Angewandte Softwareentwicklung
Serialisierung

WS 2014/2015

Markus Berg
Hochschule Wismar
Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik

markus.berg@hs-wismar.de
http://mmberg.net

Motivation

- Inhalt von Objekten geht nach dem Beenden eines Programms verloren, da sie nur im Arbeitsspeicher liegen
- Oftmals ist die Speicherung von Daten erwünscht (z.B. Bestenliste in Spielen, etc.)
  - Z.B. als CSV Datei
  - Diese muss aufwändig erzeugt und wieder gelesen werden (um ein leeres Objekt mit den entsprechenden Werten aus der CSV zu belegen)
- Einfacher ist die 1:1 Speicherung des Objektes als Datei
- Serialisierung ist die Speicherung des Zustandes eines Objektes
  - D.h. Speicherinhalt → Datei
  - Auch Marshalling genannt

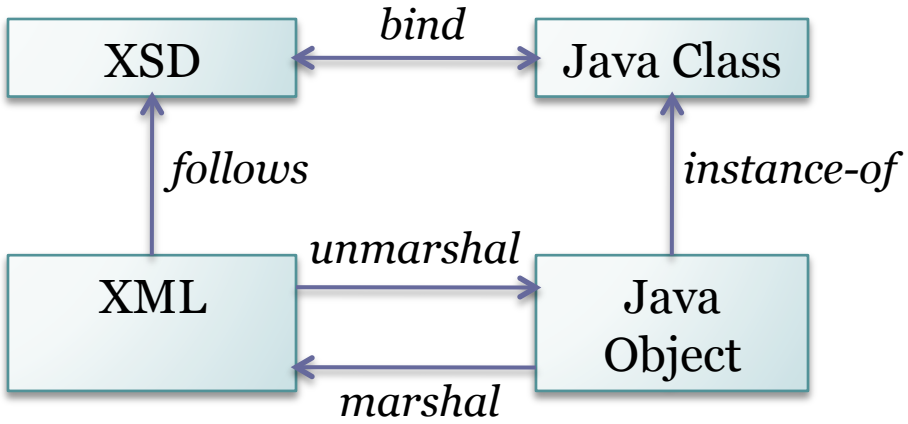
Serialisierung als XML

- Das einfachste ist die Speicherung als Binärdatei
- Alternative ist die Speicherung als XML
  - Lesbar
  - Änderbar
  - Austauschbar
  - Programmiersprachenunabhängig
- Auf Grundlage der Standard-XML-Datentypen (und ComplexTypes)
  - Diese müssen von der jeweiligen Programmiersprache in die eigenen Typen/Klassen konvertiert werden (Unmarshalling)
  - z.B. xsd:string → java.lang.String

JAXB: Java Architecture for XML Binding

- Referenzimplementierung und Teil von Jersey
- Übersetzung zw. XML und Java

- Schritte
  - Binding
  - Marshalling (Java → XML)
  - Unmarshalling (XML → Java)



- Zwei Möglichkeiten:
  - Schema first: Java Klassen aus XML Schema generieren lassen
  - Java first: XML Dokument aus Java Objekt erstellen
    - Vorher Java Klasse erzeugen und annotieren
    - Optional validieren anhand eines Schemas
    - Schema kann auch aus Beans (Java Klassen) generiert werden

JAXB

- Verarbeitet POJOs (Plain Old Java Objects)
  - Keine Abhängigkeiten (Vererbung etc)
  - Empfohlen: Java Beans (Getter und Setter, Argumentloser Konstruktor)

```
public class Person {

    private String name;

    public Person(){
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name=name;
    }
}
```

JAXB

- Basiert auf Annotations (`javax.xml.bind.annotation`)
  - @XmlRootElement
  - @XmlElement
  - @XmlAttribute
- Ohne XmlElement-Annotationen werden alle öffentlichen Methoden und Attribute serialisiert (Default JAXB Binding Rules)
  - Konvention
    - Getter `getName` wird zu XML-Element „name“
    - Element „name“ wird vom Setter `setName` verarbeitet
  - Getter: wichtig beim Marshalling
  - Setter: wichtig beim Unmarshalling
  - Es darf nicht einen öffentlichen Getter `getName` und gleichzeitig ein öffentliches Attribut `name` geben, da beides in XML zu `name` werden würde
- Das Root-Element muss immer gekennzeichnet werden

