initiiert durch ein Objekt, das eine Methode eines anderen Objekts aufruft; resultiert in der Zustandsänderung von Objekten
  - *Exceptions*: eine Möglichkeit, Fehler in einem Programm auf saubere Art und Weise zu behandeln
  - *Garbage Collection*: Freigabe nicht mehr benutzten Speichers

Das verteilte Objektmodell

- Interagierende Objekte sind auf mehr als einen Prozess verteilt
- Begriffe:
  - *Entfernte Objektreferenz*: die Adresse eines Objekts im ganzen verteilten System; muss eindeutig sein
  - *Entfernte Schnittstellen*: die Schnittstelle eines entfernten Objekts, oft beschrieben in einer formalen IDL (interface definition language)
  - *Aktionen*: Methodenaufrufe anderer Objekte können Prozessgrenzen überschreiten
  - *Exceptions*: verteilte Ausführung des Systems erweitert das Spektrum möglicher Fehler

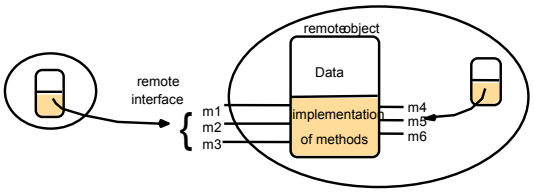
Entfernte und lokale Methodenaufrufe



Schnittstellen entfernter Objekte

- Die entfernte Schnittstelle gibt an, wie auf entfernte Objekte zugegriffen wird.
- Ihre Beschreibung enthält
  - Den Namen der Schnittstelle
  - Möglicherweise Datentypdefinitionen
  - Die Signatur aller entfernt verfügbaren Methoden, bestehend aus
    - Dem Methodennamen
    - Ihrer Ein- und Ausgabeparameter
    - Ihrem Rückgabewert
- Jede Middleware besitzt eine eigene Sprache, um solche Schnittstellen zu beschreiben, die IDL.

Entferntes Objekt und Schnittstelle



CORBA

Inhaltsübersicht

- Was ist CORBA?
- Einsatzgebiete von CORBA
- Verteilte Objektsysteme
- Architektur von CORBA
- Kommunikation zwischen Objekten
- CORBA IDL
- Entwicklung von CORBA-Anwendungen
- CORBA in Enterprise Applications
- CORBA-fähige Tools
- CORBA-Unterstützung in Java

Was ist CORBA?

- CORBA = Common Object Request Broker Architecture
- Zunächst einmal eine Spezifikation, keine Implementierung
- erstellt von der OMG (Object Management Group, http://www.omg.org), einem Konsortium von zahlreichen Firmen und Organisationen
- Ziel: Standard für verteilte Objektsysteme, Interoperabilität versch. Implementierungen

Einsatzgebiete

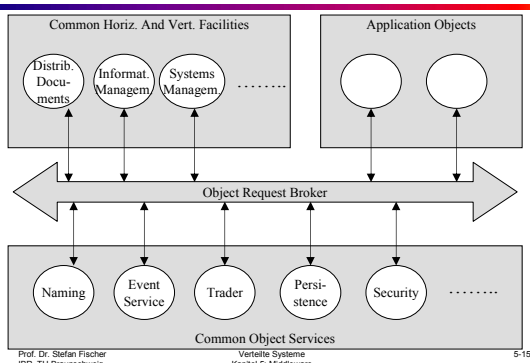
- Vollständige verteilte Systemimplementierungen
- Service-Implementierungen, die von anderen Anwendungen netzweit genutzt werden können
- Erstellen von standardisierten Dienstschnittstellen für Legacy-Anwendungen, z.B. R/3, DB2, Oracle etc.

CORBA-Eigenschaften

- CORBA erlaubt das Schreiben von Anwendungen, die aus einer Sammlung
  - heterogener
  - verteilter
  - miteinander kooperierender Objekte

  bestehen.
- Heterogen bedeutet hier, die Objekte können
  - in versch. Programmiersprachen implementiert sein
  - unter versch. Betriebssystemen laufen
  - von unterschiedlichen Organisationen betreut werden

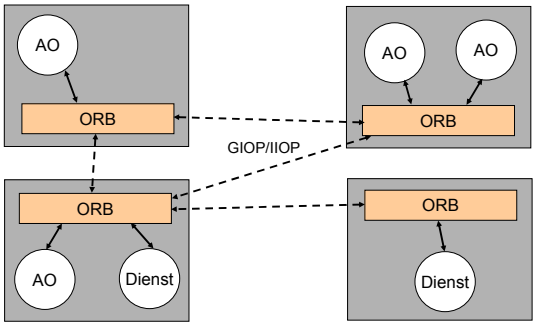
Architektur von CORBA



Object Request Broker

- Zentrale Komponente von CORBA
- Hauptaufgabe:
  - ermöglicht die für die Anwendung transparente Kommunikation von Objekten über das Netz
  - lokale und entfernte Methodenaufrufe sehen praktisch identisch aus
  - ORB sucht das Objekt für den Aufrufer
- Jeder Prozess mit CORBA-Objekten muss einen ORB besitzen.
- ORBs kommunizieren miteinander über das standardisierte GIOP bzw. IIOP im Internet.

