INFORMATIONSINTEGRATION UND WEBPORTALE

Vorlesung # 8, 08.12.2014 : Dienstorientierte Integration von Komponenten

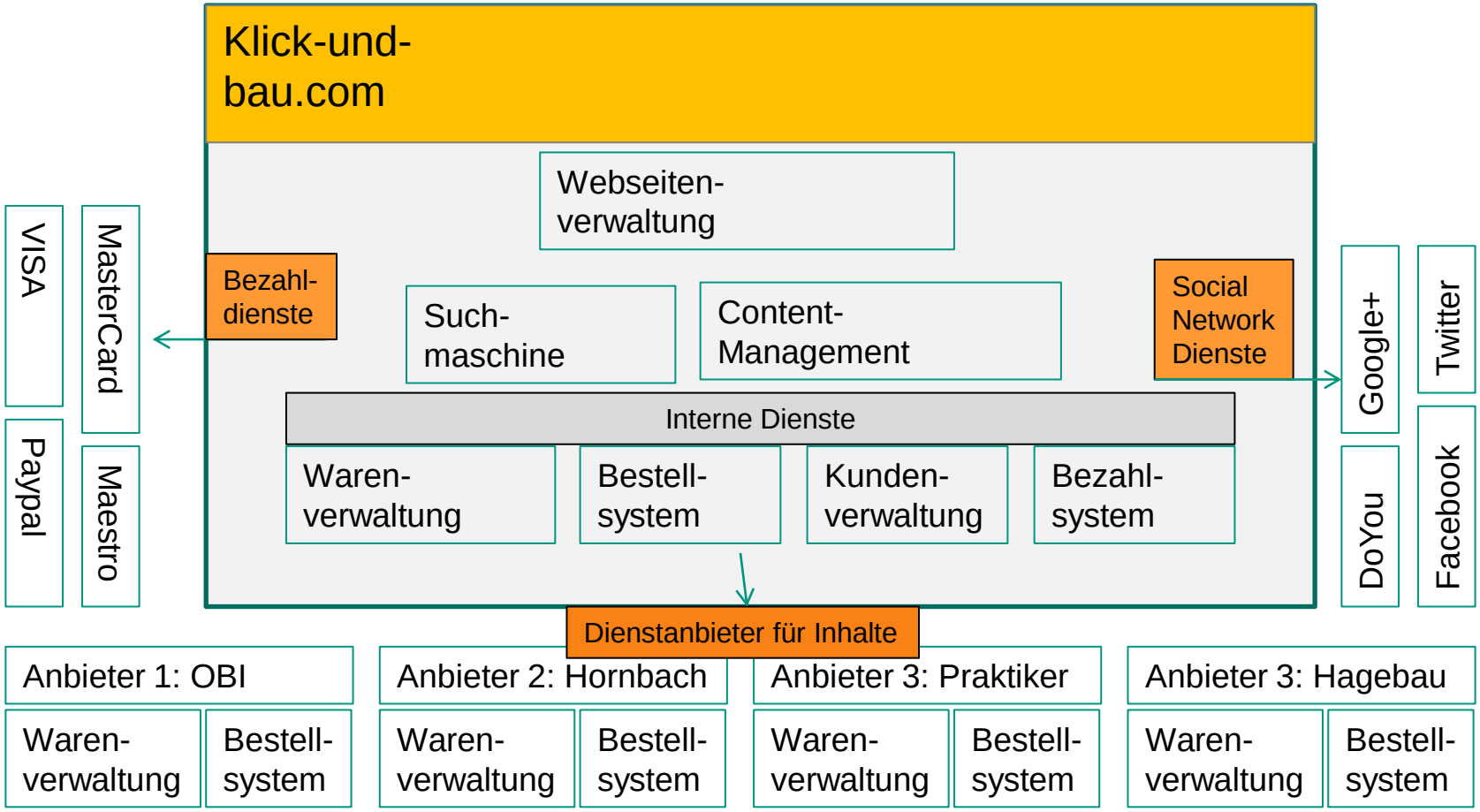
Dr. Andreas Walter

Klick-And-Bau

Informationsintegration und Webportale

Motivation

Komponentensicht auf klick-und-bau: Komponenten als Dienste

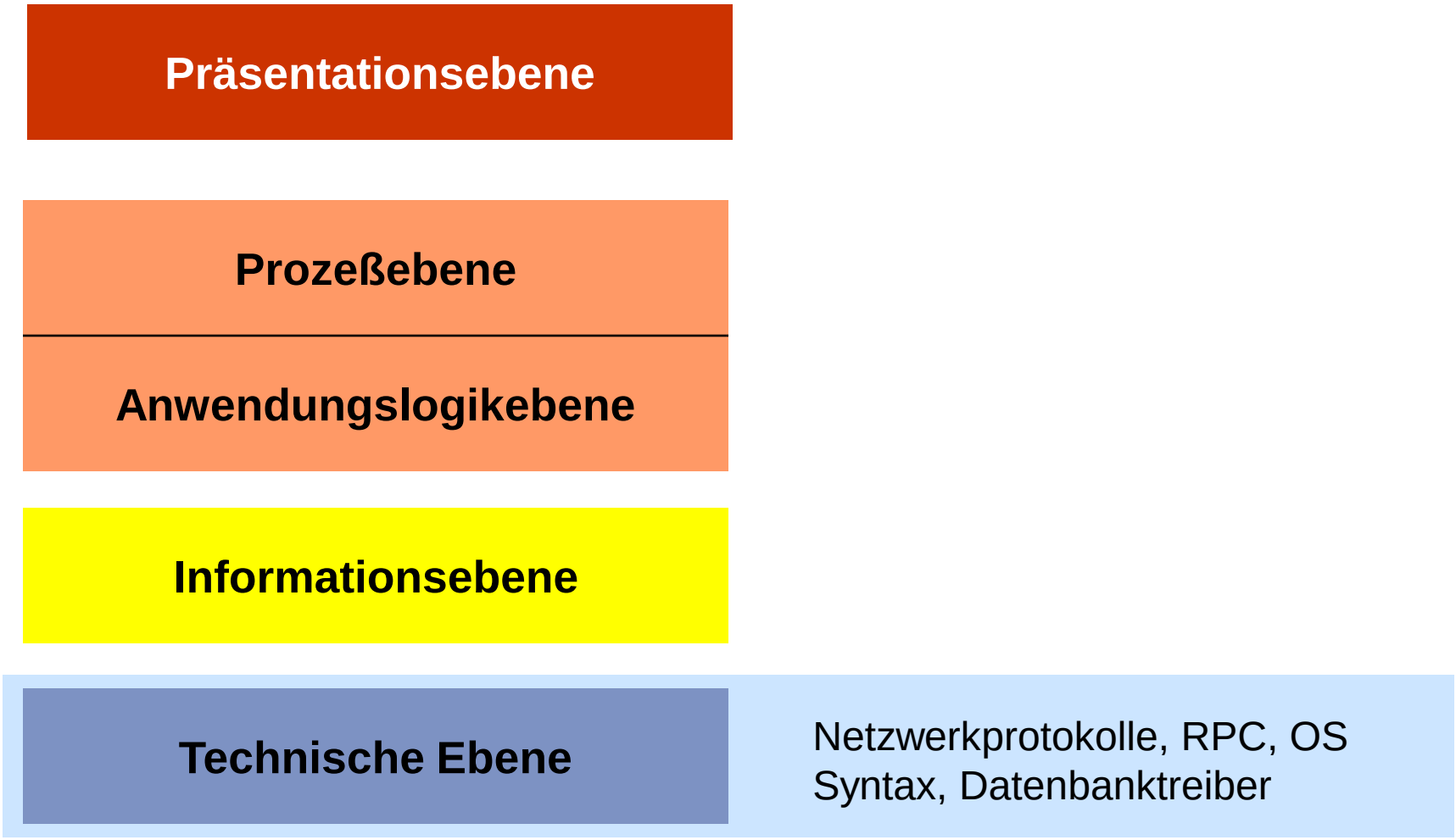


Inhalt

- **Motivation**
- **Integrationsebenen**
- **Generelle Integration von Komponenten als Dienste**
  - **SOAP basierende Webservices**
    - SOAP, WSDL, UDDI
    - Erstellung von WSDL: bottom up vs. Top down
  - **Umsetzung / Integration in klick-and-bau.com mit Web-Services**
  - **REST – vereinfachte Erstellung von Diensten**
- **Mashups: Integration auf Präsentations- und Businessebene**

INTEGRATIONSEBENEN

Integrationsebenen: technische Ebene



Technische Ebene – Sichtweise und Probleme

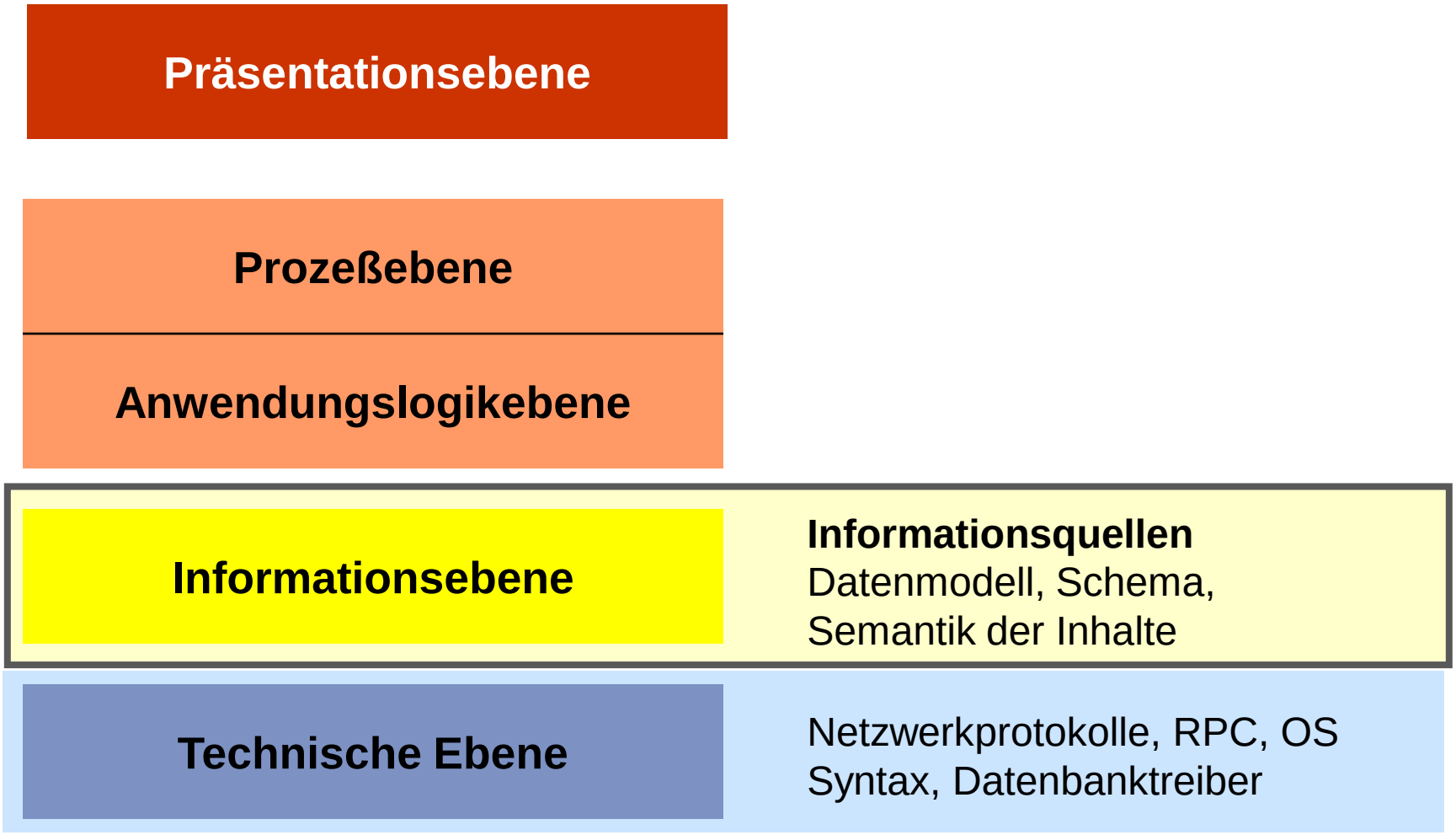
Sichtweise:

