Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Web Services

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Motivation zum Einsatz von Web Services
- Anforderungen und aktuelle Basistechnologien
  - Transport
  - Kodierung der Daten
  - Suche und Zugriff auf Web Services
  - Sicherheit und Zuverlässigkeit von WS

Motivation zum Einsatz von Web Services

Bisher betrachtete Technologien für Webanwendungen

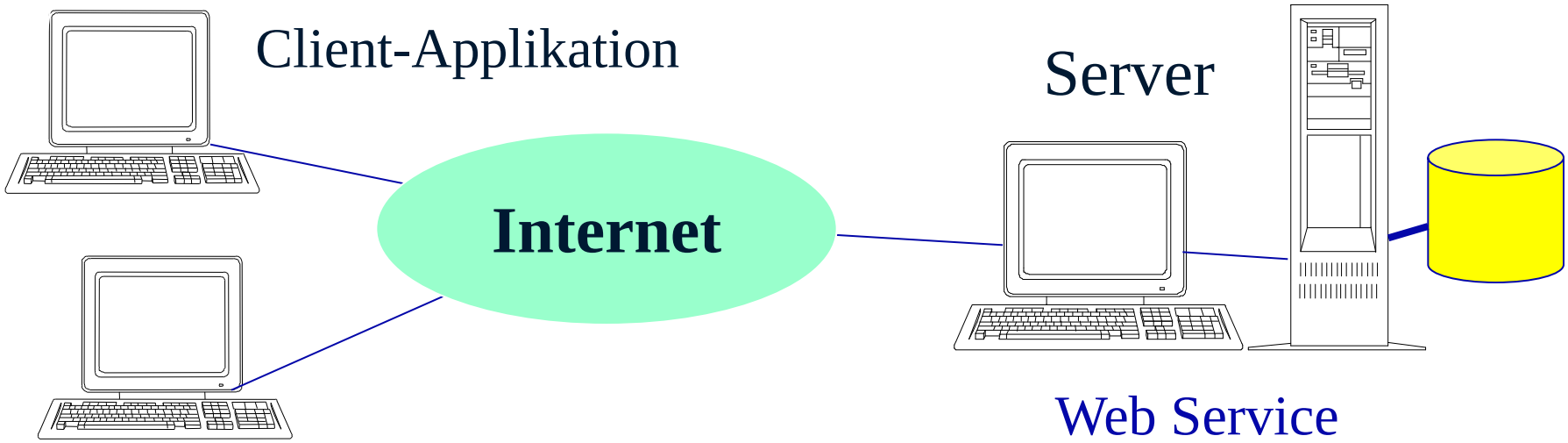
- sind ausgerichtet auf Bedienung oder Anwendung durch Menschen
  - die verwendeten Technologien und Protokolle können zwar prinzipiell auch durch Programme verwendet werden, dies erfordert jedoch einen relativ hohen Aufwand (Simulation eines Bedienvorganges -> Posten eines Formulars -> Extraktion der Ergebnisdaten aus der Ergebnis-HTML-Seite bei altem Problem der fehlenden Content-Kodierung von HTML-Inhalten)
- bereits vorhandene Technologien für verteilte Softwareanwendungen (CORBA / COM) haben (und werden) sich in der Breite NICHT durchsetzen können
  - bisher verwendete Technologien wie RPC waren meist herstellerspezifisch , CORBA wurde vom Microsoft NICHT unterstützt, während MS-Technologien wie ActiveX von den anderen Herstellern nicht unterstützt wurden
- ein neuer Ansatz zur Realisierung verteilter Anwendungen erscheint notwendig
- Web Services als Überbegriff für webbasierte Informationsdienste

Potentiale von Web Services :

- direkte Kopplung von Softwareanwendungen über Web Services erlaubt Datenaustausch über IT- und Firmengrenzen hinweg
- viele verteilte Informationsprozesse könnten besser automatisiert werden :
  - Anfragen und Buchungen von Reisen (Hotels, Flugtickets, Mietwagen)
  - Anfragen und Verhandeln von Angeboten und Bestellungen (drastische Rationalisierung im Einkauf möglich !!!)

