4. **ColdFusion**: Baut ebenfalls auf Erweiterungen von Drittanbietern (z.B. Coldbox [39]).
5. **Perl**: Drittanbieter (z.B. Catalyst [4]).
6. **Ruby**: Fester Bestandteil von Ruby on Rails [26].

Mit der Aufnahme von JavaServer Faces 1.2 in den Java-EE Standard [52] im Jahre 2006 hat Sun Microsystems[8] eine stärkere Modularisierung bei der Entwicklung der Benutzerschnittstelle vorangetrieben. Erstmals bestand eine echte Trennung von Inhalten, deren Repräsentation und der Programmsteuerung.

- *Model*: Das Datenmodell enthält die darzustellenden Daten. Bezogen auf JSF sind das die Attribute eines Managed-Bean, welche in Form von EJB-Komponenten auch die zugrundeliegende Geschäftfslogik enthalten.
- *View*: Die Ansicht ist eine grafische Repräsentation des Datenmodells. Bei JSF entspricht diese dem generierten XHTML-Dokument, welches im Webbrowser beim Client dargestellt wird. Die strukturelle Vorlage der Repräsentation bilden die Facelets.
- *Controller*: Die Programmsteuerung ist für die Ereignisbehandlung zuständig. Sie beobachtet Benutzereingaben und kann daraufhin Aktionen auslösen, die das Datenmodell und die Ansicht verändern. Typische Ereignisse einer JSF-Komponente sind z.B. `valueChange` oder `action` (Schaltfläche betätigen). Genau genommen sind Teile der Programmsteuerung in der Client-Skriptsprache JavaScript implementiert und werden clientseitig ausgeführt. Andere Teile befinden sich im Managed-Bean und werden in der *Invoke Application*-Phase aufgerufen.

Generell ist eine klare Aufteilung der Software-Artefakte in die drei Einheiten *Model*, *View* und *Controller* nicht immer offensichtlich. Insbesondere bei der Geschäftslogik ist sowohl eine Zuordnung zum Datenmodell möglich als auch die Zuordnung zur Programmsteuerung.

2.2. Webservice

Ein Webservice bildet eine Schnittstelle zum Datenaustausch lose gekoppelter heterogener Systeme. Die Kommunikation erfolgt dabei über ein Netzwerk bzw. das Internet. Wie im klassischen Paradigma der *serviceorientierten Architektur* (SOA) [32] verteilter Systeme gibt es auch beim Webservice die Rollen des Konsumenten und des Dienstleisters.[9] Letzterer realisiert Dienstzugangspunkte (end points) in Form von Netzwerkadressen, welche durch URIs[10] spezifiziert werden, über die potentielle

[8] heute Oracle Corporation
[9] Auch Diensterbringer und Dienstnehmer genannt [24].
[10] Uniform Resource Identifier [1]

Konsumenten auf den Dienst zugreifen können.

Das *World Wide Web Consortium* (W3C) beschreibt einen Service als eine abstrakte Ressource:

> „A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent.“[25]

Die wichtigsten Protokolle und Datenaustauschformate für Webservices sollen im folgenden vorgestellt werden.