Realisierung des ORB



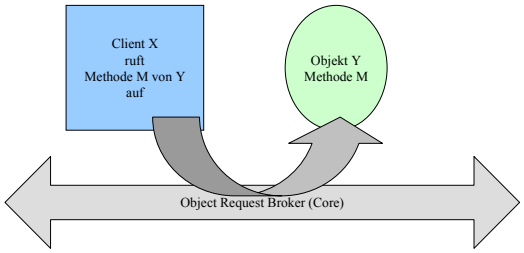
Application Objects

- Dies sind die eigentlichen Anwendungsobjekte.
- Sie werden vom Anwendungsentwickler geschrieben.
- AOs nutzen die ORBs, um miteinander zu kommunizieren und um die verschiedenen CORBA-Dienste zu anzusprechen.
- Erstellung von AOs behandeln wir etwas später.

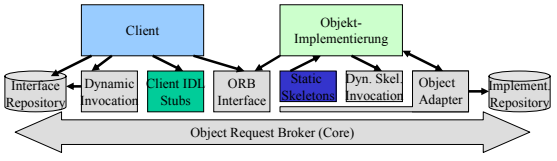
CORBA-Dienste und Facilities

- CORBA definiert eine Reihe von Standarddiensten, die von Anwendungen verwendet werden können.
- Dadurch ergibt sich eine erhebliche Einsparung bei der Anwendungsentwicklung.
- Beispiele:
  - *Persistence Service*: erlaubt das Speichern und Laden von Objekten auf nichtflüchtigem Speicher
  - *Naming Service*: Finden von Objekten aufgrund des Namens
  - *Transaction Service*: erlaubt das Durchführen von Transaktionen („alle Aktionen oder keine“)
  - Trader Service: „Gelbe Seiten“, Finden von AOs aufgrund von Eigenschaften
  - Event Service: Entkopplung von Client und Server; asynchrone Kommunikation

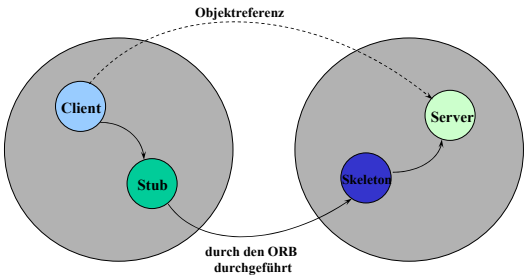
Kommunikation zwischen Objekten



ORB und Objekte im Detail



Virtuelle und tatsächliche Kommunikation



CORBA IDL

- Die *Implementierung* von Objekten kann in verschiedenen Sprachen erfolgen.
- Die *Schnittstelle* muss jedoch auf standardisierte Weise erfolgen, damit jeder Nutzer eines Objekts weiß, wie er es ansprechen kann.
- Zu diesem Zweck wird in CORBA die **Interface Definition Language (IDL)** verwendet.
- Sie erlaubt die Beschreibung der
  - Signaturen der Methoden eines Objekts sowie
  - evtl. dazu benötigter Datentypen

IDL Syntax

- IDL-Beschreibung enthalten die folgenden Elemente
  - Module: zusammengehörige Definitionen einer Anwendung, Schlüsselwort `module`
  - Datentypen: Definition ähnlich wie in C
  - Schnittstelle eines oder mehrerer Objekte: Schlüsselwort: `interface`
  - innerhalb einer Schnittstellenbeschreibung: Methoden und Instanzvariablen, falls diese öffentlich sein sollen

Beispiel

```
module QueueApp {

        struct PersonenInfo {
                string firstname;
                string lastname;
                int age;
        }

        interface Queue {
                boolean      enqueue(in PersonenInfo pi);
                PersonenInfo dequeue();
                boolean      isEmpty();
        }
}
```

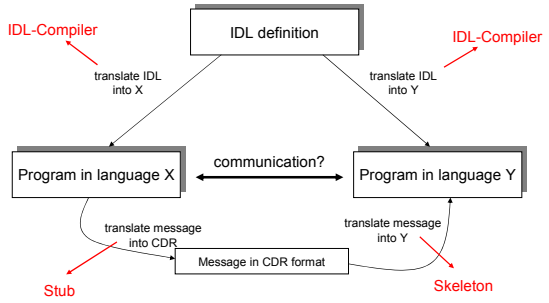

IDL Compiler

- Großer Vorteil einer standardisierten formalen Beschreibung einer Objektschnittstelle: sie ist automatisch behandelbar!
- Zum Beispiel lässt sich automatisch Code für eine Programmiersprache ableiten.
- Damit wird ein wesentlicher Schritt bei der Umsetzung des Designs in eine Implementierung automatisiert.
- Es gibt heute für sehr viele Programmiersprachen schon IDL-Compiler.

Ausgaben des IDL-Compilers

- Ein typischer IDL-Compiler erzeugt
  - die Stubs und Skeletons
  - die leeren Prozeduren (die Köpfe) auf der Server-Seite
  - ein Serverprogramm, das Client-Anfragen entgegen nimmt und die entsprechenden Prozeduren aufruft
  - ein einfaches Client-Programm zum Testen
- Der Programmierer muss dann nur noch
  - die Prozedurrümpfe auf der Server-Seite ausfüllen
  - das Client-Programm entsprechend der gewünschten Anwendung modifizieren

Zur Erinnerung: Gemeinsames Format



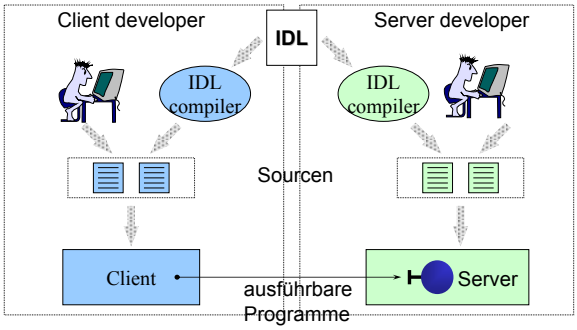
IDL Stubs und Skeletons

- Stubs und Skeletons werden aus der IDL generiert.
- Ein Stub ist die Verbindung zwischen Client und ORB, ein Skeleton die zwischen Server und ORB.
- Nach oben (Client bzw. Server) wird die Schnittstelle des Objekts übersetzt in die verwendete Programmiersprache angeboten.
- Zum ORB hin wird eine Nachrichtenschnittstelle angeboten.
- Stubs und Skeletons übersetzen demnach das eine Format in das andere.

