Überblick

Web-Services
- Beschreibung von Web-Services
- Implementierung von Web-Services
- Registrierung von Web-Services
- Aufgabe 1

Web Services Description Language (WSDL)

- Beschreibungssprache für Web-Services
- Repräsentation als XML-Dokument (Beispiel für WSDL 1.1)

```
<definitions [...] targetNamespace="http://adder.mw/" name="MWAdderSvc">
<types/>
<message name="add">
  <part name="arg0" type="xsd:int"></part>
  <part name="arg1" type="xsd:int"></part>
</message>
<message name="addResponse">
  <part name="return" type="xsd:int"></part>
</message>
<portType name="MWAdderSvcInterface">
  <operation name="add" parameterOrder="arg0 arg1">
    <input message="tns:add"></input>
    <output message="tns:addResponse"></output>
  </operation>
</portType>
<binding name="MWAdderSvcInterfacePortBinding" type="tns:MWAdderSvcInterface">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc">
  </soap:binding>
  <operation name="add">
    <soap:operation soapAction=""></soap:operation>
    <input>
      <soap:body use="literal" namespace="http://adder.mw/"></soap:body>
    </input>
    <output>
      <soap:body use="literal" namespace="http://adder.mw/"></soap:body>
    </output>
  </operation>
</binding>
<service name="MWAdderSvc">
  <port name="MWAdderSvcInterfacePort" binding="tns:MWAdderSvcInterfacePortBinding">
    <soap:address location="http://localhost:12345/MWAdderSvc"></soap:address>
  </port>
</service>
</definitions>
```

Struktur eines WSDL-Dokuments

- Äußerstes Element: `definitions` (WSDL 1.1) bzw. `description` (WSDL 2.0)
- Repräsentation von Informationen in inneren Elementen
  - Datentypen (`types`)
    - Beschreibung der vom Web-Service verwendeten Datentypen
    - Bleibt leer, falls ausschließlich Standarddatentypen zum Einsatz kommen
  - Nachrichten (`message`)
    - Beschreibung der zur Kommunikation mit dem Web-Service benötigten Nachrichtenformate (z. B. Aufrufparameter)
    - Beispiele

```
<message name="add">
    <part name="arg0" type="xsd:int"/>
    <part name="arg1" type="xsd:int"/>
</message>
<message name="addResponse">
    <part name="return" type="xsd:int"/>
</message>
```

[`xsd`: Präfix der XML Schema Definition, einer Kollektion von Standarddatentypen.]

Struktur eines WSDL-Dokuments

- Schnittstellen (WSDL 1.1: `portType`, WSDL 2.0: `interface`)
  - Beschreibung der angebotenen Methoden
  - Sammlung von Methoden: *Port* (WSDL 1.1) bzw. *Endpoint* (WSDL 2.0)
  - Zuordnung, welche Nachrichten bei welchen Methoden zum Einsatz kommen
  - Beispiel

```
<portType name="MWAdderSvcInterface">
    <operation name="add" parameterOrder="arg0 arg1">
        <input message="tns:add"/>
        <output message="tns:addResponse"/>
    </operation>
</portType>
```

[`tns`: Präfix des Standardnamensraums („`target namespace`“).]

- Kommunikationsprotokoll (`binding`)
  - Beschreibung der zu verwendenden SOAP-Nachrichtenformate
  - Nachrichtenformate können/müssen für jede Operation separat definiert sein
- Dienst (`service`)
  - Beschreibung der vom Web-Service unterstützten Ports
  - Auflistung der Zugriffsmöglichkeiten auf den Web-Service (z. B. Adresse)

Überblick

Implementierung von Web-Services

- Typische Vorgehensweise
  1. Implementierung des Diensts
  2. Zusammenfassung der Dateien zu einem Archiv
  3. Integration des Archivs in einen Web-Server (z. B. Tomcat)
  4. Implementierung des Clients
- Java API for XML Web Services (JAX-WS)
  - Standardmäßig integriert in Java 6
  - Client-Service-Kommunikation via SOAP
  - Zentrales Hilfsmittel: Annotationen
  - Tutorial: `http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html`