Allgemeine Anforderungen an Web Services

- als Client treten Programme oder komplexe IT-Systeme auf



Anforderungen (mit Hinweis zur aktuellen Lösung)

1. Vereinbarungen zum Transport der Daten (XML -> SOAP)
2. Kodierung der Daten (XML -> SOAP)
3. Protokoll zum Zugriff auf Web Service (HTTP -> SOAP)
4. eindeutige Beschreibung eines Web Services (-> WSDL)
5. Firmenübergreifendes Suchen und Verwalten von Web Services (-> UDDI - siehe www.uddi.org)

Überblick zum aktuellen Stand von Web Services

Allgemeines

- im Gegensatz zu bisherigen Entwicklungen (Browser / HTML / ..) ist der Bereich Web Services durch eine relativ konstruktive und offene Zusammen-arbeit aller maßgeblichen IT-Firmen geprägt (Microsoft / IBM / SUN/ Oracle / HP / SAP / Software AG / ...)
- die bislang definierten Entwürfe und Standards basieren generell auf den Basistech-nologien (TCP/IP, http , XML, ..) des Internets und sind offen für alle Anwender
- Die wichtigste Eigenschaft der Entwürfe ist, daß KEINE STRIKTEN VORGABEN zur Verwendung bestimmter Technologien gemacht werden
- aktueller Haupttrend sind Web Services auf der Basis von SOAP – dieses Protokoll soll nachfolgend auch vorgestellt und diskutiert werden
- aufgrund der noch laufenden Entwicklung haben alle nachfolgenden Aussagen eine Gültigkeit von nur wenigen Monaten bis maximal 2 Jahre und sind deshalb bei späterer Verwendung unbedingt noch einmal kritisch zu überprüfen

Quellen:

- http://www.w3.org/2000/xp/Group/
- http://www.uddi.org/
- http://www.w3.org/DesignIssues/Architecture

SOAP

Grundkonzeption

- SOAP stand zu Beginn für Simple Object Access Protocol (heute nicht mehr ganz dieser Abk. entsprechend)
- ist zur Beschreibung von Eingangsdaten und Ergebnisdaten von Webservices und zur Übertragung derselben konzipiert
- basiert auf XML und kann mit XML-Werkzeugen generiert und analysiert werden

Historie:

- erstmals vorgestellt 1999
- Hauptinitiatoren: Microsoft, IBM, W3C
- aktuelle Version: 1.2 (vom April 2007)

  unter http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

SOAP – Grundgerüst

SOAP-Nachrichten bestehen aus

- einem Gesamtumschlag (Container) kodiert durch Tag <Envelope>
- den Kerninformationen im Tagbereich <Body> (muß existieren)
- Zusatzinformation in <Header>
- und optionale Fehlernachrichten über das Tag <Fault>

Das resultierende SOAP-Grundgerüst :

```
<message: Envelope
xmlns: message=" http:// www. w3. org/ 2001/ 12/ soap- envelope"
message: encodingStyle=" http:// www. w3. org/ 2001/ 12/ soap- encoding">
<message: Header>
   ....
</ message: Header>
 <message: Body> ...
   ...
      <message: Fault>  ... <message: Fault>

  </ message: Body>
</ message: Envelope>
```

SOAP-Headerinformationen

- der SOAP-Header definiert Rahmenbedingungen und Verarbeitungshinweise
  - das actor-Attribut gibt einen Verarbeitungshinweis an (hier ..checkin)
  - Das mustUnderstand-Attribut definiert, ob der Partner den Tag kennen und verstehen (=verarbeiten) können muß

```
<message: Header>
 <info: country xmlns: info=" http:// www. europa. org/ country/">
 <info: id actor=" http:// www. europa. org/ country/ checkin/">
 GE- NRW- 48163
 </ info: id>
 <info: language message: mustUnderstand=" 1">
   GE</ info: language>

 <info: currency message: mustUnderstand=" 0">
  EURO</ info: currency>

</ message: Header>
```