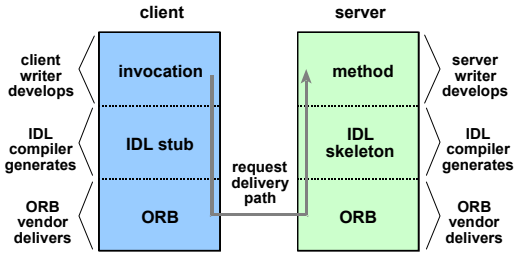
Dynamische Schnittstellen

- Nicht immer sind in einer Anwendung alle Objekte und deren Schnittstellen von Anfang an bekannt.
- Um Objekte dynamisch einbinden zu können, stehen die Dynamic Invocation Interfaces zur Verfügung.
- Um ein solches Objekt zu nutzen, muss sich der Aufrufer zunächst die Schnittstellenbeschreibung besorgen, um die Methoden und Parameter zu erfahren.
- Wird eher selten genutzt.

Entwicklung von CORBA-Anwendungen

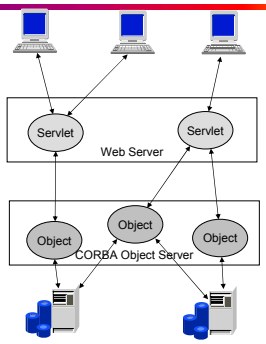


Rollen bei der CORBA-Entwicklung

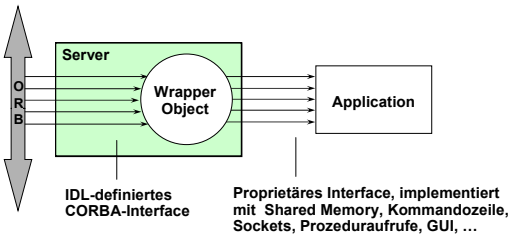


CORBA in Enterprise Applications

- CORBA-Objekte können die komplette Business-Logik-Ebene einnehmen.
- Presentation-Implementierungen werden dann oft als Applets vorgenommen.
- Häufige Variante: Servlets als Front-End, die den Kontakt zu den CORBA-Objekten herstellen
- Oft für Legacy-Anwendungen



Integration von Legacy-Anwendungen



CORBA-fähige Produkte und Tools

- Es gibt heute ein Reihe von Tools, die CORBA-fähig sind. Dies bedeutet zumeist mindestens Bereitstellung
  - eines GIOP/IIOP-kompatiblen ORBs
  - eines IDL-Compilers
  - einiger CORBA-Dienste, vor allem des Name Service
- Bekannte Produkte
  - Sun J2SE
  - Borland Visibroker
  - Objectspace Voyager
  - Oracle Web Request Broker

CORBA-Unterstützung in Java

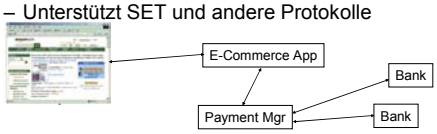
- Der IDL-Compiler in Java heisst in der JDK Version 1.3 `idlj`.
- Hat man seine IDL-Datei geschrieben, so übersetzt man sie mittels
  > `idlj beispiel.idl`
- Ergebnis: mehrere Dateien, die Stubs, Skeletons und Server in Java enthalten

Beispiel: Datum-Server in CORBA

- Wir schreiben einen CORBA-Server, der als Dienst die Rückgabe des aktuellen Datums anbietet.
- Der Dienst wird in der Datei `Datum.idl` definiert.
- Danach übersetzen wir `Datum.idl` und erhalten die entsprechenen Quelldateien in Java.
- Beispieldateien s. Demo in der Vorlesung und Web-Seite des Kurses

Beispiel 2: IBM Payment Manager

- Der IBM Payment Manager übernimmt die Kommunikation mit den Rechnern der Banken
  - Unterstützt SET und andere Protokolle



  - Im sog. hosted Szenario wird die Abbuchung, z.B. von Kreditkarte, als Service angeboten

Payment Manager Interface

- Sessionverwaltung

```
interface PaymentManager {
 PaymentServerHandle etInitializeAPI(in long sync,
                                     out long primaryRC);
    long etCloseAPI(Handle handle);
}
```

- Zahlungsvorgänge

```
interface PaymentServerHandle {
    long etAcceptPayment(
        in Key keys,                        // order & merchant ID
        in Amount orderamount,              // amount & currency
        in string paymenttype,              // „SET“
        in OrderDescription orderdescription, // comments
        in Approve approveinfo,          // split payment allowed,
etc.
        in ProtocolData protocoldata,       // Credit Card Info
        out long secondaryrc,               // return code
        out ReturnData returndata);         // return message
    ...
}
```

Java RMI

Grundlagen von RMI

- Definiert ein Rahmenwerk für die Kommunikation von Java-Objekten unabhängig von ihrem Ort
- Eine reine Java-Lösung
- Alle entfernten Objekte müssen ein entferntes Interface besitzen
- Es sind Werkzeuge vorhanden für die Generierung von Stubs und Skeletons.
- JDK stellt eine Implementierung des Naming Service zur Verfügung, die *RMIregistry*.
- Ein RMI-Dämon erlaubt einen flexible (on-demand)-Instanziierung von Objekten.