Annotationen

- Web-Service-Endpunkt (`@javax.jws.WebService`)
  - Festlegung einer Klasse als Web-Service-Endpunkt
  - Zentrale Parameter
    - Name (`name`): Abbildung auf das WSDL-Element `portType`
    - Dienstname (`serviceName`): Abbildung auf das WSDL-Element `service`
- SOAP-Anbindung (`@javax.jws.soap.SOAPBinding`)
  - Konfiguration des Nachrichtenformats
  - Zentrale Parameter
    - Codierung von Methodenaufrufen (`style`): *document* (Default) oder *RPC*
    - Information über Parameter-Codierung (`use`): *encoding* oder *literal* (Default)
    - Für Details siehe `http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/`
- Web-Service-Methode (`@javax.jws.WebMethod`)
  - Freigabe einer Methode für die Nutzung per Web-Service
  - Methode muss zwingend als `public` deklariert sein
- ...

Erzeugung des Web-Service-Endpunkts

- Klasse: `javax.xml.ws.Endpoint`
- Statische Methode: Erzeugung des Endpunkts und Veröffentlichung der WSDL-Beschreibung des Web-Service

```
Endpoint publish(String address, Object implementor);
```

  - `address`: Adresse, unter der die Service-WSDL veröffentlicht werden soll
  - `implementor`: Implementierung des Web-Service
- Aufruf von `publish()`: Start eines in Java eingebetteten *Web-Server*
  - Verwendung einer Default-Konfiguration
  - Aufgaben
    - Annahme der SOAP-Anfrage
    - Aufruf der angeforderten Methode der Web-Service-Implementierung
    - Zurücksenden einer SOAP-Antwort an den Client

Erzeugung des Client-Kommunikationsendpunkts

- Die Client-Anwendung kommuniziert mit dem Web-Service unter Verwendung eines *Proxy*
  - Proxy fungiert als Stellvertreter für die eigentliche Dienstimplementierung
  - Lokale Methodenaufrufe am Proxy
- Aufgaben des Proxy
  - Abfangen des Methodenaufrufs
  - Umwandeln des Methodenaufrufs in eine SOAP-Anfrage
  - Senden der Anfrage an den Web-Server
  - Empfangen der SOAP-Antwort vom Web-Server
  - Auspacken des Rückgabewerts aus der Antwort
  - Rückgabe des Ergebnisses an den Aufrufer der Methode
- Automatisierte Erzeugung von Proxy-Hilfsklassen
  - Kommandozeilen-Tool: `wsimport`
  - Als Grundlage dient die WSDL-Beschreibung des Diensts
  - Dienstspezifische Hilfsklassen: Service-Zugangspunkt und -Schnittstelle

Arrays als Parameter/Rückgabewerte

- Zusätzliche Erzeugung einer Hilfsklasse für jeden verwendeten Array-Typ durch `wsimport`
- Kapselung einer Liste von Objekten des Array-Typs
  - Zugriff auf die Liste mittels `getItem()`-Methode
  - Einfügen/Auslesen von Objekten muss über diese Liste erfolgen
- Beispiel: Array-Klasse für String-Arrays (`StringArray`)

```
String[] s = { "a", "b" };

// Einpacken
StringArray sa = new StringArray();
List<String> sal = sa.getItem();
sal.add(s[0]);
sal.add(s[1]);

// Auspacken
List<String> xsal = sa.getItem();
String[] xs = xsal.toArray(new String[0]);
```

Beispiel: Addierer als Web-Service — Server-Seite

- Schnittstelle (`mw/adder/MWAdderSvcInterface.java`)

```
package mw.adder;

public interface MWAdderSvcInterface {
    public int add(int a, int b);
}
```