- Tatsächliche Verbindung zu einem System
- Spezielle Eigenschaften von Hardware, Betriebssystem
- Eigenschaften von Treibern, auch z.b. für Datenbankverbindung

Probleme (in umfangreichen System)

- Viele spezifischen Details, große Heterogenität
- Hoher Wartungsaufwand (z.b. Sicherheitsupdates)
- Wenig Wiederverwendbarkeit (falls z.b. 4 Systeme mit 4 versch. OS)
- Wenig bis kein Anwendungsbezug („nur die Datenbank an sich“)

Integrationsebenen: Informationsebene



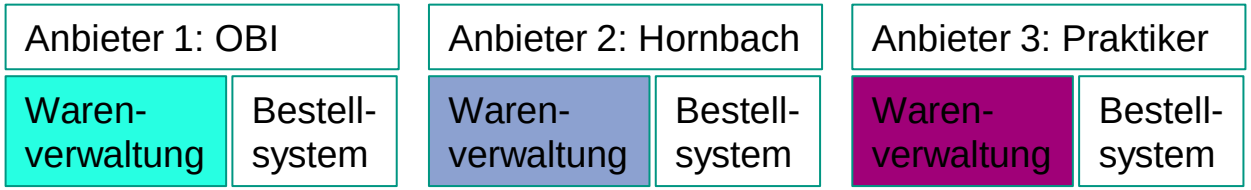
Informationsebene – Sichtweise und Probleme

Sichtweise:

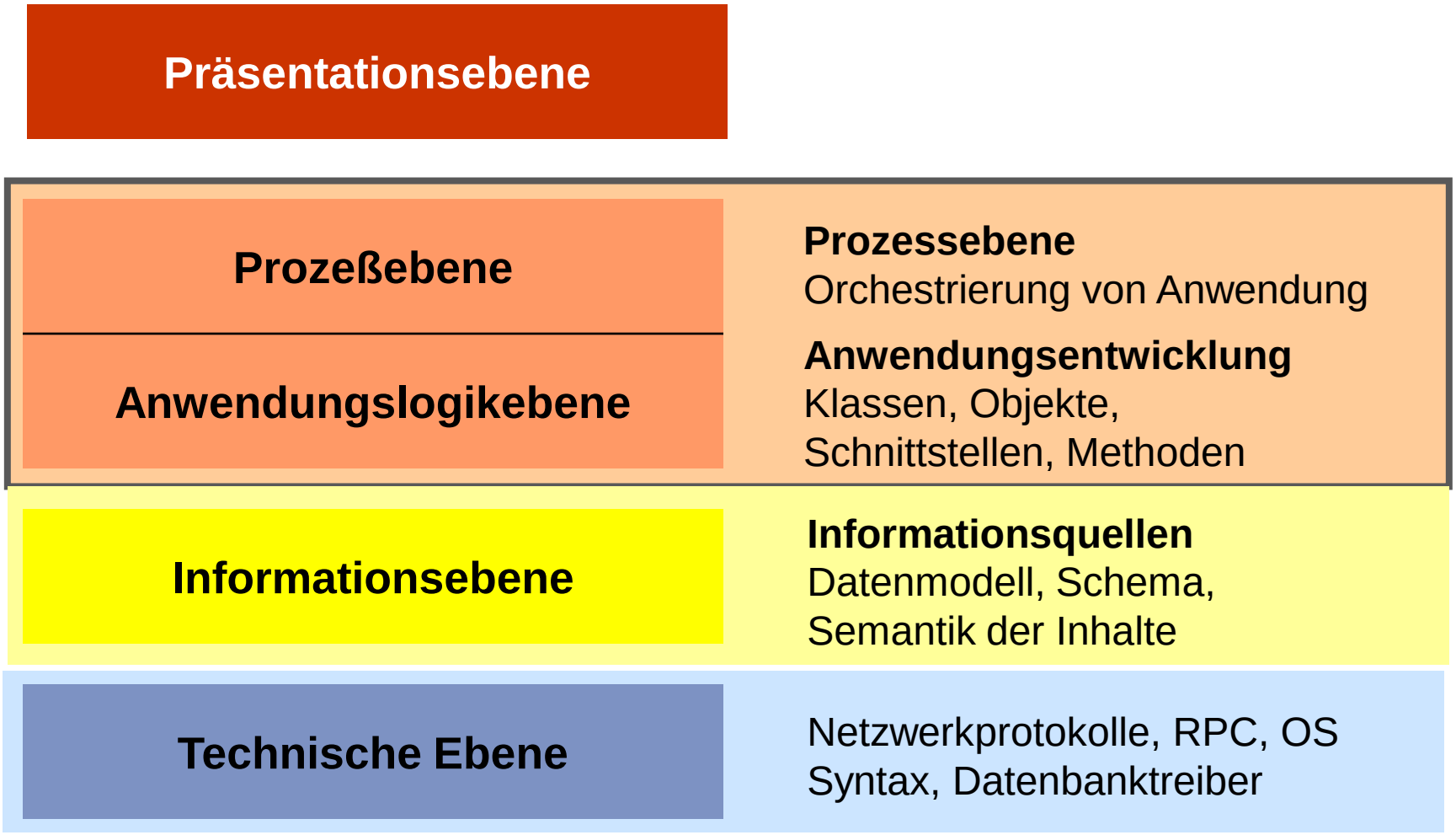
- Tatsächliche Verbindung zu einer Datenquelle, z.b. SQL Datenbank
- Spezielle Eigenschaften von Datenbanksystem, z.b. SQL standard
- Eigenschaften von Datenkommunikation (Verbindungsaufbau, ORM, ...)

Probleme (in umfangreichen System)

- Viele spezifischen Details, große Heterogenität
- Viel Anpassungsaufwand (z.b. ETL für Daten versch. Quellen) bzw. Bei Änderung von Schema / Attribute durch Betreiber
- Wenig Anwendungsbezug („nur die Daten an sich“)



Integrationsebenen: Anwendungslogik



Anwendungsebene

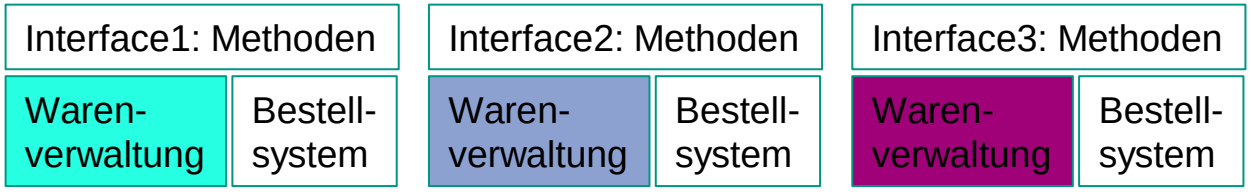
Sichtweise:

- Klassen und Objekte: enthalten Daten & Logik der Anwendung
- Schnittstellen und Methoden: Kapseln bestimmte Logik, z.B. Zur Anfrage von Warendaten

Prozesse:

- Kontroller: Orchestrieren Aufrufe von Methoden.
  Beispiel: ladeWaren aller Anbieter, erstelleGesamtliste

=> **Ziel in verteilter Anwendung**: Komponenten als Schnittstellen



Komponenten als Schnittstellen

- Verdecken unterliegende Datenquellen und Systeme:
  löst Problem der Heterogenität von z.b. Datenbanksystemen
- Schnittstelle kann durch mehrere Systeme intern realisiert werden:
  löst Problem der Skalierbarkeit („System wächst mit Anforderungen“)
  - Beispiel: Clouddienste, Google App Engine (Software as a Service)
- Datenquelle kann durch unterschiedliche Systeme realisiert werden:
  verdeckt Problem unterschiedlicher SQL Anfragen
- Schnittstelle liefert Objekte statt relationaler Daten
  verdeckt ORM Probleme, wird direkt in Komponente implementiert

GENERELLE INTEGRATION VON KOMPONENTEN ALS DIENSTE

Denken in Diensten

- Übertragung der Dienstleistungsorientierung in der internen (und externen) Unternehmensorganisation auf die IT-Infrastruktur

- **Einzelne Komponenten erbringen Dienste**

- Wichtig: Dienstschnittstelle als Vertrag
  - welche Dienste / Methoden werden angeboten?
  - wie kann man sie in Anspruch nehmen?

- Zusätzlich: Wie kann man Dienste finden?

Denken in Diensten (2)

