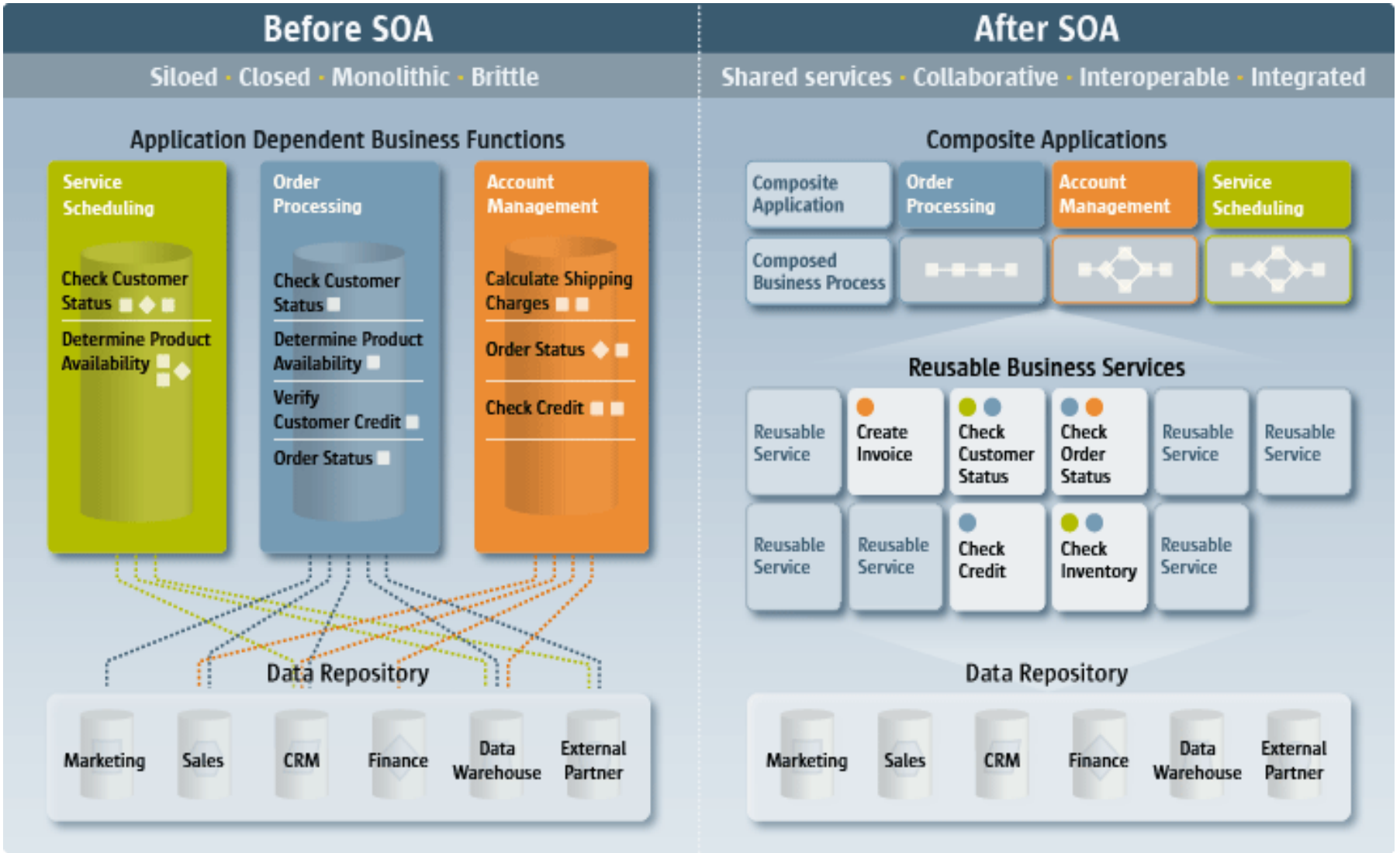
Marc Hüffmeyer

SOAP und REST

Ein Vergleich von service- und ressourcenorientierten Architekturen und deren Einsatz im VMA-Projekt