Beispiel: Serialisierung

```
@XmlRootElement
public class Person {

    private String vorname;
    private String nachname;

    public String getVorname(){
        return vorname;
    }

    public void setVorname(String vorname){
        this.vorname=vorname;
    }

    public String getNachname(){
        return nachname;
    }

    public void setNachname(String nachname){
        this.nachname=nachname;
    }
}
```

```
public static void main(String[] args) {

        Person p = new Person();
        p.setNachname("Bond");
        p.setVorname("James");

        try{
            JAXBContext context =
              JAXBContext.newInstance(Person.class);
            Marshaller m=context.createMarshaller();
            m.marshal(p, System.out);
}
        catch(Exception ex){
            ex.printStackTrace();
        }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <nachname>Bond</nachname>
  <vorname>James</vorname>
</person>
```

Schema aus Beans generieren

- Mit Hilfe des schemagen-Tools im bin-Order des JDK

```
> cd src/net/mmberg/ase/
> ls
Serialisierung.java
Person.java
> schemagen Person.java
Note: Writing src/net/mmberg/ase/schema1.xsd
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>¬
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">¬
¬
  <xs:element name="person" type="person"/>¬
¬
  <xs:complexType name="person">¬
    <xs:sequence>¬
      <xs:element name="nachname" type="xs:string" minOccurs="0"/>¬
      <xs:element name="vorname" type="xs:string" minOccurs="0"/>¬
    </xs:sequence>¬
  </xs:complexType>¬
</xs:schema>¬
```

Klassen aus Schema generieren

- Wir probieren nun den Schema-first-Ansatz
- Tool: XJC (Binding Compiler)
  - Ebenfalls im bin-Ordner des JDK

```
> xjc schema1.xsd
parsing a schema...
compiling a schema...
generated/ObjectFactory.java
generated/Person.java
```

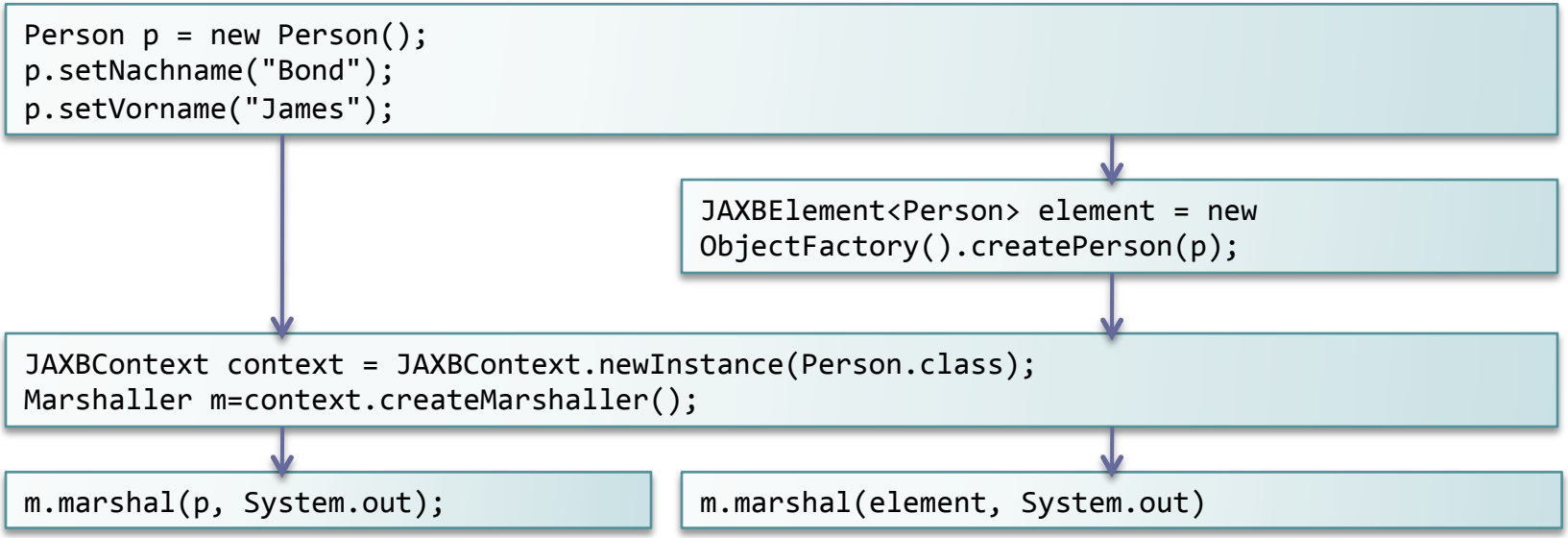
- Optional Package angeben:

```
> xjc -p net.mmberg.ase.generated
schema1.xsd
```

```
public class Person {¬
¬
    protected String nachname;¬
    protected String vorname;¬
¬
    /**¬
     * Gets the value of the nachname property.¬
     * ¬
     * @return¬
     *     possible object is¬
     *     {@link String }¬
     *     ¬
     */¬
    public String getNachname() {¬
        return nachname;¬
    }¬
¬
    /**¬
     * Sets the value of the nachname property.¬
     * ¬
     * @param value¬
     *     allowed object is¬
     *     {@link String }¬
     *     ¬
     */¬
    public void setNachname(String value) {¬
        this.nachname = value;¬
    }¬
¬
```