Konsequente Trennung zwischen

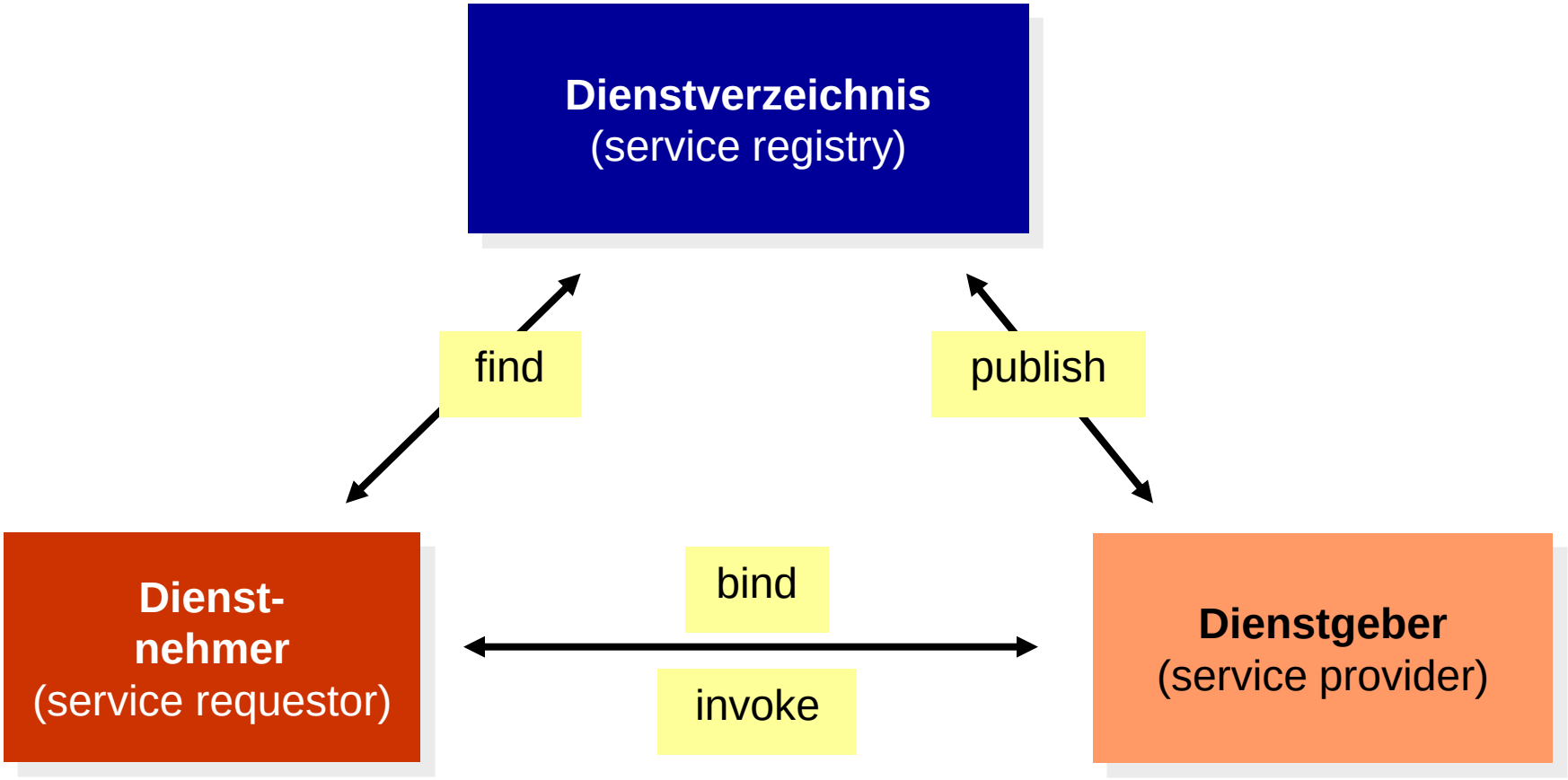
- **Schnittstelle**
  - nach außen veröffentlicht
  - dauerhaft
  - Standardisierung von Diensten möglich
- **Implementierung**
  - vollkommen privat
  - kann beliebig geändert werden
  - insbesondere: Austausch von Komponenten gegenüber besseren möglich

Analogie

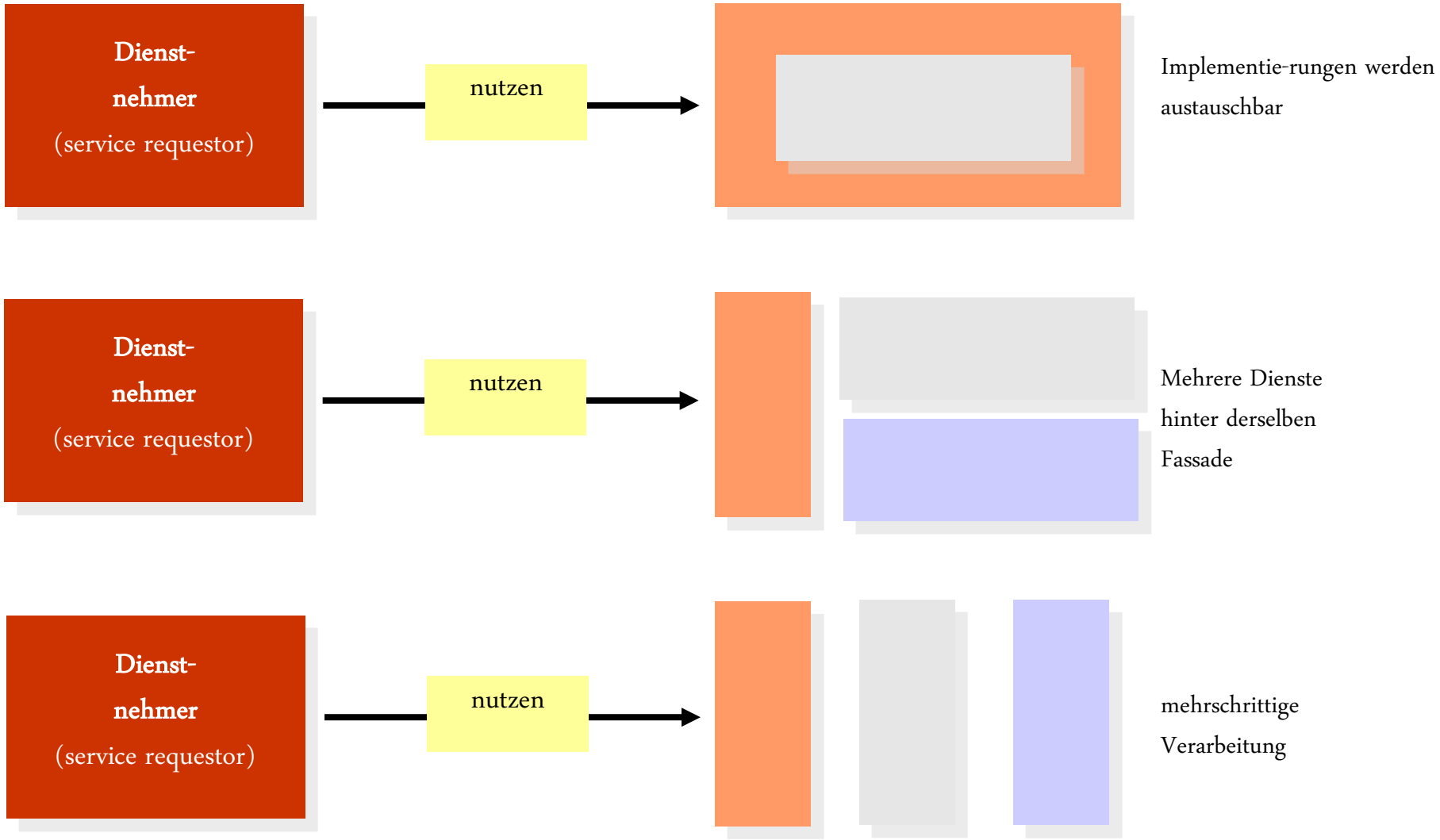
Produktkatalog mit
Preisen und Bestell-/ Lieferbedingungen

Interne Geschäftsprozesse
(Logistik, Buchhaltung etc.)

Dienstorientierte Architekturen



Dienstorientierte Architekturen – Lose Kopplung



Kernprobleme dienstorientierter Architekturen

- Kommunikationsprotokoll (technische Ebene)
  - Kodierung von Methodenaufruf und Parameterübergabe
  - Datenaustausch (Serialisierung)
  - Authentifizierung, Sicherheit etc.
- Dienstbeschreibung
  - Konkrete Adresse (wie erreiche ich den Dienst)
  - Dienstschnittstelle (wie rufe ich ihn auf)
  - Semantik des Dienstes (was tut der Dienst)
  - Organisatorisches (was kostet der Dienst o.ä.)
- Verzeichnisdienste
- Möglichkeiten zur Aggregierung von Diensten

Interaktionsparadigmen

- Entfernter Methodenaufruf (**Remote Procedure Call**)
  - Übertragung des Methodenaufrufs in Programmiersprachen auf verteilte Umgebungen
  - Semantik liegt in der Methode und ihren Parametern
  - meist synchrone Kommunikation
  - resultiert meist in feingranularen Schnittstellen
- Nachrichtenbasierte Kommunikation (**Messaging**)
  - es werden Dokumente ausgetauscht
  - Semantik liegt in den Nachrichtentypen und ihren Instanzen
  - sehr gut für asynchrone Kommunikation geeignet
    - Messaging Queuing Systems
  - oft sehr generische Schnittstellen

SOAP – als ein Beispiel zur Erstellung von Komponenten als Diensten

WEBSERVICES

SOAP basierte WEBSERVICES

SOAP basierte Webservices als Beispiel, da diese

- Klar definierte Schnittstelle ermöglichen
- Klar definierte Serialisierung per XML
- Standardisierte Verzeichnisdienste ermöglichen

Alternativen

- Früher: Corba, Remote Procedure Call APIs (Spezifisch z.b. Für JAVA)
- Prinzipien von SOAP Webservices übertragbar, z.b. Auf REST (in nächster Vorlesung): kompaktere Serialisierung als SOAP, aber keine eindeutige Serialisierung (z.b. JSON, CSV,...) / Methoden (Parameter als HTTP GET)

Was sind Web Services?

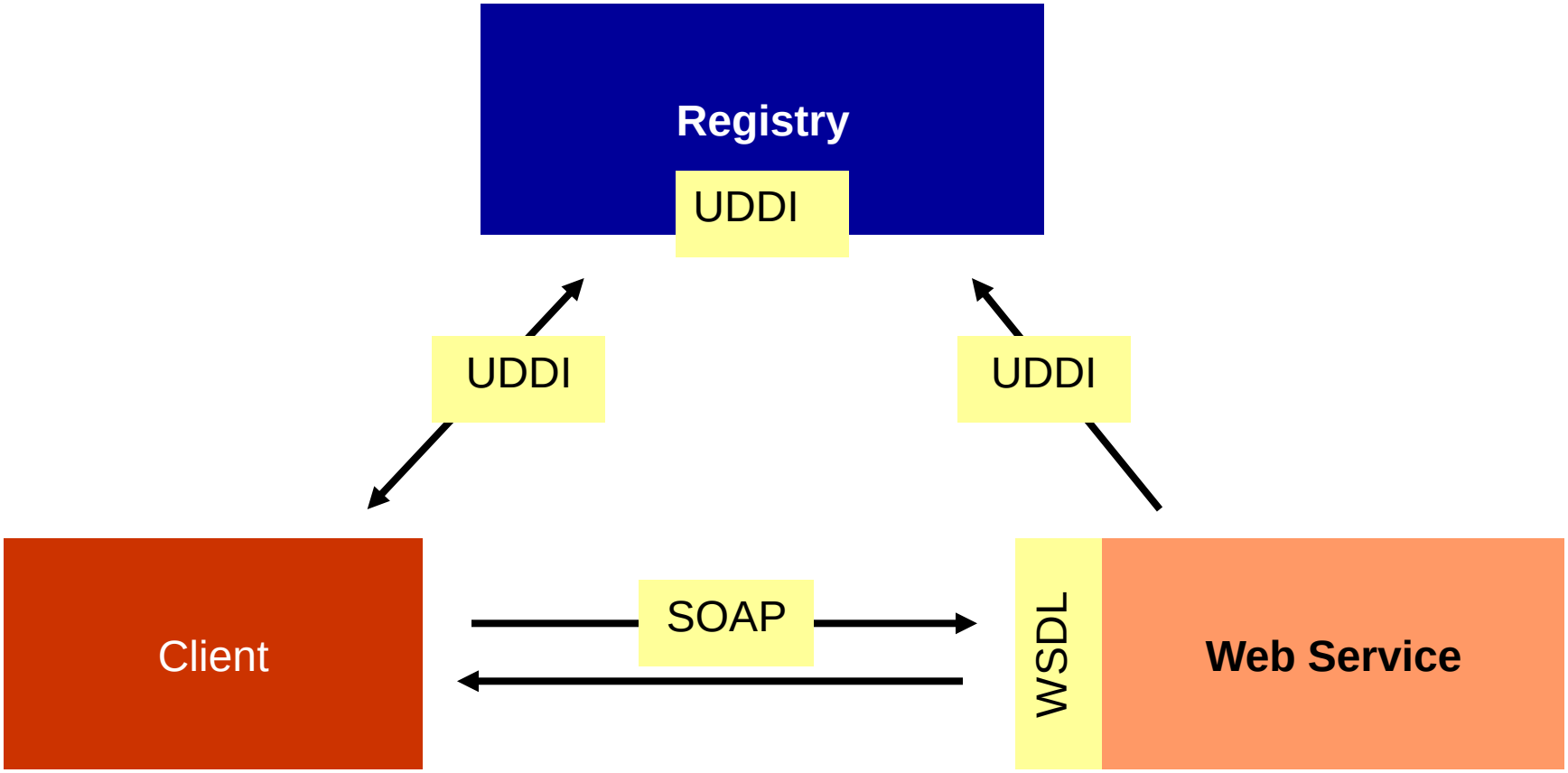
- Web Services sind
  - verteilte,
  - lose gekoppelte und
  - wiederverwendbare Software-Komponenten, auf die
  - über Standard-Internetprotokolle
  - programmatisch zugegriffen werden kann.