- Implementierung (`mw/adder/MWAdderSvcImpl.java`)

```
[...] // Package und imports

@WebService(name = "MWAdderSvcInterface", serviceName = "MWAdderSvc")
@SOAPBinding(style = SOAPBinding.Style.RPC)

public class MWAdderSvcImpl implements MWAdderSvcInterface {

    @WebMethod
    public int add(int a, int b) {
        return a + b;
    }

}
```

Beispiel: Addierer als Web-Service — Server-Seite

- Server (`mw/adder/MWAdderServer.java`)

```
[...] // Package und imports

public class MWAdderServer {

    public static void main(String[] args) throws Exception {
       String wsdl = "http://localhost:12345/MWAdderSvc?wsdl";

       MWAdderSvcInterface adder = new MWAdderSvcImpl();
       Endpoint e = Endpoint.publish(wsdl, adder);
       System.out.println("Adder ready: " + e.isPublished());

       while(true) {
         Thread.sleep(Long.MAX_VALUE);
       }
    }

}
```

Beispiel: Addierer als Web-Service — WSDL

- Nach dem Server-Start: Service-WSDL abrufbar (z. B. im Browser)

```
<definitions [...] targetNamespace="http://adder.mw/" name="MWAdderSvc">
<types/>
<message name="add">
  <part name="arg0" type="xsd:int"></part>
  <part name="arg1" type="xsd:int"></part>
</message>
<message name="addResponse">
  <part name="return" type="xsd:int"></part>
</message>
<portType name="MWAdderSvcInterface">
  <operation name="add" parameterOrder="arg0 arg1">
    <input message="tns:add"></input>
    <output message="tns:addResponse"></output>
  </operation>
</portType>
<binding name="MWAdderSvcInterfacePortBinding" type="tns:MWAdderSvcInterface">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc">
  </soap:binding>
  <operation name="add">
    <soap:operation soapAction=""></soap:operation>
    <input>
      <soap:body use="literal" namespace="http://adder.mw/"></soap:body>
    </input>
    <output>
      <soap:body use="literal" namespace="http://adder.mw/"></soap:body>
    </output>
  </operation>
</binding>
<service name="MWAdderSvc">
  <port name="MWAdderSvcInterfacePort" binding="tns:MWAdderSvcInterfacePortBinding">
    <soap:address location="http://localhost:12345/MWAdderSvc"></soap:address>
  </port>
</service>
</definitions>
```

Beispiel: Addierer als Web-Service — Client-Seite

- Erzeugung der Proxy-Hilfsklassen für die Client-Seite
  - Kommandozeilenaufruf

```
> wsimport -p mw.adderclient -d bin -s src
           -keep http://localhost:12345/MWAdderSvc?wsdl
parsing WSDL...

generating code...

compiling code...
```

[Annahmen: Quellordner mit Source-Dateien ist `src`, Zielordner für Class-Dateien ist `bin`.]

  - Erzeugte Dateien
    - Subpackage: `mw.adderclient`
    - Service-Zugangspunkt: `mw/adderclient/MWAdderSvc.java`
    - Service-Schnittstelle: `mw/adderclient/MWAdderSvcInterface.java`
- Client-Anwendung (`MWAdderClient.java`)
  - Implementierung mittels Proxy-Hilfsklassen
  - Zugriff auf Web-Service wirkt wie lokaler Methodenaufruf

Beispiel: Addierer als Web-Service — Client-Anwendung

```
import mw.adderclient.MWAdderSvc;
import mw.adderclient.MWAdderSvcInterface; // Achtung: NICHT
                                           // mw.adder.MWAdderSvcInterface

public class MWAdderClient {
    private MWAdderSvcInterface adder;     // Proxy

    public MWAdderClient() {
        MWAdderSvc service = new MWAdderSvc();
        adder = service.getMWAdderSvcInterfacePort();
    }

    public void add(int a, int b) {
        int result = adder.add(a, b);
        System.out.println(a + " + " + b + " = " + result);
    }

    public static void main(String[] args) {
        MWAdderClient client = new MWAdderClient();
        client.add(40, 7);
    }
}
```