2.2.1. SOAP und WSDL

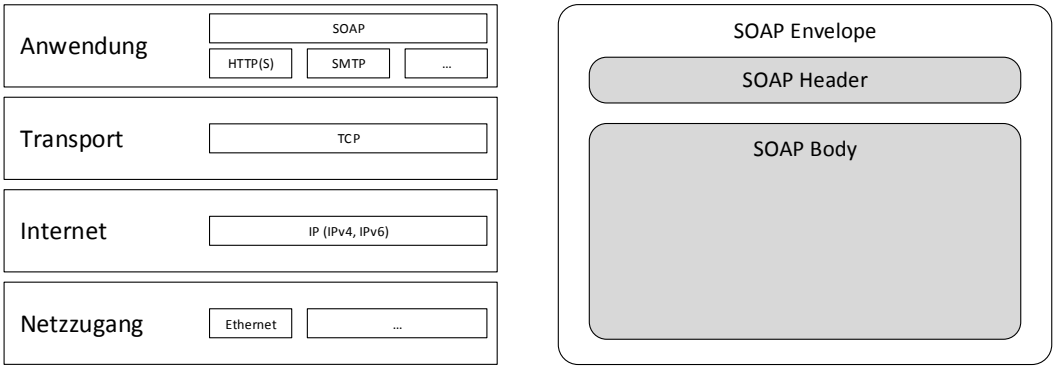
SOAP[11] ist ein Netzwerkprotokoll zum Austausch XML[12]-basierter Nachrichten und gehört seit 2003 in Version 1.2 als industrieller Standard dem *World Wide Web Consortium* (W3C) an [22]. Historisch ging es 1999 aus der ein Jahr zuvor von Microsoft entwickelten Spezifikation XML-RPC[13] für entfernte Methodenaufrufe hervor. Wenig später wurde die Weiterentwicklung der Spezifikation von IBM und anderen Unternehmen unterstützt und schließlich beim W3C eingereicht, wo die bis heute gültige und letzte Version 1.2 erarbeitet wurde. Im Kontext des TCP/IP-Protokollstapels befindet sich SOAP auf Ebene der Anwendungsschicht (siehe Abbildung 2.4a). Theoretisch können zur Übertragung von SOAP-Nachrichten beliebige Protokolle der darunterliegenden Schichten und der Anwendungsschicht selbst benutzt werden. Die Spezifikation sieht für den Standard-Fall jedoch das wohlbekannte und bewährte Übertragungsprotokoll HTTP in Kombination mit TCP/IP vor.

Die Schnittstellenbeschreibung bei SOAP-Webservices übernimmt die *Web Services Description Language* (WSDL), welche ebenfalls auf XML basiert und als sogenannte W3C-Empfehlung veröffentlicht wurde [65]. Ein SOAP-Webservice ist demzufolge selbstbeschreibend, da ein potentieller Konsument (Client) die verfügbaren Funktionen des Webservice via WSDL abfragen kann. Auf weitere Details soll hier nicht eingegangen werden.

Wie in Abbildung 2.4b zu sehen, bestehen SOAP-Nachrichten aus drei Hauptelementen *Envelope*, *Header* und *Body*.

- Das Envelope-Element ist das äußere Element jeder SOAP-Nachricht und beinhaltet die folgenden zwei spezifischen Kindelemente.
- Das Header-Element ist optional und kann anwendungsspezifische

[11] SOAP ist ein Markenname
[12] Extensible Markup Language
[13] XML Remote Procedure Call



(a) SOAP im TCP/IP-Protokollstapel (b) Aufbau einer SOAP-Nachricht

Abbildung 2.4.: SOAP

Meta-Informationen enthalten, z.B. zur Kodierung, Verschlüsselung, Weiterleitung der Nachricht oder zur Transaktionsverwaltung.

- Das Body-Element ist verpflichtend und enthält die eigentlichen Nutzdaten, die auch in XML kodiert sind.

SOAP ist ein Rahmenwerk mit vielseitigen Möglichkeiten zum Datenaustausch über ein Netzwerk. Durch sogenannte *SOAP Protocol Bindings* ist es möglich andere Netzwerkprotokolle, wie z.B. SMTP[14] oder JMS[15] zu benutzen. Wie eingangs erwähnt ist HTTP über TCP/IP das von der Spezifikation bevorzugte Protocol Binding für SOAP. Ein wichtiger Grund hierfür ist die Tatsache, dass bestehende Infrastrukturen von Rechnernetzen SOAP-Nachrichten ohne spezielle Anpassungen problemlos durch Firewalls und Proxyserver hindurch zum Kommunikationspartner weiterleiten. Es soll nicht unerwähnt bleiben, dass SOAP-Nachrichten sehr wortreich sind, was hauptsächlich auf ihre Beschaffenheit, dem XML-Format, zurückzuführen ist. Das kann einen klaren Nachteil gegenüber eng gekoppelten Systemen mit binärem Datenaustauschformat bezüglich der Nachrichtengröße und Verarbeitungsgeschwindigkeit darstellen, was sich letztlich nachteilig auf die Performance auswirkt. Daher sieht die Spezifikation in Abschnitt 5.3 unter dem Namen *Message Transmission Optimization* ein Verfahren vor Binärdaten direkt in die XML-Nachricht einzubetten. Dennoch sollte für jeden Anwendungsfall spezifisch abgewägt werden, ob die mit SOAP verbundenen Performance-Einbußen die Vorteile egalisieren (siehe dazu später Abschnitt X Architekturentscheidung pro JSON). Zu den positiven Aspekten gehört die Eigenschaft des für Menschen lesbaren Formates der Nachrichten sowie die damit verbundene einfache Wartung und Erweiterbarkeit von Aufbau und Struktur. Damit geht eine geringere Fehleranfälligkeit einher. Mit WSDL existiert außerdem eine klar definierte