Instantiieren & Serialisieren

- Wir haben nun Klassen aus dem Schema erzeugt
  - Nun können wieder Objekte erzeugt werden
- Serialisieren schlägt fehl, da Annotation @XmlRootElement fehlt
- Es wurde zusätzlich eine ObjectFactory erzeugt
  - Kapselt das Objekt in einem JAXBElement
  - Diese muss zum Erzeugen der Objekte genutzt werden
  - (Alternativ per Hand die Annotation hinzufügen)



Programmiersprachenunabhängigkeit

- Bis jetzt: innerhalb der Java-Welt
- Ziel: Erzeugen von Klassen in C# anhand des Schemas (das wiederum aus der Java-Klasse erzeugt wurde)

```
> xsd.exe -c schema1.xsd
Microsoft (R) Xml Schemas/DataTypes support utility
Writing file 'C:\...\SerialisierungC\xsd\schema1.cs'.
```

```
public partial class person {

    private string nachnameField;
    private string vornameField;

    /// <remarks/>
    [System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
    public string nachname {
        get {
            return this.nachnameField;
        }
        set {
            this.nachnameField = value;
        }
    }
```

Serialisieren in C#

```
person p = new person();
p.vorname = "James";
p.nachname = "Bond";

XmlSerializer serializer = new XmlSerializer(typeof(person));
serializer.Serialize(Console.Out, p);
Console.ReadKey();
```

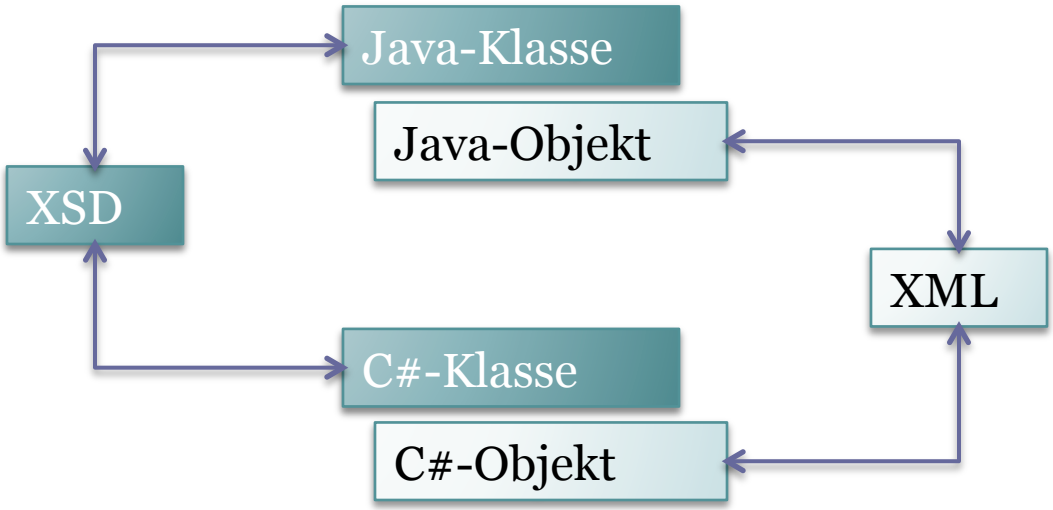


```
<?xml version="1.0" encoding="ibm850"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <nachname>Bond</nachname>
  <vorname>James</vorname>
</person>
```

- Statt auf Konsole kann natürlich auch in eine Datei geschrieben werden, die dann von einer anderen Anwendung wieder gelesen werden kann
- Somit realisieren wir einen Datenaustausch zwischen verschiedenen Programmen