Überblick

Registry für Web-Services

- Registry
  - Treffpunkt zwischen Dienstanbieter und Dienstnehmer
    - Dienstanbieter registriert Web-Service unter einem Namen und/oder einer Reihe von Attributen
    - Dienstnehmer findet Web-Service mittels Suchanfrage nach Namen und/oder Attributen
    - Achtung: Die anschließende Kommunikation zwischen Dienstnehmer und Dienstanbieter findet direkt (d. h. ohne Einbeziehung der Registry) statt
  - Registry-Dienst ist üblicherweise selbst als Web-Service implementiert
- Java API for XML Registries (JAXR)
  - Einheitliche API für den Zugriff auf XML-Registries
  - Unterstützt u. a. Kommunikation mit UDDI-Registries
  - Tutorial: `http://docs.oracle.com/javaee/1.4/tutorial/doc/JAXR.html`
  - API: `http://docs.oracle.com/javaee/6/api/javax/xml/registry/package-summary.html`

Java API for XML Registries (JAXR)

- Package: `javax.xml.registry`
- Zentrale Schnittstellen
  - `Connection`
    - Grundlegende Verbindung zur Registry
    - Authentifizierung gegenüber der Registry
  - `RegistryService`
    - Komponente zum Zugriff auf die Registry
    - Bereitgestellt von der Registry-Verbindung
- Zugriff auf Registry mittels Unterkomponenten von `RegistryService`
  - `BusinessQueryManager`
    - Schnittstelle zum Durchsuchen der Registry
    - Rein lesende Operationen → keine Authentifizierung notwendig
  - `BusinessLifeCycleManager`
    - Schnittstelle zum Erstellen, Ändern und Löschen von Registry-Einträgen
    - Modifizierende Operationen → Authentifizierung notwendig

Verbindungsaufbau zur Registry

```
String registryURL = "http://localhost:12345/juddi";
String queryManagerURL = registryURL + "/inquiry";
String lifeCycleManagerURL = registryURL + "/publish";

// Zusammenstellung der Verbindungsdaten
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
                  queryManagerURL);
props.setProperty("javax.xml.registry.lifeCycleManagerURL",
                  lifeCycleManagerURL);

Connection connection;
RegistryService regSvc;
try {
    // Aufbau der Verbindung
    ConnectionFactory factory = ConnectionFactory.newInstance();
    factory.setProperties(props);
    connection = factory.createConnection();
    regSvc = connection.getRegistryService();
} catch(Exception e) {
    [...] // Fehlerbehandlung
}
```

Authentifizierung gegenüber der Registry

- Beispiel: Setzen von Credentials für einen Nutzer „gruppe0“

```
String user = "gruppe0";
String password = "";

// Credentials erzeugen
PasswordAuthentication pa = new PasswordAuthentication(user,
                                    password.toCharArray());
Set<PasswordAuthentication> credentials =
                        new HashSet<PasswordAuthentication>();
credentials.add(pa);

// Credentials setzen
try {
    Connection connection = [...];
    connection.setCredentials(credentials);
} catch(JAXRException jre) {
    [...] // Fehlerbehandlung
}
```

Verwaltung der Registry-Einträge

- Verwaltung in Form von `RegistryObject`-Objekten
  - Eindeutig identifizierbar über einen Schlüssel (*Key*)
  - Zusätzlich: Vergabe eines Namens (*Name*) möglich
- Kategorien von `RegistryObject`-Objekten
  - `Organization`
    - Informationen über einen Dienstanbieter (z. B. Adresse, Kontaktperson)
    - Hierarchie aus `Organizations` möglich
  - `Service`
    - Informationen über den eigentlichen Dienst (z. B. Name, Beschreibung)
    - Zuordnung zu einer `Organization`
  - `ServiceBinding`
    - Informationen über die Implementierung eines Diensts (z. B. Zugangs-URI)
    - Zuordnung zu einem `Service`