- Pragmatisch: Dienste, die mittels SOAP angesprochen werden können
  - und (meist) mittels WSDL beschrieben werden

Web Services — Die Technik

- **SOAP**
  (Simple Object Access Protocol)
  - Aufrufprotokoll
  - unterschiedliche Transportprotokolle: HTTP, SMTP
- **WSDL**
  (Web Services Description Language)
  - Schnittstellenbeschreibung
- **UDDI**
  (Universal Description, Discovery and Integration)
  - Dienstbeschreibung und Auffindung
  - universeller Verzeichnisdienst für Dienstleistungen

SOAP, UDDI, WSDL



SOAP: Überblick

- In XML kodierter entfernter Methodenaufruf (RPC) bzw. Nachrichtenaustauschprotokoll
- Beliebiges Transportprotokoll, z.B.
  - **Meistens verwendet:** synchrone Aufrufe über HTTP:
    Aufrufer wartet auf die Dienstantwort
    klassischer RPC
  - asynchrone Aufrufe über SMTP (Mail):
    Entkopplung von Aufruf und Antwort
    Messaging
- Kodierungsregeln für Datentypen
  - Standard-Regeln entstammen XML Schema

SOAP: Ein Beispiel

Aufruf

```
<soap:Envelope>
 <soap:Body>
  <xmlns:m="http://www.stock.org/stock" />
  <m:GetStockPrice>
     <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
 </soap:Body>
</soap:Envelope>
```

Antwort

```
<soap:Envelope>
 <soap:Body>
  <xmlns:m="http://www.stock.org/stock" />
  <m:GetStockPriceResponse>
   <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
 </soap:Body>
</soap:Envelope>
```

SOAP: Aufbau eines Aufrufs

- SOAP-Envelope
  - SOAP-Header
    - z.B. Authentifizierung, Routing, Logging, Transaktionsnummern
    - Metadaten über den Aufruf
  - SOAP-Body
    - die eigentliche Methode, die aufgerufen werden soll
    - die Parameter
    - bzw. die ausgetauschte Nachricht

SOAP: Beispiel Amazon Artikel WS

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <a:KeywordSearchRequest xmlns:a="urn:PI/DevCentral/SoapService">
            <KeywordSearchRequest xsi:type="m:KeywordRequest">
                <keyword >dog</keyword>
                <page >1</page>
                <mode>book</mode>
                <tag>webservices-20</tag>
                <type>lite</type>
                <dev-tag>your-dev-tag</dev-tag>
                <format>xml</format>
                <version>1.0</version>
            </KeywordSearchRequest>
       </a:KeywordSearchRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP und Sicherheit

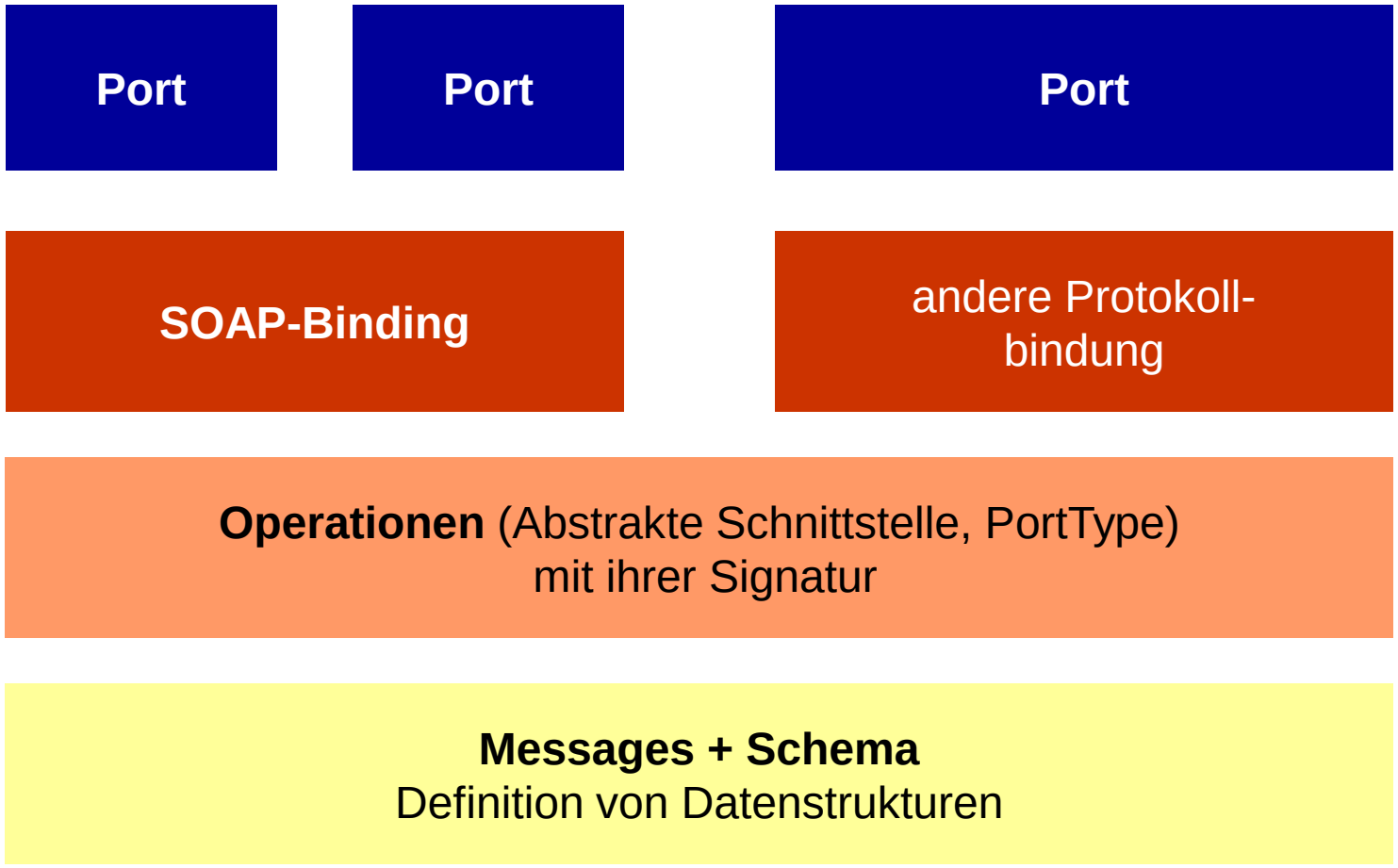
- Authentisierung
  - Verteilte Authentisierungsprotokolle über SOAP (z.B. Kerberos)
- Signaturen
  - XML Signature (REC)
- Verschlüsselung
  - XML Encryption (REC)

- Transaktionen
  - BPEL4WS hat hier einige Ansätze zu bieten
  - + diverse andere Ansätze

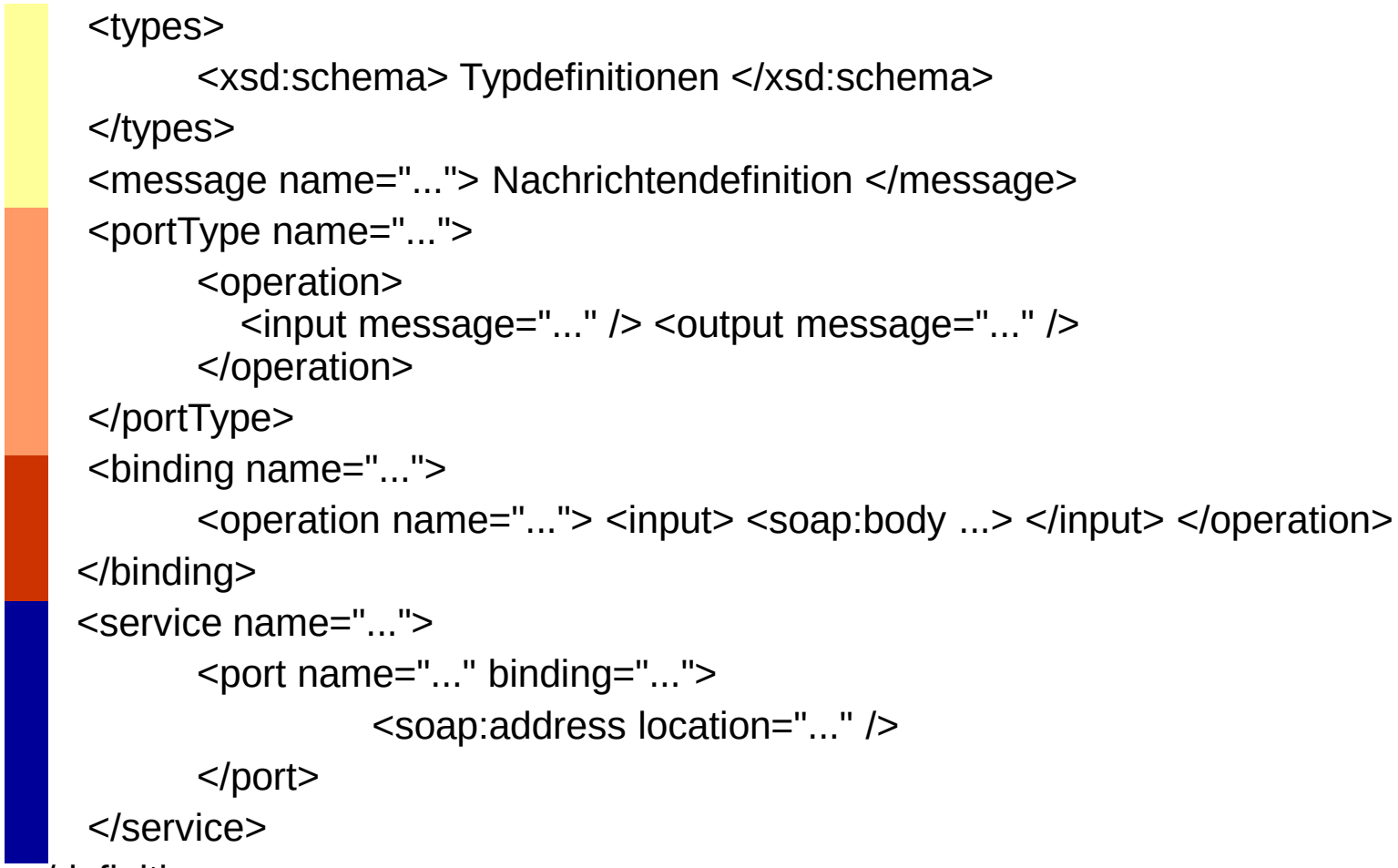
WSDL: Überblick

- Welche Operationen bietet der Dienst an?
- Welche Parameter haben die Operationen
  - Struktur der Aufrufnachricht
  - Struktur des Ergebnisses
  - Rückgriff auf XML Schema
- Wo und wie kann ich einen Dienst erreichen?
  - Adresse
  - Informationen über das Protokoll
    - z.B. SOAP, auch mehrere Protokolle!
    - verwendetes Transportprotokoll: HTTP, SMTP etc.
- aber: keine semantische Dienstbeschreibung