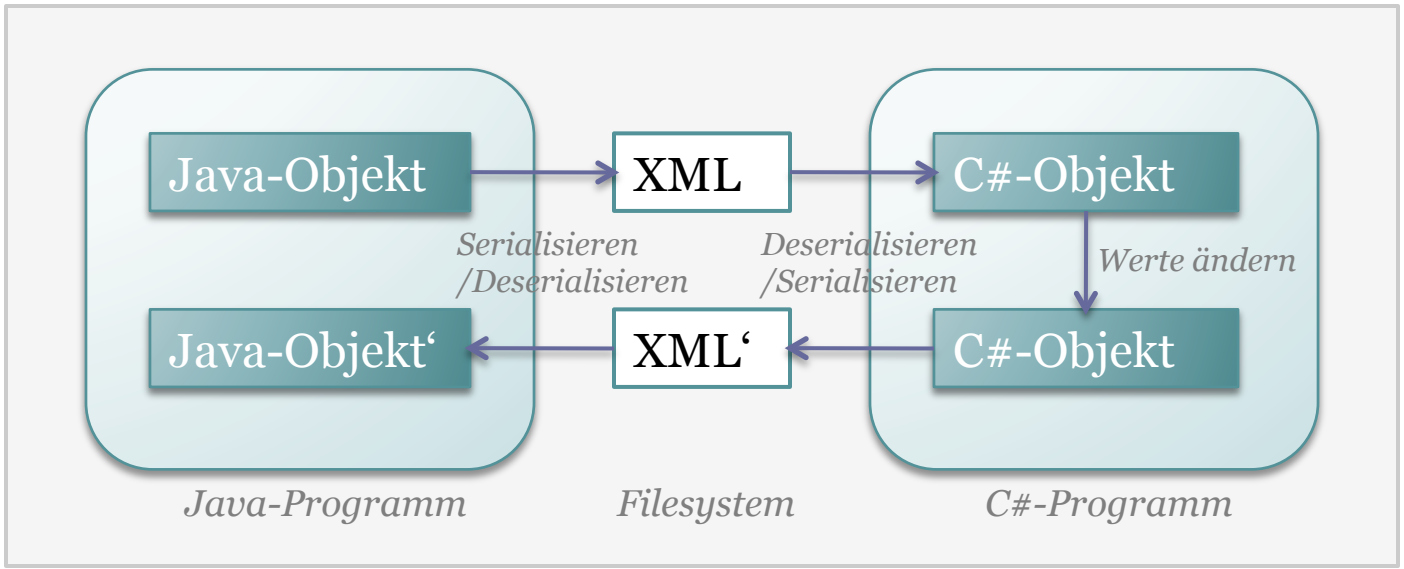
Ablauf

- Nutzung von Binding/Marshalling zwischen verschiedenen Programmiersprachen mit Hilfe eines Schemas