Durchsuchen der Registry

- Anfrage erfolgt durch Methodenaufruf am `BusinessQueryManager`
  - `findOrganizations()`: Suche nach `Organizations`
  - `findServices()`: Suche nach `Services`
  - `findServiceBindings()`: Suche nach `ServiceBindings`
  - ...
- Spezifizierung der Suchparameter
  - Suchkriterien (*Find-Qualifiers*)
    - `Collection` von `Strings`
    - Beispiele: `FindQualifier.{EXACT_NAME_MATCH,SORT_BY_NAME_ASC}`
  - Namenskriterien (*Name-Patterns*)
    - `Collection` von `Strings`
    - Beispiele: „MyService“, „%Service“
- Rückgabe der Antwort per `BulkResponse`-Objekt
  - Kapselung einer `Collection` von `Organizations`, `Services`, ...
  - Kapselung einer `Collection` von Exceptions

Durchsuchen der Registry

- Beispiel: Suche nach `Organizations`, deren Namen mit „g“ beginnen; Ausgabe in aufsteigender alphabetischer Reihenfolge

```
// Erzeugung der Suchkriterien
Collection<String> findQualifiers = new ArrayList<String>();
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);

// Erzeugung der Namenskriterien
Collection<String> namePatterns = new ArrayList<String>();
namePatterns.add("g%");

// Ausfuehrung der Suche
try {
    RegistryService regSvc = [...];
    BusinessQueryManager m = regSvc.getBusinessQueryManager();
    BulkResponse br = m.findOrganizations(findQualifiers,
                      namePatterns, null, null, null, null);
    [...] // Antwort auf Exceptions ueberpruefen
    Collection<Organization> orgs = br.getCollection();
    for(Organization o : orgs) [...] // Auswertung d. Resultate
} catch(Exception e) { [...] }
```

Einschub: Strings in JAXR

- Kapselung in `InternationalString`-Objekten
- Erzeugung durch den `BusinessLifeCycleManager`
- Extraktion des Java-Strings per `getValue()`-Methode
- Beispiel

```
try {
    String s = "Hallo";

    // Einpacken
    BusinessLifeCycleManager lcm = regSvc.getBusinessLifeCycleManager();
    InternationalString is = lcm.createInternationalString(s);

    // Auspacken
    String xs = is.getValue();
    System.out.println(xs);
} catch(JAXRException jre) {
    [...] // Fehlerbehandlung
}
```

Erzeugung von Registry-Einträgen

- Zentrale Aufgabe des `BusinessLifeCycleManager`
- Vorgehen
  1. Erzeugung eines neuen `Organization`-Eintrags
  2. Erzeugung eines neuen `Service`-Eintrags für die `Organization`
  3. Erzeugung eines neuen `ServiceBinding`-Eintrags für den `Service`
- Anmerkungen
  - Falls einer der Einträge bereits existiert, kann der entsprechende Schritt weggelassen werden
  - Das Erzeugen eines Eintrags für eine `Organization`, die bereits existiert, löscht alle `Service`- und `ServiceBinding`-Einträge dieser `Organization`

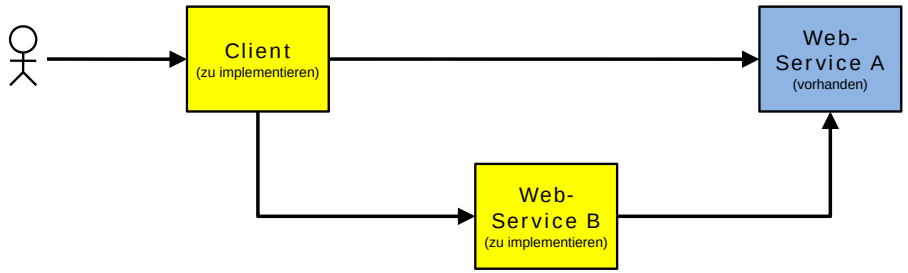
Erzeugung von Registry-Einträgen — Beispiel

```
String on = "MyOrg";
String sn = "MyService";
BusinessLifeCycleManager lcm = [...];

// Organization erzeugen
InternationalString onis = lcm.createInternationalString(on);
organization = lcm.createOrganization(onis);

// Service fuer diese Organization erzeugen
InternationalString snis = lcm.createInternationalString(sn);
Service service = lcm.createService(snis);
organization.addService(service);

// ServiceBinding fuer diesen Service erzeugen
ServiceBinding binding = lcm.createServiceBinding();
[...] // ServiceBinding spezifizieren
service.addServiceBinding(binding);

// Organization in der Registry sichern
Collection<Organization> orgs = new ArrayList<Organization>(1);
orgs.add(organization);
BulkResponse response = lcm.saveOrganizations(orgs);
[...] // Antwort auf Exceptions ueberpruefen
```