Schnittstellen-Definition

- Schnittstellen werden definiert
  - Mit Hilfe des Java `interface` Konstrukts
  - Indem sie die Eigenschaften des `Remote` interface erben
  - Und indem sie `RemoteException`s auslösen können.
- Ansonsten können alle Java-Typen verwendet werden. Neue Klassen können definiert und als Parameter von Methoden verwendet werden.
- Beispiel: der Date-Server in Java

```
import java.rmi.*;
import java.util.Date;
public interface DateServer extends Remote {
        public Date getDate() throws RemoteException;
}
```

Das entfernte Objekt

- Um den von der Schnittstelle „versprochenen“ Dienst zu erbringen, muss es ein entferntes Objekt geben, das die Methoden der Schnittstelle implementiert.
- Gewöhnlich erweitert es die Klasse `UnicastRemoteObject` was aus dem Objekt einen nicht-replizierten Server macht, der über TCP kommuniziert.
- Beispiel:

```
public class DateServerImpl extends UnicastRemoteObject
                            implements DateServer {
        ...
}
```


Der RMI-Compiler

- Basierend auf der Implementierung des Objekts mit seinen Methoden kann man nun automatisch Stubs und Skeletons implementieren.
- JDK stellt ein Werkzeug namens `rmic` für diesen Zweck zur Verfügung.
- Folgender Aufruf des Werkzeugs

```
> rmic DateServerImpl
```

- Erzeugt zwei Dateien:
  - `DateServerImpl_Stub.class`
  - `DateServerImpl_Skel.class`

Die RMI-Registry

- Die RMI-Registry, implementiert durch das Programm `rmiregistry`, ist der Name Server für Java RMI.
- Server registrieren entfernte Objekte unter einem Namen.
- Clients fragen die Datenbank ab, um die Referenz für einen entfernten Dienst zu erhalten.
- Name Services behandeln wir noch ausführlich in Kapitel 7.

Die *Naming* Klasse der Java RMIregistry

void rebind (String name, Remote obj)
This method is used by a server to register the identifier of a remote object by name.

void bind (String name, Remote obj)
This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)
This method removes a binding.

Remote lookup(String name)
This method is used by clients to look up a remote object by name. A remote object reference is returned.

String [] list()
This method returns an array of Strings containing the names bound in the registry.

Erzeugen einer RMI-Anwendung

1. Definiere das Remote Interface
2. Implementiere das Remote Interface durch ein Remote Object
3. Generiere Stubs und Skeletons mit `rmic`
4. Schreibe einen Client
5. Starte den Name Server mit `rmiregistry`
6. Starte den Server
7. Starte den Client

Beispiel: der Date Server

- Das Beispiel implementiert einen RMI-Server, der das aktuelle Datum seiner Maschine zurückgibt.
- Wir müssen selbst die folgenden Dateien schreiben:
  - `DateServer.java`: die Schnittstelle
  - `DateServerImpl.java`: das entfernte Objekt
  - `DateClient.java`: der Client

Date Server Interface

```
/* Copyright (c) 1997-1999 Merlin Hughes, Michael Shoffner, Derek Hamner;
* all rights reserved; see license.txt for details. */

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface DateServer extends Remote {
        public Date getDate () throws RemoteException;
}
```


DateServerImpl.java

```
/* Copyright (c) 1997-1999 Merlin Hughes, Michael Shoffner, Derek Hamner;
* all rights reserved; see license.txt for details. */
import java.rmi.*;
import java.rmi.server.*;
import java.util.Date;
public class DateServerImpl extends UnicastRemoteObject
                                implements DateServer {
        public DateServerImpl () throws RemoteException { }
        public Date getDate () { return new Date (); }
        public static void main (String[] args) throws Exception {
                DateServerImpl dateServer = new DateServerImpl ();
                Naming.bind ("Date Server", dateServer);
        }
}
```


DateClient.java

```
/* Copyright (c) 1997-1999 Merlin Hughes, Michael Shoffner, Derek Hamner;
* all rights reserved; see license.txt for details. */
import java.rmi.Naming;
import java.util.Date;
public class DateClient {
        public static void main (String[] args) throws Exception {
                if (args.length != 1)
                        throw new IllegalArgumentException
                                        ("Syntax: DateClient ");
                DateServer dateServer = (DateServer)
                        Naming.lookup ("rmi://" + args[0] + "/Date Server");
                Date when = dateServer.getDate ();
                System.out.println (when);
        }
}
```

Automatische Objektaktivierung

- In großen Systemen mit Tausenden von Objekten ist es unrealistisch, immer alle Objekte im Speicher aktiv zu haben.
- Mit Hilfe des Java-Aktivierungsmechanismus können Objekte persistent gemacht und erst bei Bedarf wieder instanziiert werden.
- Dieses Objektmanagement wird durch den Dämon `rmid` realisiert.
- Jini basiert sehr weitgehend auf diesem Aktivierungsmechanismus.