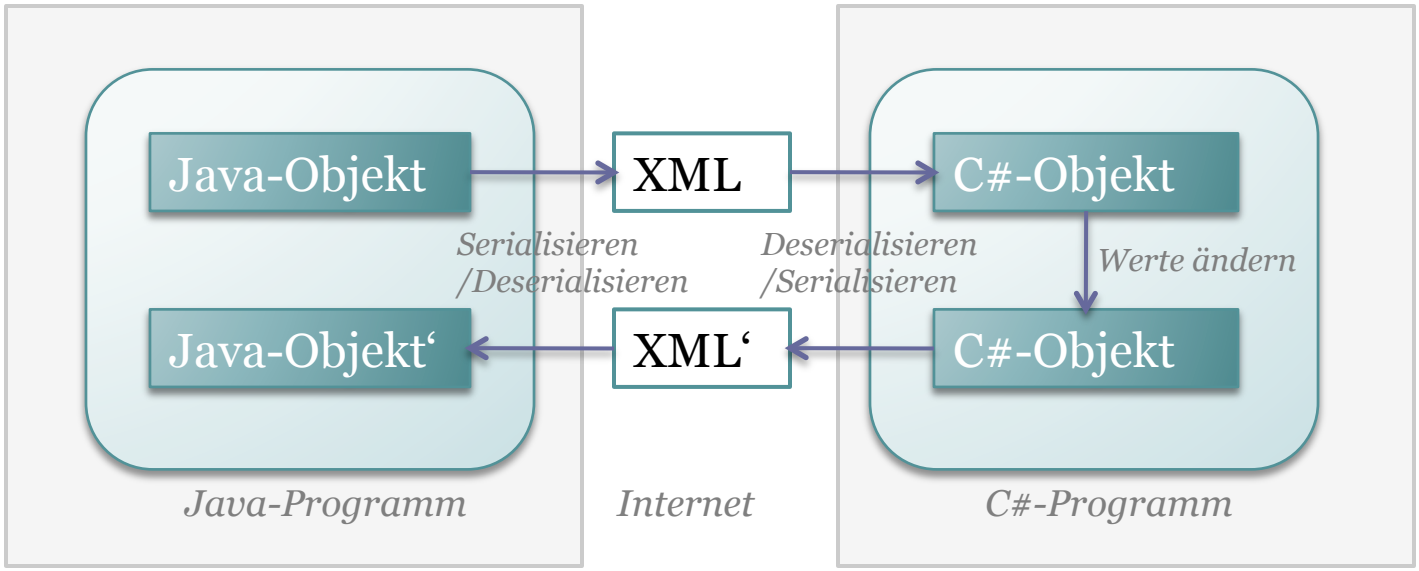
Datenaustausch

- Bis jetzt: Datenaustausch zwischen Programmen auf dem gleichen Rechner (bzw. über Datenträger)
  - Daten mit einem Programm erzeugen und speichern
  - Mit anderem Programm laden und weiterverarbeiten
  - Änderungen dem ersten Programm übergeben



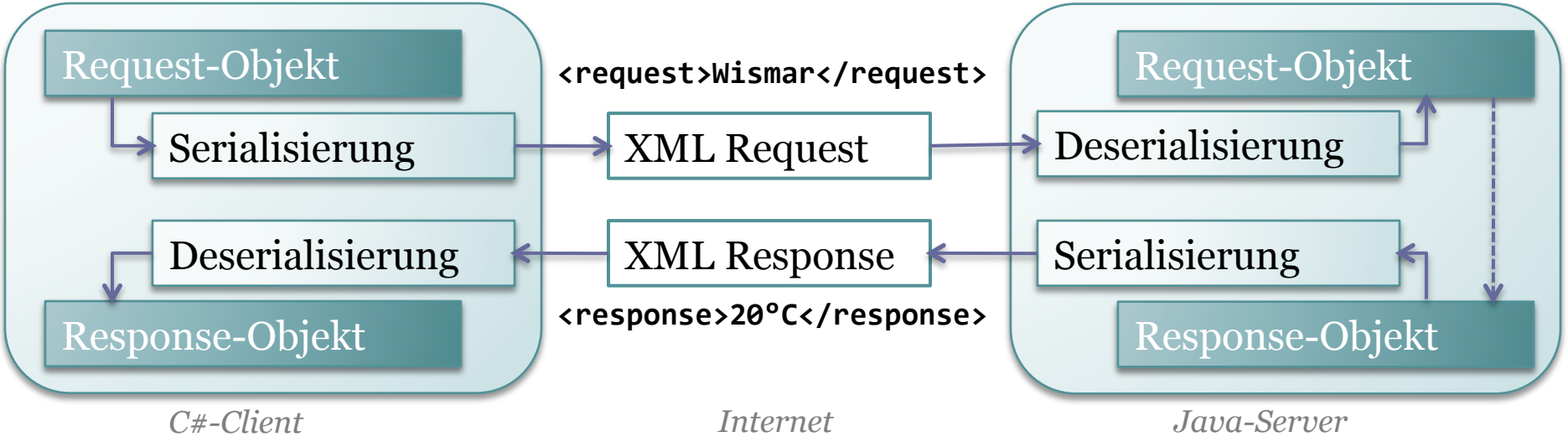
Datenaustausch

- Ziel: Datenaustausch und Kommunikation im Internet



Beispiel

- Server (auf Java-Basis) stellt Wetterinformationen zur Verfügung
- Sie möchten diese in Ihrem Client (auf C#-Basis) nutzen



- Notwendigkeit das Format der Schnittstelle zu beschreiben und verfügbar zu machen

➔ Realisierung mit Web Services

Quellen und weiterführende Literatur

- http://docs.oracle.com/cd/E13222_01/wls/docs103/webserv/data_types.html
- https://jaxb.java.net/2.2.4/docs/schemagen.html