SOAP-Body

- der SOAP-Body kodiert die eigentlichen Daten
  - im Beispiel fragt der Client den Server "Einhaendler" nach dem Preis eines Artikels
  - als Namespace (=xmlns) wird der Namespace des Händlers verwendet

```
<message: Body>
<preis: PreisAuskunft
      xmlns: preis=” http//www.Einhaendler.de/preise/”>

<preis: Artikel> NOKIA 8210</ preis: Artikel>
<preis: Artikel> SINUS 710K</ preis: Artikel>

</ preis: PreisAuskunft>

<message: Body>
```

SOAP-Antwort und Fehlerinformationen

- die Antwort wird ebenfalls wieder als XML-Nachricht kodiert
- Fehlerinformationen können als Code oder im Klartext (Problem Sprach-abhängigkeit -> deshalb Sprachangabe bei Request) zurückgegeben werden

```
<message: Body>
<message: Body xmlns: response=" http// www.einhaendler.de/ preise/">
<response: PreisAuskunftAntwort">
        <response: Artikel> NOKIA 8210</response: Artikel>
        <response: Preis>239</response: Preis>
</response: PreisAuskunftAntwort>

<message: Fault>
<faultcode> message: Artikel</faultcode>
 <faultstring> Unbekannter Artikel</faultstring>
 <detail>
   Der von ihnen angegebene Artikel SINUS 710K ist uns
   nicht bekannt
 </detail>
 </message: Fault>

</message: Body>
```

Transport von SOAP-Nachrichten

Übertragung von SOAP-Nachrichten über alle Internetprotokolle :

- am Häufigsten über http – Port 80 (zur Umgehung von Firewalls)
- auch Versand über SMTP/POP3/IMAP
- FTP prinzipiell ebenfalls möglich (größere Dateien für Batchverarbeitung ?)

```
HTTP/ 1.1 200 OK
Connection: close
Content- Type: application/ soap; charset= utf- 8
Date : Tue, 28 May 2016 05: 28: 03 GMT
< ?xml version=" 1.0"?>
<message: Envelope
xmlns: message=" http:// www. w3. org/ 2001/ 12/ soap- envelope"
message: encodingStyle=" http:// www. w3. org/ 2001/ 12/ soap-
encoding">
<message: Body xmlns: response=" http// www. einhaendler. de/
preise/">
<response: PreisAuskunftAntwort">
<response: Preis> 239</ response: Preis>
</ response: PreisAuskunftAntwort>
... </ message: Body>
</ message: Envelope>
```

Beschreibung von Web-Services

Verfügbare Web Services werden beschrieben durch die

Web Service Description Language (WSDL)

WSDL definiert :

- die Nachrichten, welche ausgetauscht werden,
- wie sie ausgetauscht werden,
- wo der Service zu erreichen ist und
- mit welchem Protokoll.

Bestandteile einer WSDL-Definition sind:

- Datentypdefinitionen für den Datenaustausch (<types>)
- Nachrichtendefinitionen (<message>)
- Porttypes zur Beschreibung der abstrakten Kommunikationsart zwischen den Partnern (One-way, Request-response , Solicit-response, Notification)
- Bindings zur konkreten Definition des Austauschprotokolls (<bindings>)
- Services zur Zusammenfassung von mehreren Ports (<services>)

WSDL-Beispiel zu Datentypen und Messages

```
<?xml version="1.0"?>  <definitions name="StockQuote"
    targetNamespace="http://example.com/stockquote.wsdl"   ...
<types><schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema"
 <element name="TradePriceRequest">
 <complexType> <all><element name="tickerSymbol" type="string"/></all>
 </complexType> </element>
<element name="TradePrice">
<complexType> <all>  <element name="price" type="float"/> </all>
 </complexType> </element> </schema> </types>

<message name="GetLastTradePriceInput">
<part name="body" element="xsdl:TradePriceRequest"/></message>

 <message name="GetLastTradePriceOutput">
 <part name="body" element="xsdl:TradePrice"/></message>
```