Überblick

- **Einleitung**
  - Serviceorientierte Architekturen
  - Ressourcenorientierte Architekturen
  - Anforderungen im VMA-Projekt
- **SOAP**
  - Aufbau von SOAP-Nachrichten
  - Beschreibung von SOAP-basierten Web Services
- **REST**
  - Ressourcen und Repräsentationen
  - REST und HTTP
  - URI-Design
  - Beschreibung von RESTful Services
- **Vor- und Nachteile von SOAP und REST**
- **SOAP und REST am Beispiel**
  - Location Service
  - Picture Service
- **Unterstützung in Mobile Clients**
- **Services im VMA-Projekt**
  - Entscheidung für REST
  - Beispielimplementierung für Android


Before SOA
Siloed · Closed · Monolithic · Brittle
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
After SOA
Shared services · Collaborative · Interoperable · Integrated
Composite Applications
Composite Application
Order Processing
Account Management
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

- **Implementieren von verteilten Anwendungen**
  - Gleiche Ziele wie SOA
    - Vermeidung von Redundanz
    - Zentralisieren von Informationen
    - Flexibilität erhöhen
  - Identifikation von Ressourcen statt Services
  - Abbilden der Anwendungsfunktionalität auf Ressourcen
- **Merkmale einer ROA**
  - Addressierbarkeit
  - Einheitliche Schnittstelle
  - Zustandslosigkeit
- **Kein Verzeichnisdienst**
  - Ressourcen verweisen auf Ressourcen

- **Unterscheidung zum „klassischen“ Web Service**
  - Viele Dienstnutzer und viele Dienstanbieter
  - Nicht ein Dienst an zentraler Stelle, sondern „überall“ die gleichen Dienste
  - Kein Verzeichnisdienst
- **Einfachheit der Dienste**
  - Einfachheit bei der Nutzung der Dienste
  - Einfachheit beim Implementieren neuer Dienste

- **Simple Object Access Protocol**
- **Kommunikationsprotokoll zum Austausch von Daten zwischen Systemen**
- **Entwickelt vom World Wide Web Consortium (W3C)**
- **Aktuellster Standard SOAP 1.2**
- **Verwendet wird überwiegend SOAP 1.1**
  - Ist kein Standard, sondern hat den Status eines Note
- **Basiert auf verschiedenen anderen Standards**
  - XML
  - HTTP

- **Envelope**
  - Umschlag der die SOAP-Nachricht beinhaltet
  - Wurzel des XML-Dokuments
  - Muss auf gültige SOAP-Version referenzieren
- **Body**
  - Beinhaltet die auszutauschenden Daten

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
    </soap:Body>
</soap:Envelope>
```

- **Header**
  - Applikationsspezifische Daten, bspw. zur Authentifikation
  - Optional
  - Muss nicht von Zwischenstation verstanden werden
    - Ausnahme: mustUnstand-Attribut

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header xmlns:ns="http://www.example.org/soap">
        <ns:myelement soap:mustUnderstand="1">verstehe</ns:myelement>
    </soap:Header>
    <soap:Body>
    </soap:Body>
</soap:Envelope>
```

- **Fault**
  - Dient der Meldung von Fehlern
  - Kindelement des Body

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Server</faultcode>
            <faultstring>java.lang.IllegalArgumentException</faultstring>
            <detail>
                <jax:exception xmlns:jax="http://jax-ws.dev.java.net/">
                    <jax:stackTrace>
                        …
                    </jax:stackTrace>
                </jax:exception>
            </detail>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

- **Web Service Description Language**
  - Ebenfalls vom W3C entwickelt
  - Primär zur Beschreibung von Web Services
  - Sekundär zur Lokalisierung von Web Services
  - XML-Dokument
    - Abstrakte Definitionen
      - Verwendete XML-Schemata
      - Aufbau der Nachrichten
      - Zusammengehörige Nachrichten
    - Konkrete Definitionen
      - Binding
        - Transportprotokoll
        - Style: RPC oder Document
      - Lokalisierung des Service

- **Respresentational State Transfer**
- **REST ist kein Standard sondern Architekturstil**
- **Dissertation Thomas Roy Fielding**
- **Basiert auf verschiedenen Standards**
  - URI
  - HTTP
  - XML

- **Ressource**
  - Eindeutig identifizierbares Objekt (bspw. eine Person)
  - In ressourcenorientierter Architektur immer adressierbar
  - Kommunikationsgegenstand in verteilter Anwendung
  - Wird vom Client angefordert
- **Repräsentation**
  - Darstellung eines Objekts (Bild, Personalausweis, HTML-Dokument, …)
  - Wird vom Server geliefert