Überblick

Aufgabe 1

- Bereitstellung eines eigenen Web-Service
- Teilaufgaben
  - Kommandozeilen-Tool für Registry-Zugriff
  - Web-Service zur Erweiterung eines bereits bestehenden Web-Service
  - Client zum Zugriff auf beide Web-Services



Registry-Zugriff

- Kommandozeilen-Client für den Zugriff auf eine Registry, bei der die URLs zu WSDL-Beschreibungen von Web-Services registriert sind
- Angebotene Kommandos
  - `LIST`: Ausgabe der registrierten WSDL-URLs für einen Dienst
  - `REGISTER`: Registrierung einer WSDL-URL für einen Dienst
- Registry-Zugangsdaten
  - Nutzername: eigener Gruppenname (z. B. „gruppe0“)
  - Passwort: leere Zeichenkette („“)
- Locale-Einstellungen
  - Die Registry unterscheidet zwischen verschiedenen Locale-Einstellungen
  - Die bereitgestellte Registry verwendet „`en_US`“
  - Einstellen der Locale im Konstruktor des Kommandozeilen-Client

```
Locale.setDefault(Locale.US);
```

Web-Services — Facebook-Dienst

- Auf einem Lehrstuhlrechner bereitgestellt
- Dienst zur Verwaltung von Nutzern und ihrer Freundschaftsbeziehungen zu anderen Nutzern
- Verwaltete Information für jeden Nutzer
  - *ID*: eindeutige Kennzeichnung des Nutzers
  - *Name*: (Klar-)Name des Nutzers
  - *Freunde*: Liste mit den IDs von Freunden des Nutzers
- Angebotene Methoden
  - Suche nach Nutzern, deren Namen die Zeichenkette `name` enthält

```
String[] searchIDs(String name);
```

  - Ausgabe des Klarnamens zu einer ID

```
String getName(String id);
```

  - Ausgabe aller Freunde eines Nutzers

```
String[] getFriends(String id);
```

Web-Services — Pfad-Dienst

- Im Rahmen von Aufgabe 1 zu implementieren
- Dienst zur Ermittlung der kürzesten Verbindung zwischen Nutzern
- Angebotene Methode

```
public String[] calculatePath(String startID, String endID);
```

  Ausgabe der kürzesten Verbindung zwischen den Nutzern `startID` und `endID` in Form einer Liste von Nutzer-IDs
- Implementierung
  - Zurückgreifen auf Methoden des Facebook-Diensts
  - Bestimmung des kürzesten Pfads
    - Bereitgestellt: Implementierung des Dijkstra-Algorithmus
    - Zu implementieren: Zusammenstellung der Eingabemenge von IDs
    - Vorgehen: Schrittweise Erweiterung der Freundeskreise von `startID` und `endID` bis diese sich überschneiden
- Achtung: Hilfsmethoden nicht als `public` deklarieren!

Web-Service-Client

- Kommandozeilen-Client für den Zugriff auf **beide** Web-Services
- Angebotene Kommandos
  - `SEARCH`: Suche nach Nutzern
  - `FRIENDS`: Ausgabe der Namen aller Freunde eines Nutzers
  - `PATH`: Kürzester Pfad (Nutzernamen) zwischen zwei Nutzern