WSDL-Beispiel zu Ports und Bindings

```
.... <!– Code von vorheriger Seite -->
<portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
      </operation>
 </portType>

<binding name="StockQuoteBinding" type="tns:StockQuotePortType">
 <soap:binding style="document"
              transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="GetLastTradePrice">
   <soap:operation
    soapAction="http://example.com/GetLastTradePrice"/>
      <input>   <soap:body use="literal"/>  </input>
      <output> <soap:body use="literal"/>  </output>
 </operation>
```

Zeitliche Charakteristik des Transportes

Web Services können zeitlich wahlfrei übertragen werden :

- **synchron**, d.h. es wird nach dem Versand auf eine Rückantwort gewartet
- oder **asynchron**, es erfolgt nur ein Versand ohne Warten auf eine Rückantwort
  - Vorteile : damit sind auch Kommunikationsformen wie Email möglich

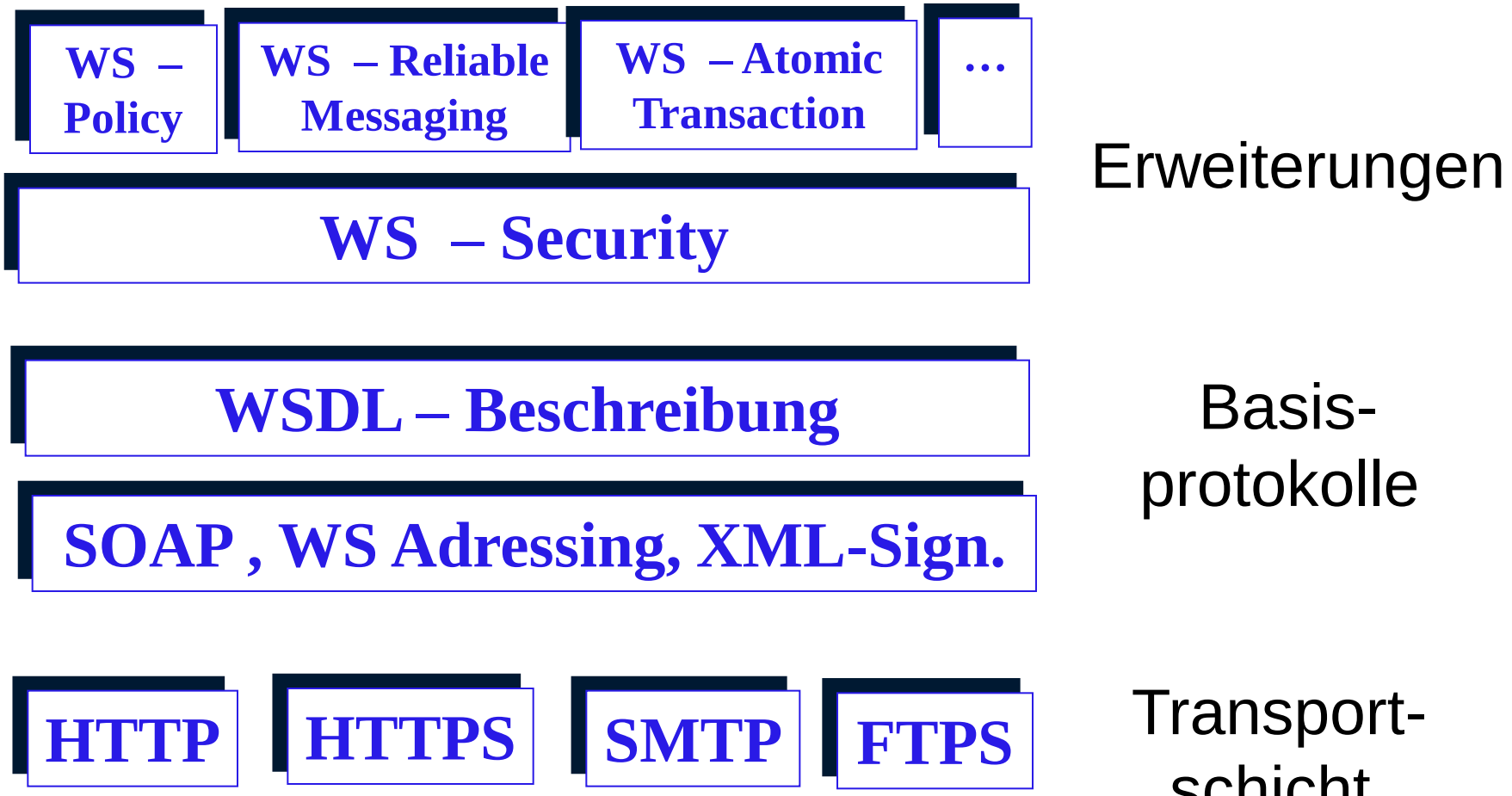
Probleme:

- Das http-Protokoll kann nur synchrone Übertragungen durchführen, eine asynchrone Übertragung kann durch Ignorieren der Rückantwort und späteren Callback (wie realisieren ? - > Polling ???)
- mit **WS-Adressing / WS-Webservice Notification** sind entsprechende Standards zur Lösung verfügbar
- andere Übertragungsprotokolle wie JMS oder Websphere MQ ((IBM) sind bereits auf asynchrone Übertragungen vorbereitet.

SOAP-Sicherheit

Im Basisstandard von SOAP sind noch keine Sicherheitsmaßnahmen definiert (Kommunikation kann abgehört / gefälscht werden)

- Zusätzliche Standards und Maßnahmen zur Absicherung auf der Ebene der Basisprotokolle und durch zusätzliche Standards :



SOAP-Sicherheit - Zusätzliche Standards – Überblick

Es sind über 150 Webservice-Erweiterungsstandards definiert !

- Komplette Liste siehe :

  http://www.oasis-open.org/specs/index.php#wssv1.0

- Auch kommentierte Liste unter

  http://de.wikipedia.org/wiki/WS-*

- **Management durch OASIS -** Organization for the Advancement of Structured Information Standards
  - internationale, nicht-kommerzielle Organisation
  - außer E-Business- und Web-Service-Standards auch Standardisierung von OpenDocument und DocBook

Generelle Aufgaben

- Zugriffskontrolle auf WS - „Wer darf auf welche WS zugreifen ?“
- Vertraulichkeit und Integrität von WS-Inhalten – (Abhören / Fälschen)
- Zuverlässigkeit der Kommunikation

SOAP-Sicherheit - Absicherung auf Transportschicht

Absicherung über die Transportebene

- prinzipiell sind alle bekannten Verfahren wie SSL oder TLS nutzbar zur Absicherung von Punkt zu Punkt-Verbindungen
- Probleme bei Zwischenstationen (Man in the Middle-Attacke) 2
- Eine abgestufte Sicherung (nur Msg-Teile) ist nicht möglich !
- Probleme ergeben sich bei asynchroner Kommunikation, d.h. der Absender WARTET NICHT auf Empfangsquittung
  - teilweise auch Offline-Modus über Queues oder Email-Server
  - damit sind Verschlüsselungsverfahren, welche auf einen Schlüsselaustausch zwischen den Stationen basieren NICHT möglich !
  - Auswege
    - Zusätzliche Dienste zum Schlüsseltausch (widerspricht aber loser Kopplung)
  - Probleme damit auch bei https (http Status OK 200 als Quittung)
- Bei synchroner Kommunikation ergeben sich durch Schlüsseltausch i.d.R. zusätzliche Laufzeiten.