- **Ressourcendesign**
  - Übergang zwischen Ressource und Repräsentation fließend
  - Entscheidung liegt beim Entwickler
  - Tendenziell mehr Ressourcen, da diese eindeutig ansprechbar

- **Ressourcenidentifikation**
  - Im Web einheitlicher Standard: Uniform Resource Identifier
  - Eine URI adressiert eine Ressource, Umkehrschluss gilt nicht

- **REST ist eng verknüpft mit HTTP**
  - Operationen werden durch HTTP-Standard festgelegt
    - GET
    - POST
    - PUT
    - DELETE
  - Lösen vom RPC-Gedanken, Vergleich mit SQL
    - SELECT
    - INSERT
    - UPDATE
    - DELETE
- **HTTP ermöglicht Angabe bevorzugter Repräsentation**

```
GET /ressourcen/1234 HTTP/1.1
Accept: text/html, image/jpeg, */*
Host: www.example.org
```

- **URI-Design**
  - Leitfaden
    - Substantive
    - Keine Verben
  - Beispiele
    - http://www.example.org/findPerson?id=1234
    - http://www.example.org/person/1234
- **Falsches URI-Design stellt große Gefahr dar**
  - Abbilden von Funktionen kann großen Schaden anrichten
  - Operationen werden durch HTTP-Methoden bestimmt!
  - Beispiel
    - http://www.example.org/deletePerson?id=1234
      - Mensch erkennt Kontext
      - Maschine geht von gefahrlosem GET aus (Idempotenz)

- **Web Application Description Language**
  - Entwickelt von SUN
  - XML-Dokument
    - Ressourcen
    - Erlaubte HTTP-Methoden
    - Requests und Responses
- **Beschreibung von RESTful Services nicht so bedeutent wie Beschreibung von SOAP-basierten Web Services**

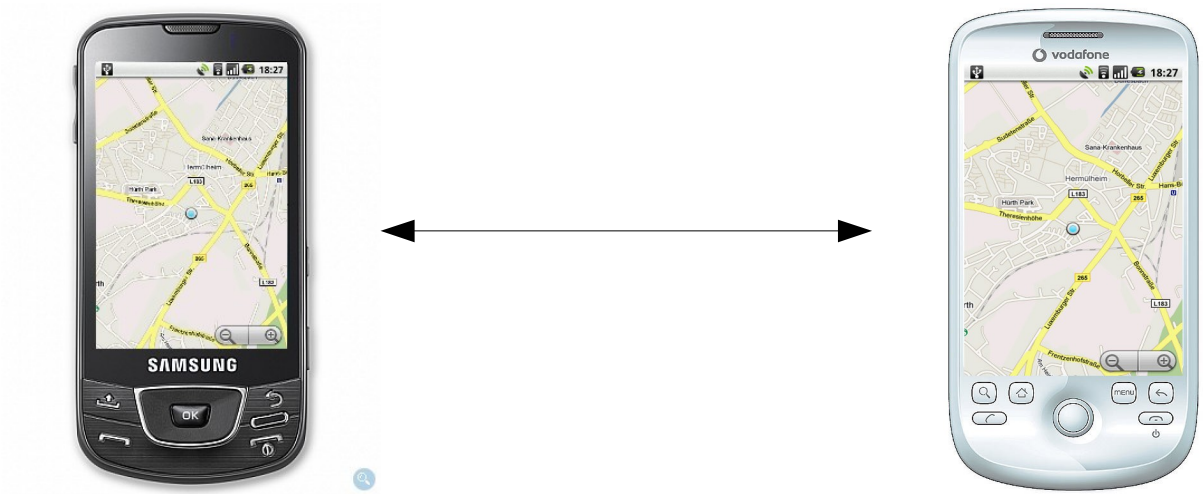
Vor- und Nachteile von SOAP und REST

- **Vorteile SOAP**
  - Intuitive Umsetzung von Services
  - Große Freiheit bei der Umsetzung
- **Nachteile SOAP**
  - Nur „zugeschnittene“ Clients können Request/Responses verarbeiten
  - Zusätzlicher Overhead durch XML