Vergleich: CORBA vs. RMI

- RMI ist eine reine Java-Variante, aber ähnlich zu CORBA
  - \+ Die Programmierung ist einfacher als bei CORBA
  - \+ Komplexe Objekte können übergeben werden
  - – Nur für Java
- Zusammenfassung:
  - – Gute Alternative für reine Java-Lösungen
  - – Die Übertragung von aktiven Objekten wird selten genutzt (z.B. für mobile Agenten)

Web Services

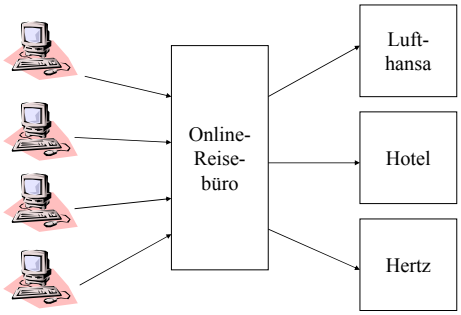
Web Services

- Web Services sind die jüngste Entwicklung im Bereich Middleware.
- Sie beruhen auf der Verwendung standardisierter Protokolle und Sprachen wie z.B.
  - XML
  - SOAP
  - Web Service Description Language (WSDL)
  - Universal Description, Discovery, and Integration (UDDI)
- Wir wollen die wichtigsten Eigenschaften dieser Technologie betrachten, allerdings sind Web Services vor allem Thema der Vorlesung „Web-Anwendungen mit Java und XML“

Ein Beispiel

- Stellen Sie sich vor ...
- ... Sie seien ein junger dynamischer Uni-Absolvent, der sich mutig in seine erste Unternehmensgründung stürzt.
- Der Plan: Sie wollen ein Online-Reisebüro aufmachen – sensationell!
- Sie wollen aber möglichst wenig selbst machen, sondern existierende Angebote einbinden – noch besser!

Architektur Ihrer Anwendung



Ein Geschäftsvorfall

- Angenommen, ein Kunde will eine Mallorca-Pauschalreise buchen, samt Flug, Hotel, Bustouren etc.
- Sie kaufen diese Leistungen ein von Lufthansa, Hotel Playa del Sol, etc.
- Leider stehen Ihnen von den entsprechenden Firmen immer nur deren öffentliche Web-Seiten zur Verfügung.
- Wo ist das Problem??

Das Problem

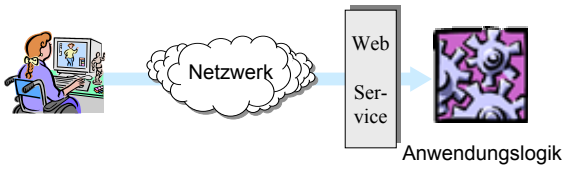
- Sie bekommen von der Lufthansa Web-Seite nur HTML.
- HTML ist ein Seitenbeschreibungsformat, kein Datenformat.
- Für Ihre Anwendungen brauchen Sie aber die Daten der Lufthansa, nicht die „schönen“ (menschenlesbaren) Seiten.
- Eine Lösung, die heute heiß diskutiert wird, heißt „Web Services“.

Definition Web Services

- **Web Service = Schnittstelle für den netzbasierten Zugriff auf eine Anwendungsfunktionalität, vollständig basierend auf Standard-Internet-Technologien**

- IBM: “Web services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the World Wide Web.”
- Microsoft: “a Web service is programmable application logic, accessible using standard Internet protocols,”
- Gartner Group: “a software component that represents a business function (or a business service) and can be accessed by another application (a client, a server or another Web service) over public networks using generally available ubiquitous protocols and transports (i.e. SOAP over HTTP).”

Architektur Web Services



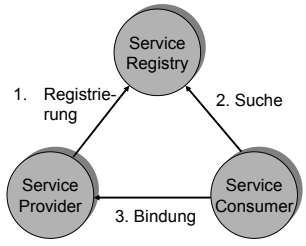
Also: genau das Modell wie bei anderer Middleware, wenn es um Einbindung von Legacy-Software geht.

Anforderungen an eine Lösung

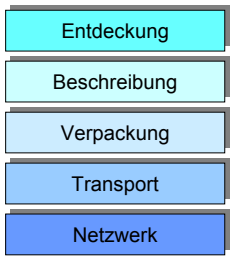
- Verwendung strukturierter Datentypen
- Verwendung offener Standards
- Flexible Kombinierbarkeit von Dienstangeboten (ganz schlecht bei Web Applications wie Amazon)
- Leichtes Bekanntmachen und Finden von Diensten (was nutzt ein guter Dienst, wenn ihn keiner kennt)

Just-in-time Integration

- Eine Anwendung findet erst zur Laufzeit heraus, welche Web Services sie tatsächlich nutzen will.
- Der Dienst wird dann „just in time“ integriert.
- Vergleichbar mit „late binding“ in objektorientierter Programmierung.
- Vorgehen beruht auf der „Web Service Architektur“

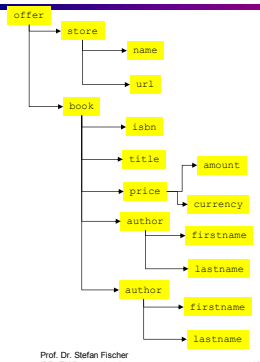


Web Services Technology Stack



- Modularisierung der Aufgaben
- Angepasst an Internet Stack
- Entdeckung (discovery): finde die Beschreibung von WebServices
- Beschreibung (description): beschreibe einen Web Service
- Verpackung (packaging): kodiere die Daten in einem allg. verständlichen Format (=marshalling, serialization)
- Transport: Datentransport zwischen Prozessen
- Netzwerk: Kommunikation zwischen Machinen im Netz