SOAP-Sicherheit - WS- Security

WS-Security ist eine Erweiterung des SOAP-Standards

- es ist eher als Rahmen für die Einbettung weiterer Verfahren zur Absicherung von Webservices zu sehen
- beinhaltet bzw. regelt den Einsatz von
  - Verschlüsselung einzelner Datenbereiche mit **XML-Encryption**
  - Signatur der Daten mit **XML-Signature**
  - Anforderungen zur Authentifikation mit **WS-SecurityPolicy**
  - Regelung des initialen Austauschs von sicherheitssensitiven Daten mit **WS-Trust** (Abgleich der verfügbaren Verfahren, Schlüssel , ...)
  - **WS-SecureConversation** und **WS-Federation** zum Aufbau von größeren Vertrauensdomänen (z.B. mehrere Firmennetzwerke mit einer großen Anzahl von Stationen …)
  - **SAML** als weiterer Standard zur Beschreibung komplexer Berechtigungen

SOAP-Sicherheit - XML-Encryption

- Verschlüsselung auch von Teilbereichen
- meist wird ganzes Body-Element der Nachricht verschlüsselt :

```
<env:Body> <enc:Encypteddata ID=„bodyID“ >
<EncyptionMethod
   Algorithm=“http://www.w3.org/2001/04/xmlenc#tripledes-cbc“>
<IV>oephdzesgdh</IV> /EncyptionMethod>
<ds:Keyinfo> <ds:Keyname> Priceinfo12</ds:Keyname></ds:Keyinfo>
<xenc:CipherData><xenc:CipherValue>
          dGHEUHDKKSKK …
    </xenc:CipherValue></xenc:CipherData>
```

- mögliche Probleme :
  - durch Verschlüsselung des Dokumentbody´s wird die XML-Struktur nicht mehr eingehalten (Probleme bei Validierung ohne Decod.)
  - bei zu kleiner Datenmenge (z.B. nur einzelne Boolesche Werte) sind Angriffe durch Erraten auf Basis der beiden möglichen Wert denkbar

SOAP-Sicherheit - XML-Signature

- wird in der Regel für einen Teilbaum der Nachricht (meist Body) durchgeführt , i.d.R. Secure Hash Alg. (kein MD5!)
- Bei Einbeziehung der Metadaten in Signatur ggf. Problem durch Ablage der Signatur in Metadaten selbst …
- generelles Problem mit formatfreier Struktur von XML-Daten -> Parser können White-Spaces der XML-Daten weglassen oder verändern -> mehrere, gleiche Dokumentversionen bei unters. Signatur !
- Lösung : Rückführung auf einheitliche Struktur – ***Canonical XML***
- Bsp.:

```
<ds:Signature xmlns="…/xmldsig" ><ds:SignedInfo>
   <ds:CanonicalizationMethod  Algorithm="…/xml-exc-c14#" />
<ds:SignatureMethod Algorithm="…/xmldsig#hmac-sha1" />
<ds:DigestValue>HuKhsjalswa</ds:DigestValue>
```

SOAP-Sicherheit - WS- Policy

WS-Policy dient zur Formulierung von Sicherheitsrichtlinien :

- meist Liste von verfügbaren Verfahren zur Authentifizierung und Verschlüsselung
- durch Preference-Wert ist eine Priorisierung bei mehreren verfügbaren Verfahren möglich:
- Bsp.:

```
<wsp:Policy xmlns:wsse="…"   xmlns:wsp="…" >
 <wsp:ExactlyOne>
    <wsp:SecurityToken wsp:Preference="100" >
       <wsse:Tokentype>wsse:keberosv5TGT </wsse:Tokentype>
    </wsp:SecurityToken>
    <wsp:SecurityToken wsp:Preference="20" >
       <wsse:Tokentype>wsse:X509v3</wsse:Tokentype>
    </wsp:SecurityToken>
 </wsp:ExactlyOne> </wsp:Policy>
```

SOAP-Sicherheit - SAML

SAML – Security Assertion Markup Language

- ist ein weiterer Standard, welcher unabhängig von WS von OASIS definiert wurde, sich jedoch für WS gut eignet
- WS-Security wurde daher auch zur Nutzung von SAML spezifiziert