Vor- und Nachteile von SOAP und REST

- **Vorteile REST**
  - Unabhängigkeit der Clients
    - Lediglich die Unterstützung von HTTP muss gewährleistet sein
  - Skalierbarkeit der Dienste
- **Nachteile REST**
  - Fehlende Standardisierung
  - REST oft falsch verstanden
  - Schwierige Abbildung der Anwendungsfunktionalität auf Ressourcen

- **Location Service**
  - Abfrage und Darstellung des Standorts eines mobilen Endgeräts
  - Sowohl mit SOAP als auch mit REST leicht umsetzbar



SOAP: Request

```
POST /service/ HTTP/1.1
Host: www.example.org
Content Type: application/xml
Content Length: 156

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header/>
   <S:Body xmlns:ns="http://www.example.org/">
      <ns:requestLocation/>
   </S:Body>
</S:Envelope>
```

SOAP: Response

```
HTTP/1.1 200 OK
Content Type: application/xml
Content Length: 270

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns: requestLocationResponse xmlns:ns="http://www.example.org/">
         <location>
            <latitude>1</latitude>
            <longitude>2</longitude>
         </location>
      </ns: requestLocationResponse>
   </S:Body>
</S:Envelope>
```

REST: Request

```
GET /service/location HTTP/1.1
Host: 192.168.1.254
Accept: application/xml; q=0.9, text/html; q=0.1
```

REST: Response

```
HTTP/1.1 200 OK
Content Type: application/xml
Content Length: 73

<location>
    <latitude>1</latitude>
    <longitude>2</longitude>
</location>
```

- **Picture Service**
  - Remote Camera
  - Basisdienst analog zu Location Service
  - Erweiterungen:
    - Angabe Auflösung
    - Angabe Format
    - …

- **Erweiterungen unter SOAP**
  - Abbildung der Auflösung/des Formats auf Übergabeparameter
  - „Überladen“ der aufgerufenen Funktion
    - Services müssen parallel existieren um Kompatiblität zu gewährleisten
- **Erweiterungen unter REST**
  - Echte Erweiterung des Basisdienst
  - Übergabe der Parameter
    - Filter (Query-Parameter):
      - `http://192.168.1.254/service/picture/?h=480&w=320`
    - Kamerakonfiguration als eigene Ressource
      - `http://192.168.1.254/service/picture/{config_id}`

Unterstützung in Mobile Clients

- **HTTP**
  - Auf praktisch allem Mobilfunkplattformen wird HTTP unterstützt
  - „Nur“ HTTP über TCP, jedoch muss auch HTTP über UDP möglich sein
- **SOAP**
  - Vereinzelte Unterstützung
    - Symbian: ja
    - Android: nein
    - IPhone: nein
  - Viele kleinere Projekte für verschiedene Plattformen
    - Nur als Client
    - HTTP über TCP
- **REST**
  - Unterstützung von Haus aus nicht gegeben
  - Lediglich Unterstützung von HTTP benötigt
    - Diese aber sehr umfangreich

- **„Freiheit“ bei der Wahl der Clients**
  - SOAP: ausschließlich zugeschnittene Clients
  - REST: potenzielle Clients zeichnen sich müssen lediglich HTTP unterstützen (Browser)
- **Skalierbarkeit der Dienste**
- **Komplexität der Umsetzung**
- **Overhead**
  - SOAP: HTTP + SOAP
  - REST: HTTP

```
@GET
@Produces("application/json")
public void answerGETwithJSON()
{
    JSONObject json = new JSONObject();
    try
    {
        json.put("latitude", location.getLatitude());
        json.put("longitude", location.getLongitude());
    }
    catch (JSONException e)
    {
        // Exception Handling
    }
    byte[] jsonBytes = json.toString().getBytes();
    httpResponse = new HTTPResponse(HTTPResponse.VERSION_1_1,
                                    HTTPResponse.STATUS_200_OK);
    httpResponse.addHeader(HTTPResponse.HEADER_ENTITY_CONTENT_TYPE,
                           "application/json");
    httpResponse.addHeader(HTTPResponse.HEADER_ENTITY_CONTENT_LENGTH,
                           jsonBytes.length + "");
    httpResponse.setBody(jsonBytes);
}
```

Fragen?
Kommentare?
Anmerkungen?

Vielen Dank für Ihre Aufmersamkeit!