XML-Dokumente




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<offer>
  <store>
    <name>Amazon</name>
    <url>http://www.amazon.com</url>
  </store>
  <book>
    <isbn>1575213966</isbn>
    <title>Teach yourself XML in 21 days</title>
    <price type="retail">
      <amount>23.99</amount>
      <currency>USD</currency>
    </price>
    <author>
      <firstname>Simon</firstname>
      <lastname>North</lastname>
    </author>
    <author>
      <firstname>Paul</firstname>
      <lastname>Hermans</lastname>
    </author>
  </book>
</offer>
```

Processing Instruction

Anatomie von XML Dokumenten

- Die erste Zeile gibt den Typ des Dokuments an.
- Jedes Dokument entspricht einer Baumstruktur
- Die Baumknoten werden Elemente genannt
- Es kann nur ein Element an der Wurzel geben
  - Eine Applikation weiß sofort wann das Dokument zu ende ist
- Syntaxregeln müssen strikt eingehalten werden
  - Jedes Starttag muss ein Endtag haben
  - Attributwerte müssen mit ‘ oder “ abgegrenzt werden

Namespaces

- Problem: werden Daten aus zwei Bereichen (Namespaces) im selben Dokument gespeichert, kann es zu Namenskollisionen kommen
  - quote (Stockquote)
  - quote (Literatur)
- Lösung:
  - Elemente eines Namespaces erhalten denselben Präfix
  - Dieser Präfix wird einer URL zugeordnet

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <edi:price units='Euro'>32.18</edi:price>
</x>
```


Document Type Definition

- Um ein Dokument zu validieren, muss das Dokument eine DTD referenzieren:

```
<?xml version="1.0"
<!DOCTYPE offer SYSTEM "offer.dtd">
<offer> ...
```

- Die DTD besteht aus drei wesentlichen Bestandteilen
  - Elementdefinitionen

```
<!ELEMENT book (isbn, title, price+, author+)
```

  - Attributdefinitionen

```
<!ATTLIST price type (retail | wholesale)
#REQUIRED>
```

  - Entity Definitionen

```
<!ENTITY legal_disclaimer SYSTEM 'legal.txt'>
```


Wieso eine Alternative zu DTDs?

- Die Syntax ist in der XML Welt unnatürlich
- Keine Datentypen, nur Strings
- Begrenzte Erweiterbarkeit (Vererbung)
- Funktionieren nicht mit Namespaces

- XML Schema behebt diese Probleme
  - Die Idee bleibt dieselbe
  - Empfehlung liegt noch nicht in der endgültigen Version vor
  - Parser unterstützen derzeit nicht alle Features

XML Schema

- Um ein Dokument zu validieren, muss das Dokument ein Schema referenzieren:

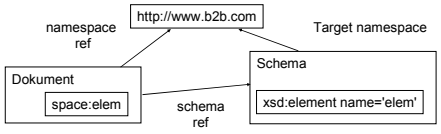
```
<?xml version="1.0"
<offer xmlns:xsi="http://www.w3.org/1999/XMLSchema-
instance"
       xsi:noNamespaceSchemaLocation="offer.xsd">
```

- Das XML Schema bietet folgende Möglichkeiten
  - Primitive Datentypen
    `<element name="float-element" type="float" />`
  - Deklaration von Datentypen (Elementen)
    `<complexType name="studenttype"> ... </ complexType>`
  - Erweiterung von Datentypen

```
<simpleType name="studentid" base="string">
  <pattern value="[A-Z][0-9]+" />
</simpleType>
```


XML Schema und Namespaces

- Jedes Schema ist einem Namespace zugeordnet
  `<xsd:schema targetNamespace='http://www.b2b.com' />`
  - Somit können in einem Dokument auch Elemente verschiedener Schemata verwendet werden



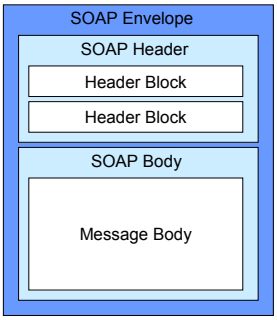
  - Es ist sogar möglich, in einem Schema Elemente eines anderen Schemas zu verwenden

Was ist SOAP?

- Ein Mechanismus zum Verpacken von XML-Nachrichten (vergleiche „Marshalling“).
- Dazu bietet SOAP
  - Nachrichtenformate
  - Erweiterbarkeit durch XML Schema und Name Spaces
  - Einen flexiblen Mechanismus zur Datenrepräsentation auch für bereits serialisierte Daten
  - Eine Konvention zur Beschreibung von RPCs
  - Bindungen an verschiedene Transportprotokolle

SOAP-Nachrichten

- Eine SOAP-Nachricht besteht zunächst auf der obersten Ebene aus einem Envelope.
- Der Envelope enthält einen optionalen Header sowie die eigentliche Nachricht im SOAP Body.
- Aufgabe des Headers: Infos über Transaktionskontexte, Authentifizierung, Autorisierung, Routing- und Auslieferungsinformationen



RPC mit SOAP

- Remote Procedure Calls bestehen gewöhnlich aus zwei Nachrichten (Request-Response)
  - Eine Nachricht für die Anfrage an die entfernte Prozedur
  - Eine Nachricht für die Antwort
- Die Antwortnachricht ist nicht zwingend.