WSDL: Aufbau


Port
Port
Port
SOAP-Binding
andere Protokoll-bindung
Operationen (Abstrakte Schnittstelle, PortType) mit ihrer Signatur
Messages + Schema
Definition von Datenstrukturen

WSDL: Aufbau (2)



Typdefinitionen

- Typdefinitionen in WSDL sind eingebettete XML Schema-Definitionen

```
<definitions name="AmazonSearch" targetNamespace="urn:PI/DevCentral/SoapService" ...>
    <types>
      <xsd:schema       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="urn:PI/DevCentral/SoapService">
            <xsd:complexType name="KeywordRequest">
              <xsd:all>
                <xsd:element name="keyword" type="xsd:string"/>
                <xsd:element name="page" type="xsd:string"/>
                <xsd:element name="mode" type="xsd:string"/>
                <xsd:element name="tag" type="xsd:string"/>
                <xsd:element name="type" type="xsd:string"/>
                <xsd:element name="devtag" type="xsd:string"/>
                <xsd:element name="version" type="xsd:string"/>
            </xsd:all>
            </xsd:complexType>
```

Nachrichten

- Nachrichten können auch aus mehreren Teilen bestehen

- Jeder Teil wird mit einem Schema-Typ typisiert

```
<message name="KeywordSearchRequest">
        <part      name="KeywordSearchRequest"
        type="typens:KeywordRequest"/>
</message>
<message name="KeywordSearchResponse">
        <part      name="return"
                   type="typens:ProductInfo"/>
</message>
```

Operationen

- Schnittstellendefinition
  - Name der Methode, Eingabe- und Ausgabeparameter
  - Fehlerbehandlung
    - <fault name="..." message="...">

```
<portType name="AmazonSearchPort">
    <operation name="KeywordSearchRequest">
            <input message="typens:KeywordSearchRequest"/>
            <output message="typens:KeywordSearchResponse"/>
    </operation>
    ...
</portType>
```

Protokollbindung

- Protokollspezifische Angaben
  - Welches (Transport-) Protokoll?
    - SOAP over HTTP (oder SMTP, FTP), HTTP GET/POST, ...
  - Zusätzliche Angaben, die notwendig sind, um einen korrekten Aufruf zu bewerkstelligen
    - Interaktionsparadigma
    - Kodierung
    - Metainformationen (SOAP Header)

Typdefinitionen (2)

```
<xsd:complexType name="ProductInfo">
    <xsd:all><xsd:element name="Details" type="typens:DetailsArray"/></xsd:all>
</xsd:complexType>
<xsd:complexType name="Details">
    <xsd:all>          <xsd:element name="Url" type="xsd:string"/>
                       <xsd:element name="ProductName" type="xsd:string"/>
                       <xsd:element name="Authors" type="typens:AuthorArray"/>
                       <xsd:element name="ImageUrlSmall" type="xsd:string"/>
                       <xsd:element name="OurPrice" type="xsd:string"/>
                       <xsd:element name="Publisher" type="xsd:string"/>
                       <xsd:element name="Isbn" type="xsd:string"/></xsd:all>
</xsd:complexType>
<xsd:complexType name="DetailsArray">
    <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
            <xsd:attribute ref="soapenc:arrayType"
                wsdl:arrayType="typens:Details[]"/>
            </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Typ für Arrays, ebenso Listen möglich...

Protokollbindung

RPC als Interaktions-paradigma

SOAP über HTTP

```
<binding name="AmazonSearchBinding"
         type="typens:AmazonSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/s
    <operation name="KeywordSearchRequest">
        <soap:operation soapAction="urn:PI/DevCentral/SoapService"/>
          <input>
            <soap:body use="encoded" namespace="urn:PI/DevCentral/SoapService"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
          </input>
        <output>
            <soap:body use="encoded"namespace="urn:PI/DevCentral/SoapService"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
    </operation>
    ...
</binding>
```

Ports

- Festlegen der konkreten Netzwerkadresse

```
<service name="AmazonSearchService">
    <port name="AmazonSearchPort"
          binding="typens:AmazonSearchBinding">
          <soap:address location="http://soap.amazon.com/onca/soap"/>
    </port>
</service>
```

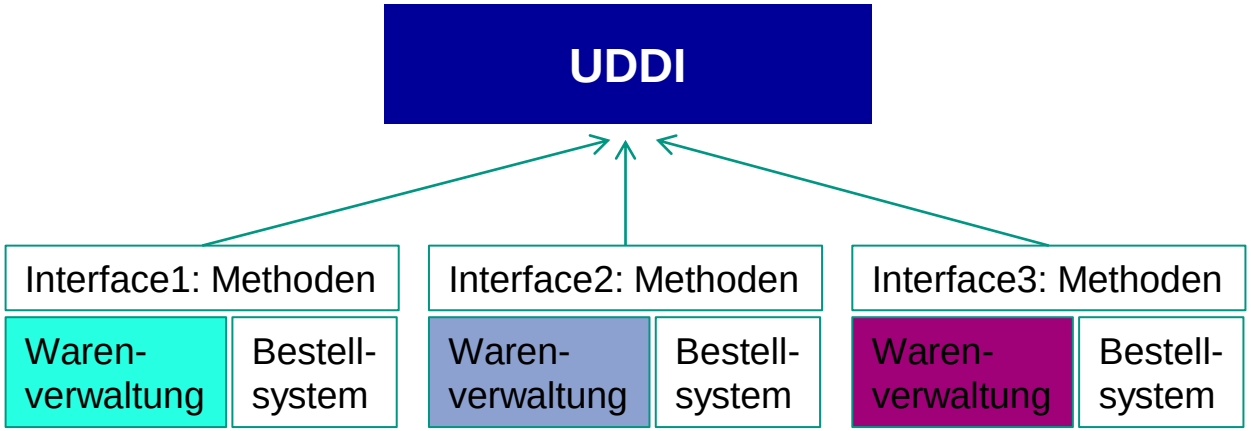
Hier wird festgestellt, wo der Dienst angesprochen werden kann

WSDL: Fazit

- WSDL erlaubt die Beschreibung aller Informationen über die Schnittstelle und die Erreichbarkeit eines Dienstes
  - Mit Hilfe der in WSDL gemachten Angaben läßt sich problemlos ein korrekter SOAP-Aufruf schreiben
  - Beispiele:
    - .NET Einbinden von beliebigen Web Services in die IDE bzw. in Office VB Script
    - Java JAX-WS (oder auch Axis) erlaubt die automatische Generierung entsprechender Klassen
- Allerdings:
  - WSDL macht keinerlei Angaben, was der Dienst denn nun wirklich tut

UDDI - Universal Description, Discovery and Integration

- White , yellow, green pages: URL eines Dienstes, Anbieterinformation, Kategorisierung der Dienste (nach Thema), Schnittstellenbeschreibung
- **UDDI in Praxis**
  - Meist fokussiert auf eine verteilte Anwendung, d.h. Semantik des Dienstes ist bekannt für Enwickler, UDDI registriert vorhandene Implementierungen, z.b. Alle Dienste „WarenverwaltungBaumarkt“
  - Oder kein UDDI: Adressen bekannter Dienste als Properties.
  - Daher dann auch keine symantische Dienstbeschreibung nötig.



Erstellung von WSDL: bottom up vs. Top down

WSDL kann über zwei Wege beschrieben werden

- Bottom up: Modellierung in UML Tool (z.b. Enterprise Architect). Schnittstelle wird vollständig modelliert und serialisiert.
  - Vorteile:
    - garantiert unabhängig von Programmiersprache
    - Genaue Spezifikation auch z.B. Von Restriktionen.
    - Implementierung in JAVA: z.b. mit AXIS
  - Nachteile:
    - Implementierung aufwändiger
    - Nicht alle Restriktionen werden von Implementierungen umgesetzt
- Top-Down: Zuerst Implemtierung von Interface in z.b. JAVA
  - Vorteil: einfacher, on-the-fly.
  - Nachteil: u.u spezifisch für eine Programmiersprache

WDL Top Down mit JAX-WS in JAVA

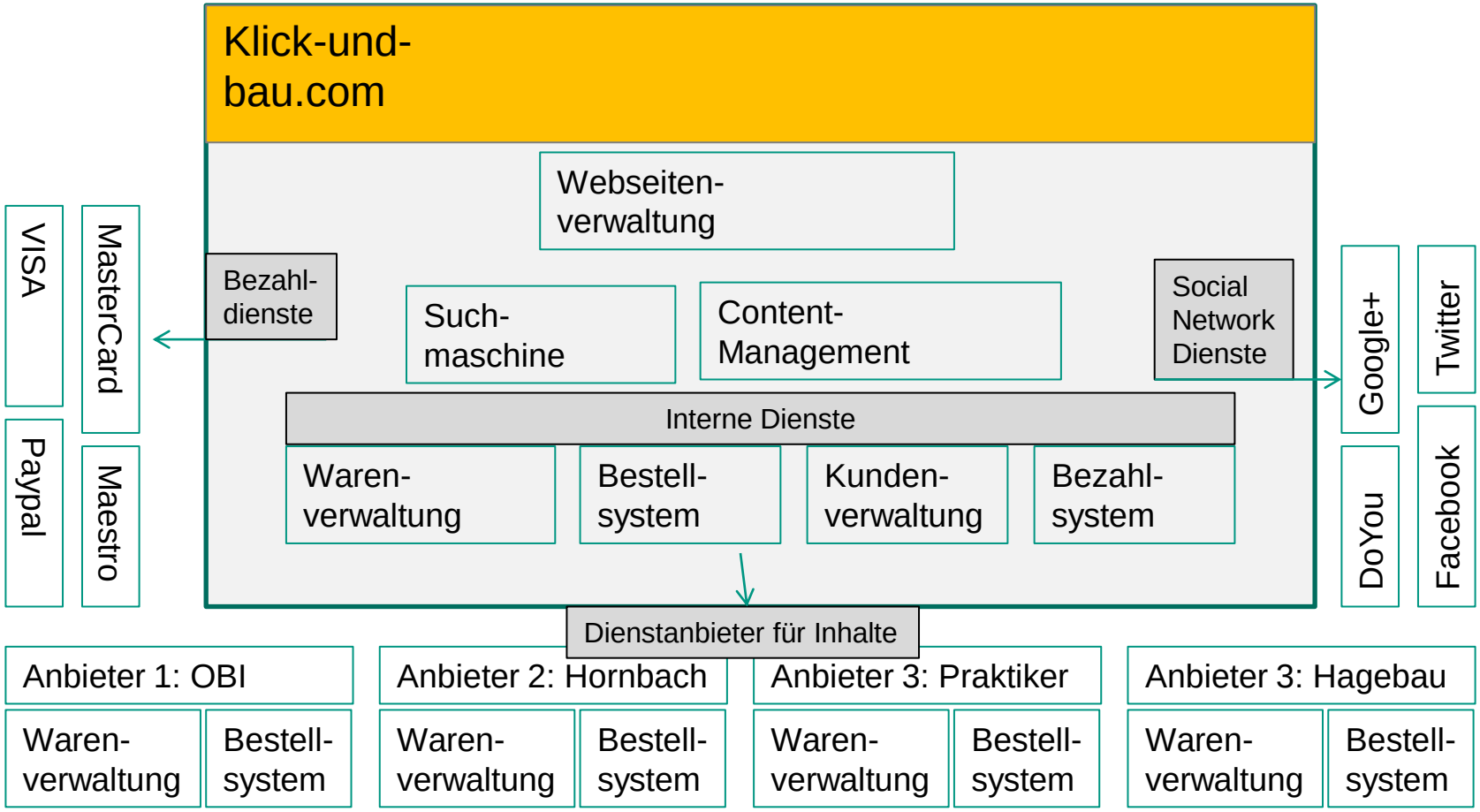
1. Erstelle gewöhnliches JAVA Interface
2. Annotiere mit spezifischen Annotationen für JAX-WS
3. Deployment in z.B. Jboss
4. Aufruf der annotierten Adresse bringt WSDL
5. Nutzbar per JAX-WS für Clienterstellung