[14]Simple Mail Transfer Protocol
[15]Java Message Service

Sprache zur konkreten Beschreibung der Schnittstelle.

2.2.2. REST

REST steht für *Representational State Transfer* und bezeichnet ein Softwarearchitekturmodell für Ressourcenzugriff bei verteilten Systemen. Die Bezeichnung wurde 2000 von Roy Fielding[16] im Rahmen seiner Doktorarbeit eingeführt und definiert und ist heute das vorherrschende Programmierparadigma für Web-APIs [18]. Ein sogenanter *RESTful Web Service* basiert auf HTTP und bildet gewissermaßen eine Untermenge des World Wide Web [64] indem bestimmte Beschränkungen eingehalten und Bedingungen erfüllt sein müssen.

Folgende fünf Eigenschaften sind essentiell für einen RESTful Web Service:

1. **Client-Server** Dieser Eigenschaft liegt das Prinzip *Separation of Concerns* (SoC) zu Grunde. Ein *Server* bietet eine Menge von Diensten an (Dienstleister) und wartet auf Anfragen an diese Dienste. Ein *Client* möchte einen Dienst in Anspruch nehmen (Konsument) und schickt eine Anfrage an den entsprechenden Server. Wie bereits eingangs zu diesem Kapitel erwähnt, erfolgt eine Anfrage an den Server über bestimmte Netzwerkadressen, die durch Uniform Resource Identifier (kurz URI) repräsentiert werden. Diese Zeichenkette bildet den Dienstzugangspunkt des Webservice. Wenn der Server eine Anfrage registriert, kann er entscheiden, ob sie bearbeitet oder abgelehnt wird.

2. **Stateless** Die Kommunikation zwischen Client und Server ist zustandslos. Das bedeutet, dass jede Anfrage eines Client an den Server alle Informationen enthält, die zum Bearbeiten der Anfrage benötigt werden. Der Server speichert keine Session-Informationen oder ähnliches zwischen zwei Anfragen. Diese Eigenschaft erleichtert die Implementierung eines Webservice und erhöht dessen Skalierbarkeit, da keine Logik realisiert werden muss, die Ressourcen bezüglich ihres Anfragekontextes verwalten muss.

3. **Cache** Wie im World Wide Web üblich muss jede Ressource implizit oder explizit ihre Gültigkeitsdauer definieren. Dadurch ist es möglich die Kommunikation zwischen Client und Server zu reduzieren, da die übermittelten Daten clientseitig zwischengespeichert werden können. Diese Eigenschaft wirkt sich positiv auf Skalierbarkeit und Leistungsverhalten der beteiligten Systeme aus.

4. **Uniform Interface** REST beschreibt eine einheitliche Schnittstelle für die Kommunikation zwischen Client und Server. Im wesentlichen sind das die Befehle create, retrieve, update und delete (CRUD) zum Erzeugen, Abrufen, Aktualisieren und Löschen einer Ressource. Im Kontext von HTTP sind das in der Regel die Anweisungen POST, GET, PUT und DELETE. Dies kann vom Service allerdings

[16] Roy Fielding war außerdem ein bedeutender Mitautor der HTTP-Spezifikation. [17]

auch auf andere Weise definiert werden.