Beispiel: Ein SOAP-Request

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <m:transaction xmlns:m="soap-transaction"
                   s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>

  <s:Body>
    <n:getPhoneNumber xmlns:n="urn:Telefonbuch">
      <nachname xsi:type="xsd:string">
        Fischer
      </nachname>
    </n:getPhoneNumber>
  </s:Body>
</s:Envelope>
```

Beispiel: eine SOAP-Response

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Body>
    <n:getPhoneNumberResponse xmlns:n="urn:Telefonbuch">
      <value xsi:type="xsd:string">
        0531 391 3294
      </value>
    </n:getPhoneNumberResponse>
  </s:Body>
</s:Envelope>
```


Transportmechanismen

- SOAP-Nachrichten können über verschiedene Transportmechanismen befördert werden
  - HTTP ist der Standard und wird im SOAP-Standard auch besonders hervorgehoben
  - HTTP/S für verschlüsselte Aufrufe
  - SMTP für asynchrone Aufrufe von Funktionen (es wird eine Email mit der SOAP-Nachricht geschickt)
  - FTP zur automatischen Authentisierung

Beispiel: SOAP über HTTP

```
POST /Telefonbuch HTTP/1.1
Content-Type: text/xml
Content-Length: xxxx
SOAP-Action: "urn:Telfonbuch#getPhoneNumber"

<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <m:transaction xmlns:m="soap-transaction"
                   s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getPhoneNumber xmlns:n="urn:Telefonbuch">
      <nachname xsi:type="xsd:string">
        Fischer
      </nachname>
    </n:getPhoneNumber>
  </s:Body>
</s:Envelope>
```

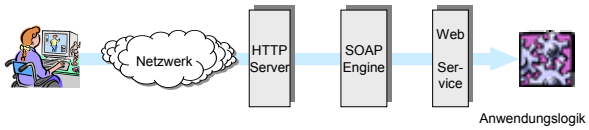
SOAP in der Praxis

- Frage: wie sende ich nun in der Praxis tatsächlich SOAP-Nachrichten von A nach B?
  - Muss ich die Nachrichten selbst in SOAP/XML kodieren?
  - Wer empfängt die Nachrichten und gibt sie an den web Service weiter?
  - Wie bekomme ich die Antworten zurück
- Lösung: man benötigt eine SOAP-Engine, also ein Stück Software, das SOAP-Nachrichten verarbeiten kann

SOAP Engines

- Typischerweise werden SOAP Engines mit einem HTTP-, SMTP- oder FTP-Server (usw.) betrieben, je nach dem, welcher Transportmechanismus benutzt wird.
- Vorgehen:
  - Transport nimmt die Nachricht entgegen
  - entpackt sie
  - Gibt sie an die SOAP Engine weiter
- SOAP Engine interpretiert die Nachricht und ruft den entsprechenden Web Service auf

SOAP Engines (II)



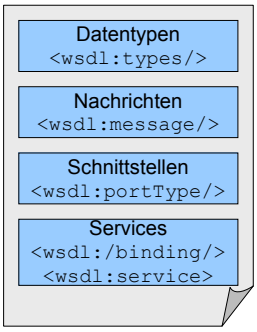
- Bekannte SOAP-Engines (s. auch Tools-Abschnitt):
  - Apache Axis (SOAP Engine=Servlet)
  - SOAP::Lite
  - .NET
- SOAP-Nachrichten werden mittels APIs erstellt und bearbeitet – Sie müssen also kein SOAP „sprechen“!

WSDL

- WSDl ist ein „proposed standard“ der W3C.
- Vorgeschlagen von IBM, Microsoft und anderen im September 2000.
- Aktuelle Version: 1.1
- Verfügbar unter: http://www.w3.org/TR/wsdl
- WSDL ist wie SOAP eine Anwendung der XML-Spezifikation. Ein WSDL-Dokument ist konform zur WSDL-Schema-Definition.
- Eine WSDL-Spezifikation beantwortet drei Fragen:
  - Was tut ein Service – welche Operationen stellt er bereit?
  - Wie wird auf den Service zugegriffen – Datenformate, Nachrichten?
  - Wo findet sich der Service – URL, etc.?

Struktur eines WSDL-Dokuments

- Ein Service ist also eine Sammlung von Ports.
- Ein Port hat eine abstrakte Definition (portType) und eine konkrete Realisierung (binding).
- PortTypes sind Sammlungen von abstrakten Operationen, etc.
- Entsprechend sind WSDL-Dokumente aufgebaut.



WSDL Werkzeuge

- Sowohl Axis als auch .NET sind in der Lage, WSDL automatisch aus der Implementation zu generieren
  - Das WSDL Dokument wird über einen GET Anfrage auf http://host/service?WSDL erzeugt
- Umgekehrt, können auch Stubs aus WSDL generiert werden
  - `wsdl PriceCheck.wsdl`
  - `java org.apache.axis.wsdl.WSDL2Java PriceCheck.wsdl`
- Web Service erscheint wie eine lokale Bibliothek