```
@Stateless
@WebService(name = ArticleService .WS_NAME, serviceName =
    ArticleService .WS_SERVICE_NAME, targetNamespace =
    "http://ws.article.klickandbau.com/")
@WebContext(contextRoot = ReportingService.CONTEXT_ROOT,
    urlPattern = ReportingService.WS_URL_PATTERN)
public class ArticleService implements ArticleRemote …
```

UMSETZUNG / INTEGRATION IN KLICK-AND-BAU.COM MIT DIENSTEN

Motivation

Komponentensicht auf klick-und-bau: Komponenten als Dienste

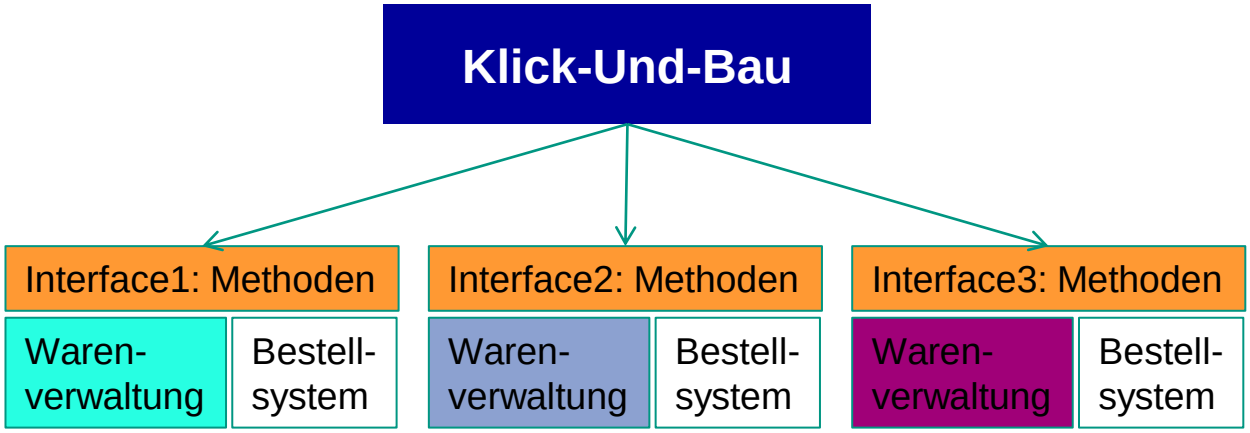


Generelle Umsetzung

- Alle Datenquellen erhalten Schnittstelle zum Zugriff
  - Definition benötigter Daten zum Laden: Abfragemethoden
  - Definition von zu speichernden Daten: CRUD – Methoden
- Alle internenen Systeme erhalten Schnittstelle zum Zugriff
  - Definition von nötigen Abfragen: Abfragemethoden
  - Systeme können intern weitere Dienste aggregieren, z.b. ArticleService

Datenintegration der Anbieterdaten: ideal

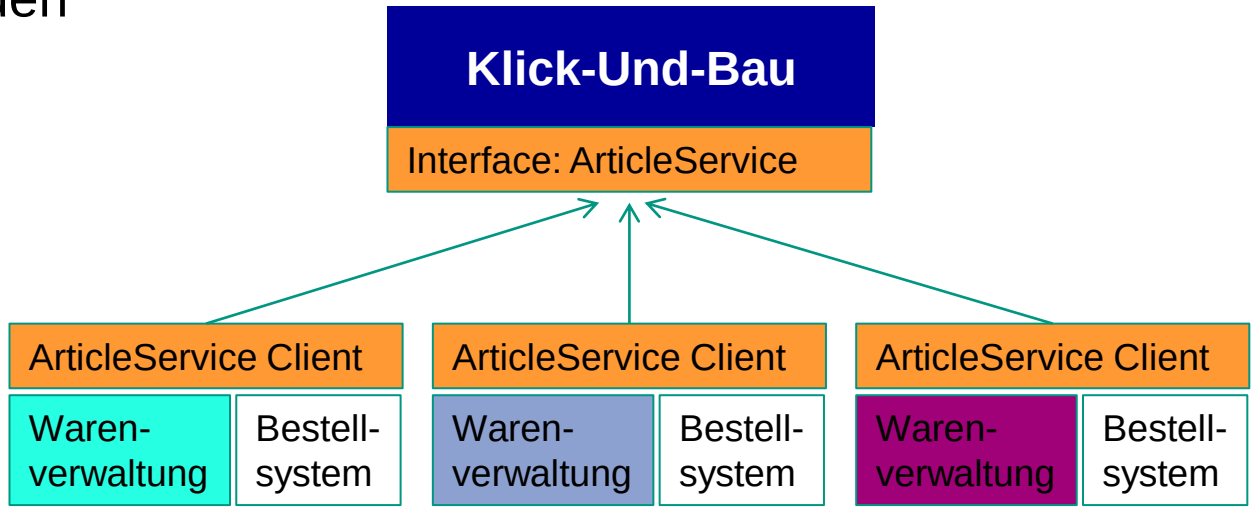
- Ideal: Spezifiziertes Interface wird von allen Anbietern umgesetzt:



  - Ermöglicht beliebige Aktualisierung der Daten.
  - Alle Daten werden in gleichen Objekten abgefragt (kein ETL mehr nötig)
  - Je nach Performanz der Schnittstelle: immer direkt abfragen (keine Datenspeicherung mehr nötig)

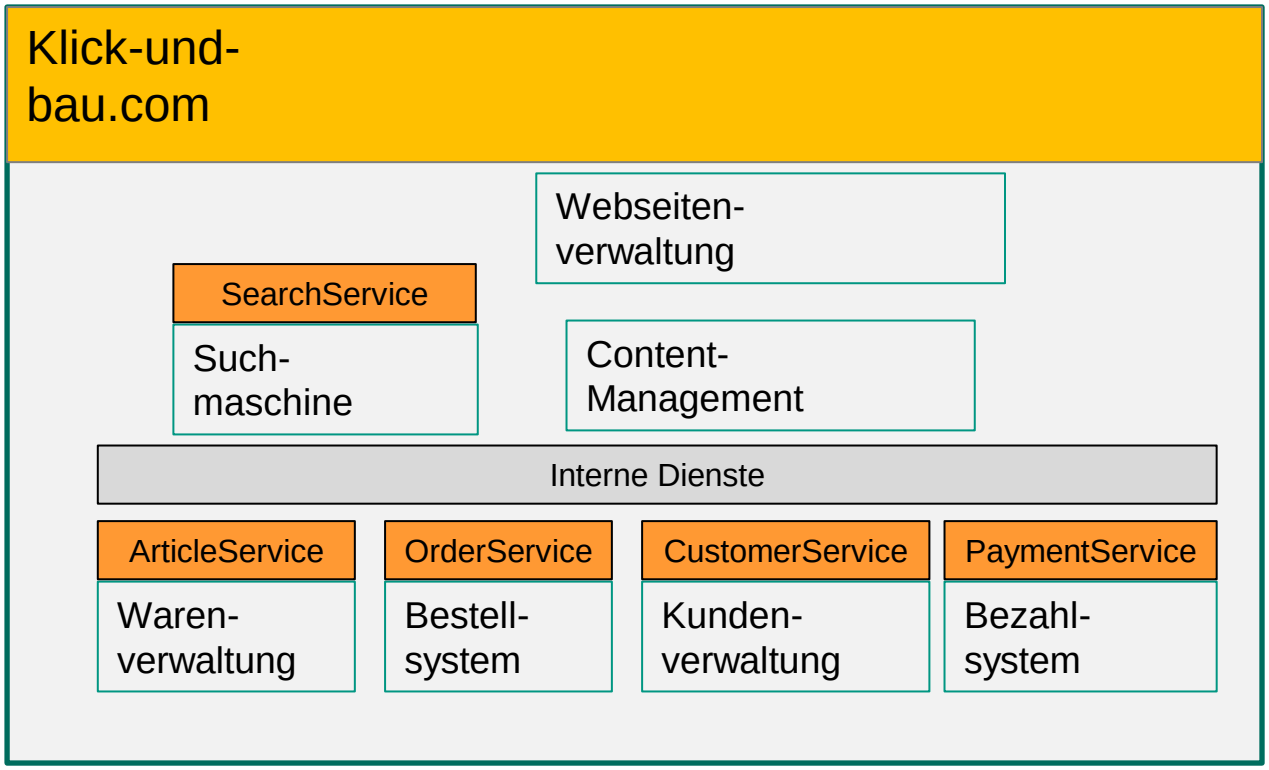
Datenintegration der Anbieterdaten: Variante

- **Variante**: Anbieter wollen selbst bestimmen, wann Daten geschickt werden



  - (auch bei Sicherheitsbedenken von Anbietern): Anbieter implementieren WS-Client des Article Service : push z.b. täglich
  - Alle Daten werden in gleicher Objektstruktur abgefragt
  - Aber: Speicherung der Daten nötig
  - Vorteil: verdeckt Heterogenität der Artikel-Datenbanken!

Interne Dienste

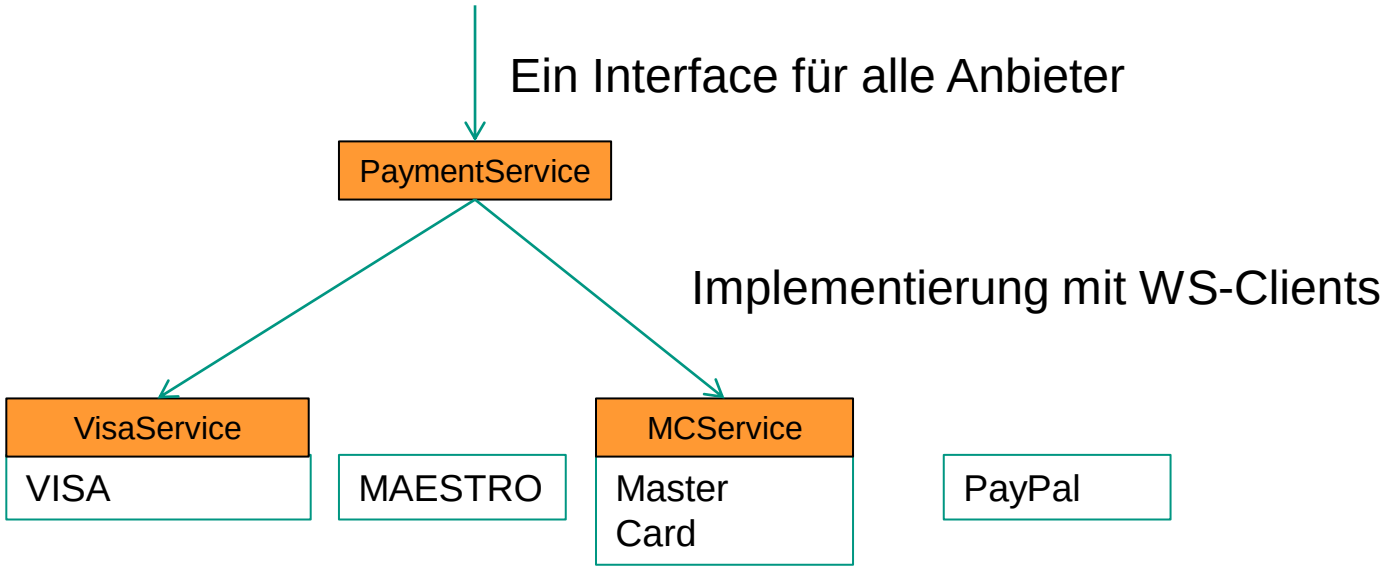


- Über alle internen Komponenten werden Schnittstellen erstellt (zumeist als Standarddienst von Anbietern erhältlich)

  => Aggregation von Diensten: z.b. ArticleService, PayementService

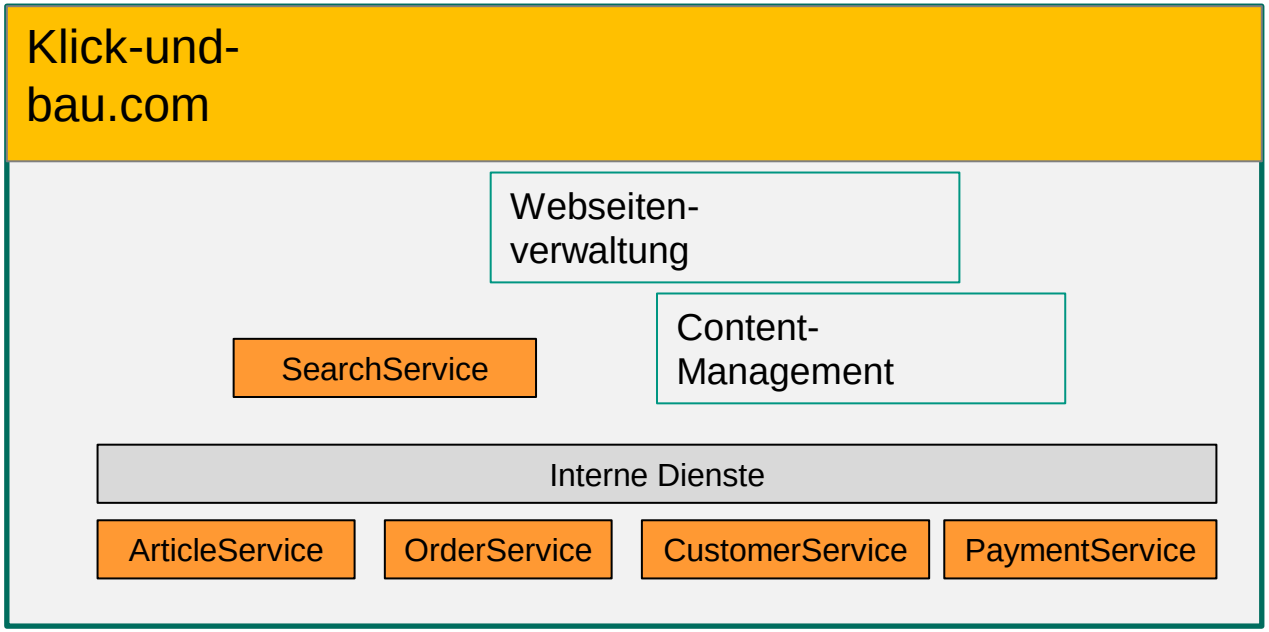
Aggregation von Diensten

Hier gezeigt an Beispiel PaymentService:
Alle Bezahldienste bieten Webservice-Schnittstellen an



Erstellung der Webanwendung