5. **Layered System** Ein Web Service kann aus mehreren Schichten aufgebaut sein. Dies kann zwecks Lastenverteilung notwendig sein um einer hohe Anzahl von Anfragen gerecht zu werden. Die Systemkomplexität aus Sicht des Clients bleibt dennoch konstant, da es pro Dienst nur eine Netzwerkadresse gibt und kein Wissen über mögliche Zwischenknoten. Letztere können zum einen die Skalierbarkeit des Systems verbessern, indem Anfragen an andere Server oder Shared Caches weitergeleitet werden, zum anderen können Sie dazu dienen Sicherheitsrichtlinien umzusetzen.

Das REST Architekturmodell selbst ist protokollunabhängig, wird im Kontext des World Wide Web jedoch mit Hilfe von HTTP umgesetzt. Die Schnittstelle eines solchen RESTful Web Service wird durch vier Aspekte charakterisiert:

- Jede Ressource wird durch ihren eindeutigen Bezeichner identifiziert. Beispielsweise `http://example.com/resources/example`. Der Begriff *Ressource* spielt bei REST eine Schlüsselrolle und ist als konzeptionelle Abbildung der Zeit auf eine Menge von Informationsobjekten zu sehen. Die Menge der Informationsobjekte einer Ressource kann im Verlauf der Zeit variieren, jedoch auch statischer Natur sein und immer durch die gleiche Menge repräsentiert werden.
- Das Datenformat der Repräsentationen wird durch einen *Internet Media Type* spezifiziert. Gängige Formate sind HTML, XML oder JSON[17]. Außerdem soll eine Repräsentation genug Informationen mitliefern um die dazugehörige Ressource theoretisch modifizieren zu können - vorausgesetzt der Client verfügt über entsprechende Berechtigungen.
- Die Nachrichten sollen selbstbeschreibend sein. Jede Nachricht enthält genug Informationen darüber, wie sie zu verarbeiten ist. Der Internet Media Type gibt beispielsweise das Datenformat an. Weitere Meta-Informationen im HTTP-Header der Nachricht geben Aufschluss über die Gültigkeitsdauer der Informationen.
- „Übermedien als treibende Kraft des Anwendungszustands“. Nach REST können Repräsentationen nicht nur einfache Daten enthalten, sondern auch Verknüpfungen zueinander haben, die es ermöglichen ausgehend von einer Ressource zu weiteren zu navigieren. Solche Repräsentationen werden als Übermedien (oder Hypermedia) bezeichnet. Mit Hilfe einer solchen Vernetzung wird dem Client die Möglichkeit gegeben, weitere Ressourcen zu entdecken und somit den Anwendungszustand des Clients zu ändern.

REST ist nicht als Gegensatz zu SOAP zu verstehen. Während es sich bei SOAP um ein voll entwickeltes Netzwerkprotokoll handelt, ist REST ein Architekturmodell bzw. eine Sammlung von Designkriterien für Webservices. Genau genommen, kann SOAP auf REST-konforme Art und Weise benutzt werden. Allerdings ist SOAP nicht immer

[17] JavaScript Object Notation

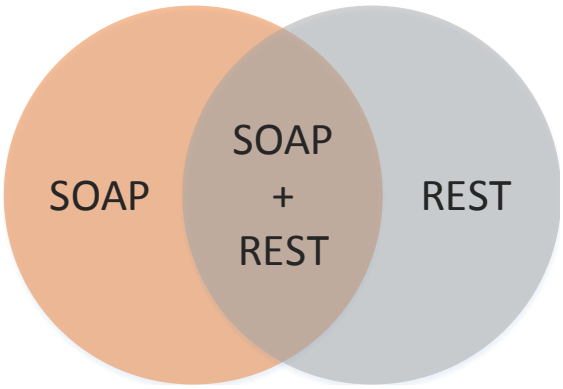


Abbildung 2.5.: Venn-Diagramm zur Beziehung zwischen SOAP und REST

automatisch REST-konform (Abbildung 2.5).