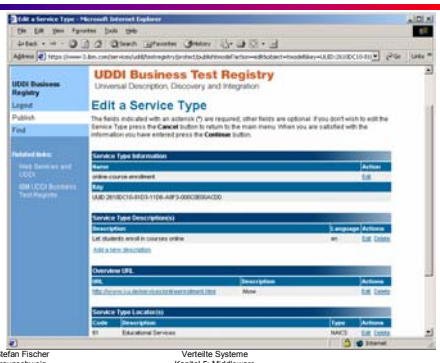
UDDI

- UDDI steht für Universal Description Discovery and Integration
  - Wurde Ende 2000 von IBM, Microsoft und Ariba initiiert
  - Inzwischen sind einige hundert Firmen dem UDDI Konsortium beigetreten (u.a. Oracle, SAP)
- UDDI spezifiziert Standards zur Publikation und Suche von Web Services
- Schnittstellen
  - Manuell per Browser
  - Automatisiert per Web Services API

Das Globale UDDI Repository

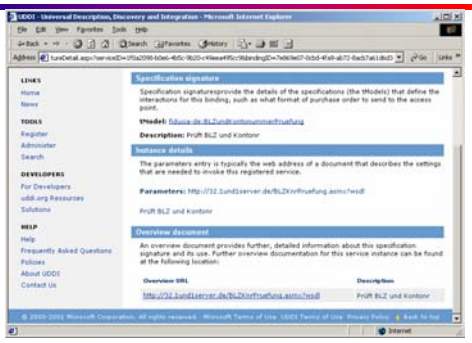
- Die Spezifikationen werden in einem globalem Web Service Repository implementiert
- Die drei Initiatoren unterhalten die technische Infrastruktur
  - Drei Server die untereinander Informationen replizieren
    - http://uddi.microsoft.com
    - http://www.ibm.com/services/uddi
    - http://www.ariba.com
  - Registrierungsmechanismus
  - Filterung von SPAM Einträgen

UDDI Browser Interface: Publikation



UDDI Browser Interface: Suche

UDDI Browser Interface: Suche II



UDDI Datenstrukturen: White Pages

- White Pages beinhalten Informationen über die Institution, die Services bereitstellt
  - Name
  - Telefon
- Dies ist vergleichbar mit der Information im Telefonbuch

UDDI Datenstrukturen: Yellow Pages

- Die Yellow Pages erlauben eine Klassifikation der Institutionen nach verschiedenen Schemata
- Momentan werden unterstützt:
  - Geographisch (ISO 3166)
  - Branchen (NAICS)
  - Services / Produkte (UN / SPSC)
- Es können auch eigene Klassifikationsschemata definiert werden (besonders interessant für lokale UDDI Repositories)
- Dies entspricht den Gelben Seiten

UDDI Datenstrukturen: Green Pages

- Green Pages beinhalten technische Informationen
  - Angebotene Web Services
  - Wie sind diese aufzurufen
- Technical Models werden zur Identifikation einer bestimmten Art von Service verwendet
  - Diese kann auf ein WSDL Dokument verweisen
  - Sie kann sich aber auch auf vorab definierte Standards wie z.B. RosettaNet beziehen

Service Discovery zur Designzeit

- Services werden dem Entwickler mit Hilfe eines Browsing Werkzeuges angezeigt
  - Entwickler wählt den aufzurufenden Service über die Browser Schnittstelle
  - Stubs werden generiert
  - Programmieren des Service Aufrufs
  - Kompilieren des Projekts
- Funktioniert ähnlich wie bereits bekannte Objektbrowser

Service Discovery zur Laufzeit

- Die Auswahl der Services erfolgt erst zur Laufzeit
  - Entwickler wählt tModel (Service Interface ID)
  - Stubs werden generiert
  - Zur Laufzeit Suche der passenden Services über die Web Services UDDI API
  - Diese werden dann über den Stub aufgerufen
- Mittels dynamisch compiliertem und gebundenem Code könnte sogar die Auswahl der tModels zur Laufzeit erfolgen

UDDI Werkzeuge

- Sowohl Java Apache als auch .NET bieten als Basisfunktionalität eine UDDI Proxy
  - Web Service API wird komfortabel als eine Klassenbibliothek angeboten
- Visual Studio .NET hat einen UDDI Browser integriert
  - Bei Apache erfolgt dies etwas weniger komfortabel per Web Browser
- Visual Studio .NET bietet weiterhin an, gerade entwickelte Services direkt an einem UDDI Server anzumelden

Vergleich Web Services - CORBA

Eigenschaft	Web Service	Java / CORBA
Client	java.net.URL	JDK ORB
Server	Web Server	JDK ORB
Name Service	DNS	tnameserv
Interface Spezifikation	WSDL	IDL
Adressierung	URL	Objekt Referenz
Kodierung	text	binär
Protokoll	HTTP, SMTP, ...	IIOP
Prog. Paradigma	Prozedural	Objektorientiert
Finden von Services	UDDI	CORBA Repository

Fazit

- Web Services haben weniger Funktionalität
  - Kann meist durch einige Zeilen Code ausgeglichen werden
- Augenmerk liegt auf
  - Interoperabilität
  - Integration organisatorisch und technisch heterogener Systeme
  - Wiederverwendung von bewährten Internet Protokollen und Software
  - Modularität durch Header Mechanismus

Literatur und Web Sites

- G. Brose et al.: *Java Programming with CORBA*, 3. Auflage, Wiley, 2001.
- W. Grosso: *Java RMI*, O'Reilly, 2001.
- S. Graham et al.: *Building Web Services with Java*, Sams, 2001.
- A. Eberhart, S. Fischer: *Java-Bausteine für E-Commerce-Anwendungen*, Hanser, 2001.

- OMG: http://www.omg.org
- Microsoft .net: http://www.microsoft.com/net/
- Java Web Services Tutorial: http://java.sun.com/webservices/docs/1.0/tutorial/index.html