Sämtliche Controller kommunizieren mit Dienstschnittstellen
(vereinfacht stark das Deployment der Webanwendung, z.b. Keine Datenbanktreiber etc. => Direkt z.b. Sehr skalierbar & cloudfähig)



REST INTERFACES

Ausgangspunkt

- Obwohl SOAP als Protokoll „einfach“ sein sollte
  (im Vergleich zu CORBA und anderen verteilten Objektprotokollen),
  stellt es doch einen gewissen Overhead dar
  - Einarbeitungsaufwand in Web-Service-Toolkits
  - Protokolloverhead

- Leichtgewichtigere Varianten für einfachen Zugriff auf Web-Dienste

REST

- **R**epresentational **S**tate **T**ransfer
- Architekturstil für Hypermedia Systeme
  - hervorgegangen aus der Doktorarbeit von Roy Fielding
- Leichtgewichtige Alternative zu Web Services
  - Grundlage sind hauptsächlich Konventionen
  - stark an Standard-HTTP orientiert

REST-Grundkonzepte

- Ressourcen
  - haben eine URI
  - sind die primären Entitäten der Anwendung
    - Z.B. Produkte, Warenkorb, etc.
- Repräsentationen
  - Sind Darstellungen von Ressourcen
  - Können selbst wiederum auf andere Ressourcen verweisen
- CRUD-Operationen (Create, Read, Update, Delete)

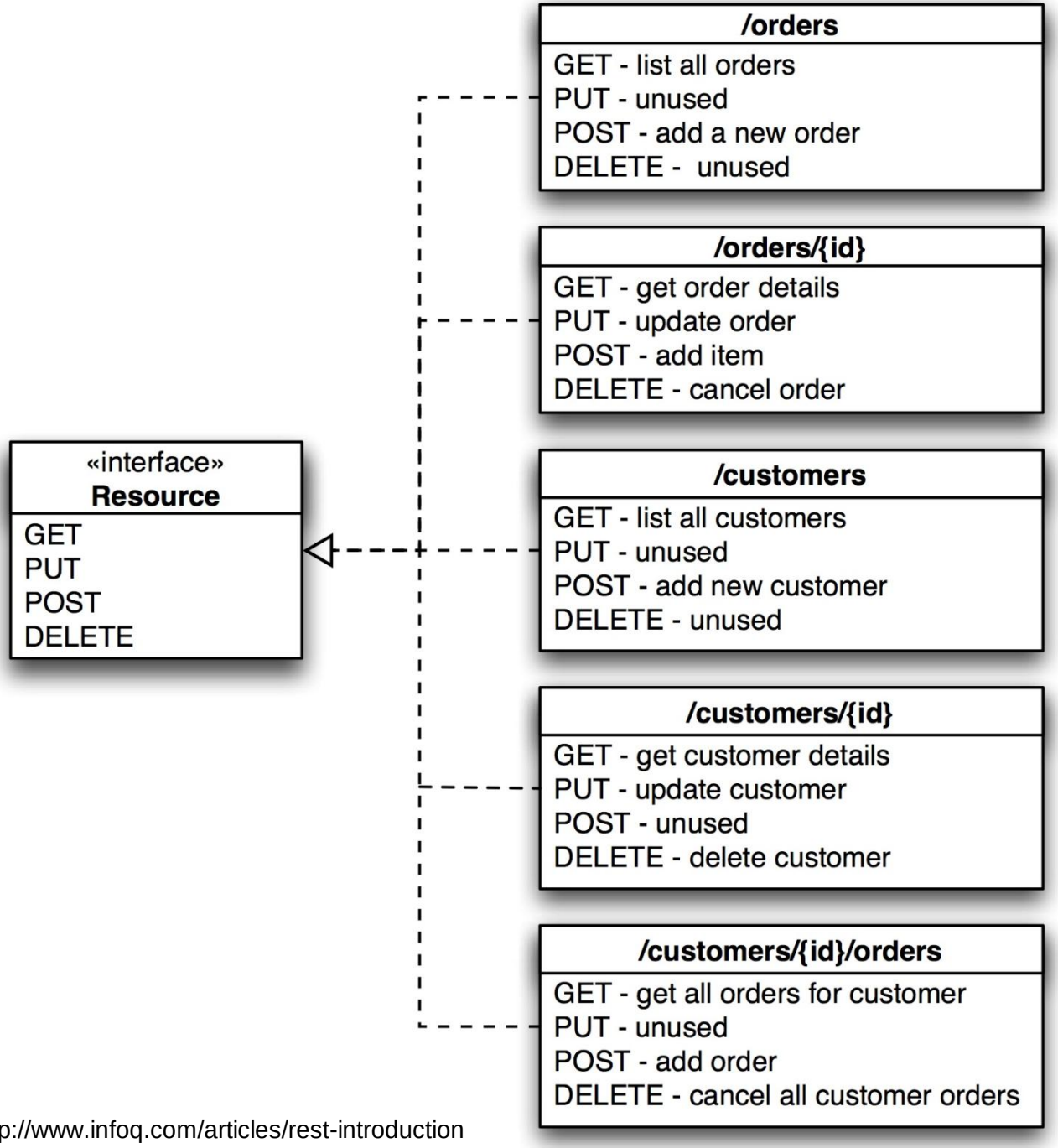
Charakteristika einer REST-Anwendung

- Wie bei der Serviceorientierung:
  - Die Kommunikation wird bei Bedarf durch den Client initiiert.
  - Zustandslose Kommunikation

- Zusätzlich:
  - Resourcen sind durch eine URI adressierbar
  - Repräsentation einer Resource kann als Dokument vom Client angefordert werden.

REST: Grundidee

- Zugriffsprotokoll ist ausschließlich HTTP
- Kann die HTTP-Methoden und ihre Semantik nutzen
  - HTTP GET für die Repräsentation einer Ressource
  - HTTP DELETE für das Löschen von Ressourcen
  - HTTP POST für das Erzeugen und Aktualisieren von Ressourcen
  - HTTP PUT für das Erstellen von Repräsentationen von Ressourcen
- Meistens:
  - komplexe Strukturen: Kodierung der Daten in XML
  - einfache Strukturen: URL-kodiert



http://www.infoq.com/articles/rest-introduction

REST-Beispiel (2)

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

⇒ Ruft die Kundendaten für einen bestimmten Kunden in einem vordefinierten XML-Format ab

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: text/x-vcard
```