> „It should be noted that SOAP 1.2 can be used in a manner consistent with REST. However, SOAP 1.2 can also be used in a manner that is not consistent with REST.“[2]

2.2.3. Datenaustauschformate

Wie bereits in Abschnitt 2.2.1 erwähnt, basieren SOAP-Nachrichten auf der Auszeichnungssprache Extensible Markup Language (XML). Sie eignet sich sehr gut für die Darstellung hierarchisch strukturierter Daten und ist gleichzeitig menschenlesbar. Die erste Veröffentlichung der Spezifikation erfolgte 1998 durch das W3C und hat seither einige kleinere Aktualisierungen erfahren.

Heutige Webservices setzen allerdings vermehrt auf ein anderes Format: *JavaScript Object Notation* (kurz JSON) ([9] [56] [55]). JSON ist ebenfalls für die Darstellung hierarchisch strukturierter Daten konzipiert und wurde für den Zweck der plattformübergreifenden Datenübertragung entworfen. Die Spezifikation wurde 2002 von Douglas Crockford auf http://json.org veröffentlicht [5]. Mit zunehmender Akzeptanz, unter anderem durch Yahoo! [66] und Google [21], wurde `application/json` im Jahr 2006 schließlich als Internet Media Type offiziell vorgeschlagen [6]. Obwohl es technisch gesehen eine Untermenge der Programmiersprache JavaScript ist, existieren Parser für alle wichtigen Programmiersprachen.

Das JSON-Format besteht aus zwei Struktureinheiten: **Objekten** und **Listen**.

- Ein Objekt ist eine Menge von Name/Wert-Paaren.
- Eine Liste ist eine geordnete Liste von Werten.

Dabei können **Werte** selbst wieder Objekte oder Listen sein oder den Wert einer

vordefinierten Menge primitiver Datentypen annehmen. Mit diesem einfachen Konzept lassen sich beliebig tief verschachtelte Datenstrukturen kodieren.

Sei folgende einfache Datenklasse zur Erfassung der Eckdaten einer Zeitschriftenpublikation gegeben:

```
1  public class Publication {
2      private List<Author> authorList;
3      private Date date;
4      private String journal;
5      private Integer volume;
6      private Boolean listed;
7
8      // Konstruktor sowie Getter und Setter ...
9  }
```

Quelltext 2.1: Publication.java

Die folgenden zwei Codelistings zeigen exemplarisch wie das fiktive Objekt „MyPublication“ als Instanz dieser Klasse einmal mit XML und ein zweites mal mit JSON serialisiert wurde. Hierbei handelt es sich jeweils um eine von vielen möglichen Repräsentationen im entsprechenden Format. Insbesondere beim Attribut `private Date date` sind viele verschiedene Darstellungsformen denkbar.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <title>Myths and lies about effective workout</title>
3  <authorList>
4    <firstName>John</firstName>
5    <lastName>Smith</lastName>
6  </authorList>
7  <authorList>
8    <firstName>James</firstName>
9    <lastName>Johnson</lastName>
10 </authorList>
11 <authorList>
12   <firstName>Robert</firstName>
13   <lastName>Williams</lastName>
14 </authorList>
15 <date>07-04-2013</date>
16 <journal>Fitness Insights</journal>
17 <volume>5</volume>
18 <listed>true</listed>
```

Quelltext 2.2: MyPublication.xml

```
1  {
2    "title": "Myths and lies about effective workout",
3    "authorList": [
4      {"firstName": "John", "lastName": "Smith"},
5      {"firstName": "James", "lastName": "Johnson"},
```

```
6       {"firstName": "Robert", "lastName": "Williams"}
7     ],
8     "date": "07-04-2013",
9     "journal": "Fitness Insights",
10    "volume": 5,
11    "listed": true
12  }
```

Quelltext 2.3: MyPublication.json

Beide Formate sind text-basiert und menschenlesbar. Die Repräsentation durch JSON fällt durch Ihre Kompaktheit auf. Die Darstellung des Attributs *authorList* verbraucht bei XML viel mehr Zeichen als im JSON-Format. Für die Serialisierung von einfachen Datenklassen sind beide Formate hinreichend ausdrucksstark. Wechselbeziehungen zwischen mehreren Objekte gehen allerdings üblicherweise verloren.