⇒ Wie oben, nur als Vcard

REST-Beispiel (3)

```
POST /orders/ HTTP/1.1
Host: example.com

<order id='http://example.com/customers/1234' >
   <amount>23</amount>
   <product ref='http://example.com/products/4554' />
   <customer ref='http://example.com/customers/1234' />
</order>
```

=> Absetzung einer Bestellung

REST + JSON

- Ein erheblicher Overhead von Web Services kommt auch durch die XML-Serialisierung zustanden, was sich insbesondere bei kleinen Parametern zeigt
  - Teilweise Faktor 10-20

- Als alternative Serialisierungsformen hat sich vor allem JSON etabliert:
  - JavaScript Object Notation
  - Lässt sich direkt in JavaScript verarbeiten:
    - my_JSON_object = JSON.parse(http_request.responseText);
  - Browser-Unterstützung

REST + JSON

```
order:
{
     "id": "http://example.com/customers/1234"
    "amount": "23",
    "product": "http://example.com/products/4554"
     "customer": "http://example.com/customers/1234"
}
```

REST vs. SOAP

- REST reduziert den Overhead von Service-Aufrufen durch Konventionen
  - Dadurch weniger Flexibilität
  - Aber der 80%-Fall lässt sich einfacher realisieren
- Zusammen mit JSON auch deutlich performanter, gerade bei häufigeren und kleineren Zugriffen
- REST fehlt die Einbettung in Enterprise-Kontexte

MASHUPS : INTEGRATION AUF PRÄSENTATIONS- UND BUSINESSEBENE

Businesssebene

- Grundidee ist hier: (Cloud) basiertes Mashup von Diensten zu einer neuen Anwendung

- Ansätze
  - Einbindung bestehender Services (SOAP, WSDL)
  - Einfache Programmierung (z.b. Auf Basis von JavaScript Bibliotheken)

Cloud Mashups (mit Google Apps Services)

REFERENCE

- Google Apps Services
  - Calendar
  - Contacts
  - DocsList Experimental!
  - Document
  - Domain Deprecated
  - Drive
  - Forms
  - Gmail
  - Groups
  - Language
  - Maps
  - Sites
  - Spreadsheet
- Advanced Google Services Experimental!
  - How to Enable Advanced Services
  - Admin SDK
  - AdSense
  - Analytics
  - BigQuery
  - Calendar New!
  - Drive New!
  - Fusion Tables
  - Google+ Domains
  - Mirror
  - Prediction

Script, the BigQuery service uses the same objects, methods, and parameters as the public API.

Sample code

The sample code below uses version 2 of the API.

Open code in new window

Run query

This sample queries for the top 30 most common words over 10 characters long that appear in the Shakespeare sample dataset.

```
function runQuery() {
  // Replace this value with the project ID listed in the Google
  // Developers Console project.
  var projectId = 'XXXXXXXX';

  var request = {
    query: 'SELECT TOP(word, 300) AS word, COUNT(*) AS word_count ' +
      'FROM publicdata:samples.shakespeare WHERE LENGTH(word) > 10;'
  };
  var queryResults = BigQuery.Jobs.query(request, projectId);
  var jobId = queryResults.jobReference.jobId;

  // Check on status of the Query Job.
  var sleepTimeMs = 500;
  while (!queryResults.jobComplete) {
    Utilities.sleep(sleepTimeMs);
    sleepTimeMs *= 2;
    queryResults = BigQuery.Jobs.getQueryResults(projectId, jobId);
  }

  // Get all the rows of results.
  var rows = queryResults.rows;
  while (queryResults.pageToken) {
    queryResults = BigQuery.Jobs.getQueryResults(projectId, jobId, {
      pageToken: queryResults.pageToken
    });
    rows = rows.concat(queryResults.rows);
  }

  if (rows) {
```

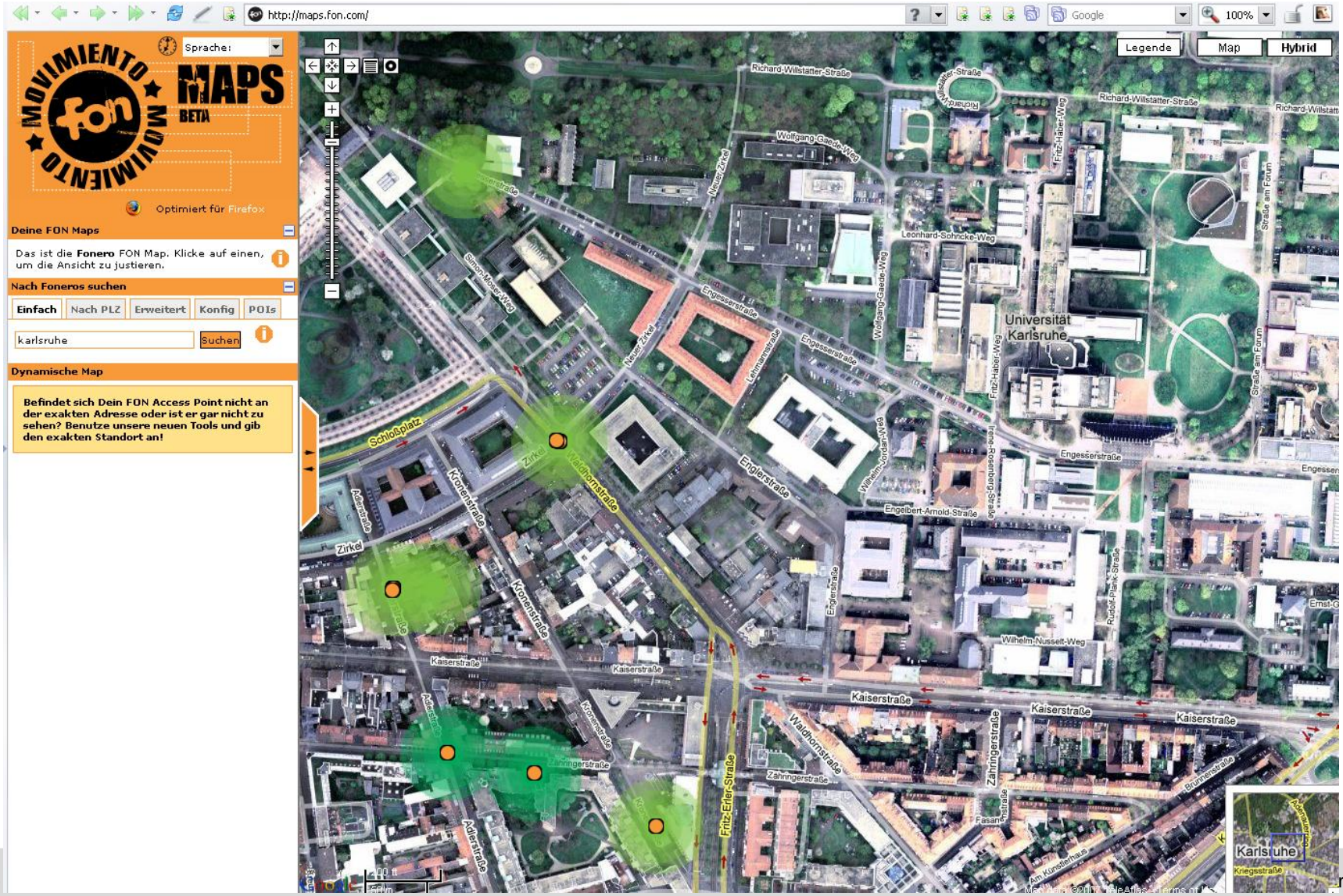
Präsentationsebene

- Grundidee ist hier: Darstellung von Inhalten unterschiedlicher Quellen

- Ansätze
  - Einbindung gerenderter Inhalte, also HTML-Fragmente (IFRAME)
  - reine Overlay-Techniken
  - Einbindung von GUI-Steuerelementen

iGoogle (Portal) / Google Sites

Google Maps Mashups



Google Maps API: Overlays

- Hinzufügen von zusätzlichen Markierungen
  - var point = new GLatLng(lat,long);
  - var marker = new GMarker(point);
  - map.addOverlay(new GMarker(point));

- Hinzufügen von Polygonen
  - var polyline = new GPolyline([
    new GLatLng(37.4419, -122.1419),
    new GLatLng(37.4519, -122.1519),
    new GLatLng(37.4619, -122.1819)
    ], "#FF0000", 10);
  - map.addOverlay(polyline);

- Anzeige von Zusatzinformationen
  - GEvent.addListener(marker, "click", function()
    { marker.openInfoWindowHtml("Infotext"); });

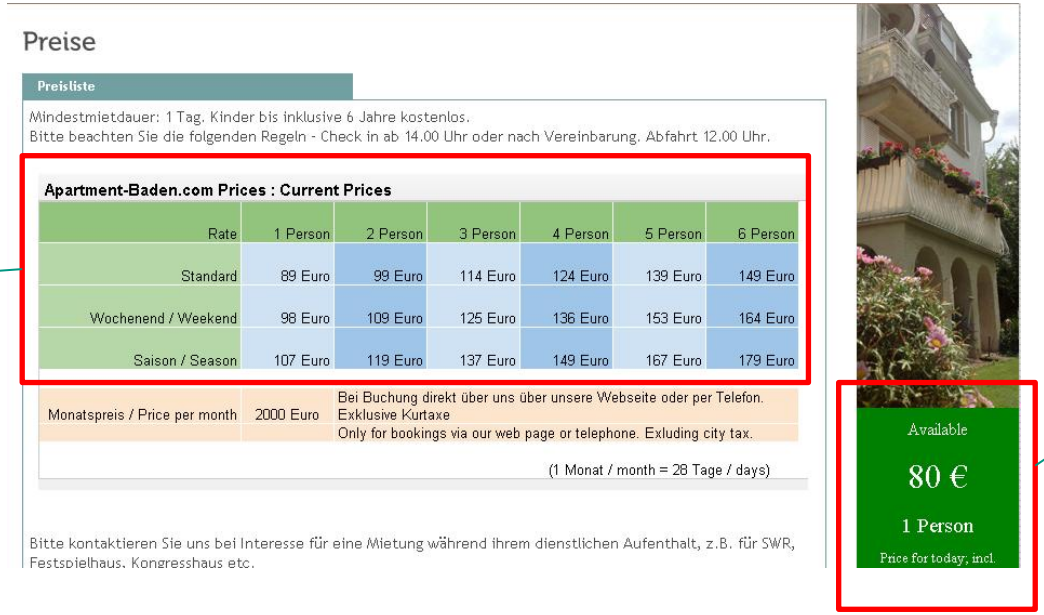
Cloud Mashups (mit Google Sites)

Cloud als Datenbasis / Businesslogik (Implementiert z.B. in JAVA / JAVA-Script)

Beispiel: Google Apps Service

- Integration von Cloud-Elementen in Portal, Web-Seite.
- Beispiel: www.apartment-baden.com
  - Webseite als Darstellung, Mashup von Seitenparts, z.B. Verfügbarkeit . Sheets / Docs / PDFs (Zentrale Aktualisierung)

Published Google Sheet



Published APP based on Google Scripts

Fazit: Leichtgewichtige Ansätze

- Leichtgewichtige, Web 2.0-inspirierte Dienste stellen kaum Hürden an Entwickler
  - keine komplexen Technologien zu erlernen
  - keine speziellen Frameworks
  - oft sehr viel performanter als Web-Service-Lösungen

- Allerdings: sie sind auch in ihrem Einsatzbereich eingeschränkt
  - oft fehlen Validierungs-, Authentifizierungs- und andere „Enterprise-Level“-Funktionen

Literatur

- SOAP, WSDL, UDDI
  - Graham et al., Building Web Services with Java, SAMS Publishing, 2002
- JAX-WS
  - http://jax-ws.java.net/BPEL4WS
- Amazon-Webservices
  - http://aws.amazon.com/de/
- Google Apps Scripts
  - https://developers.google.com/apps-script/