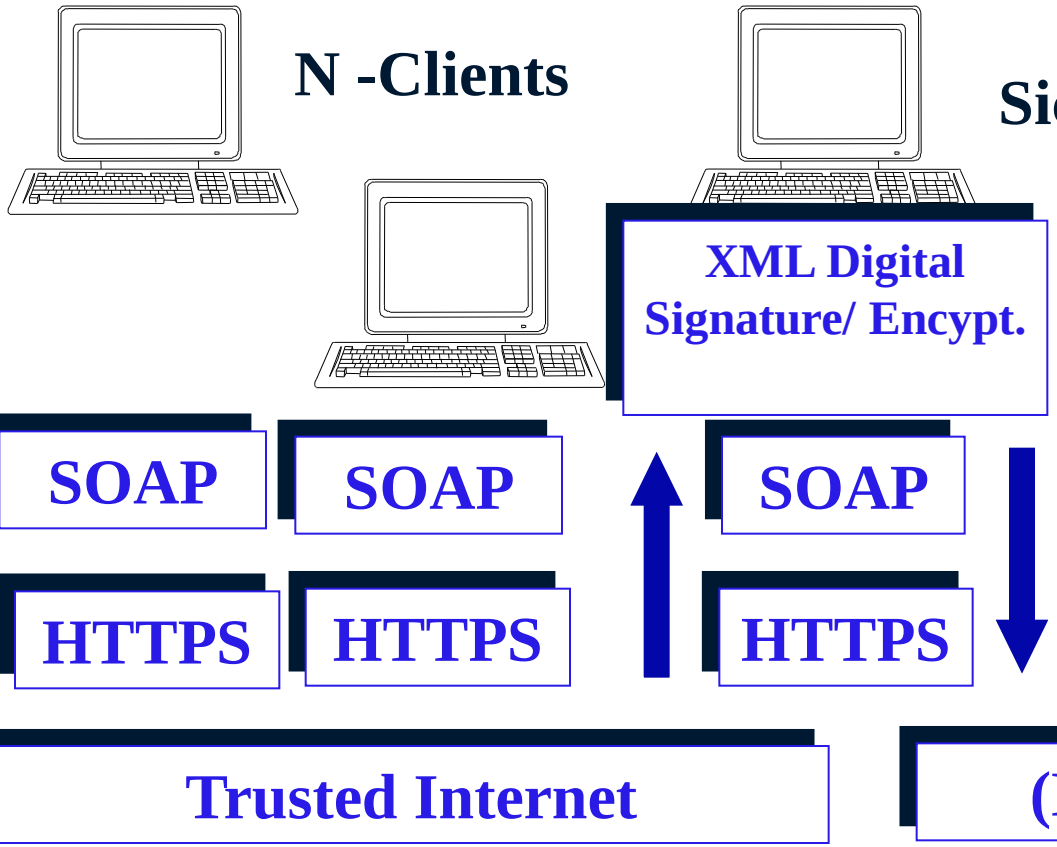
Assertions

- beinhalten Informationen zur Authentifizierung in der Form „es lag eine gültige ID vor“, die sicherheitssensitive ID selbst wird nicht mehr mitgeführt
- Unterschiedliche Typen von Assertions
  - Authentification Assertions - Zugriffsberechtigungen auf Ressourcen
  - Attribut –Assertions – Zuordnung von Rollen-Attributen
  - Decision – Assertions – Definition von Bedingungen, unter welchen auf Ressourcen zugegriffen werden kann

SOAP-Sicherheit - Typische Infrastruktren

Bei einer größeren Anzahl von Clients ist der Gesamtaufwand relativ hoch (Zertifizierungen, Signatur-Services, …)

- eine gestaffelte Sicherheitsarchitektur ist daher sinnvoll :



Proxy mit Sicherheitsfunktionen

- Prüfung auf Berechtigungen
- Verschlüsselung
- Signatur

(relativ transparent für Clients (ohne entspr. Code in Clients))

Webservice – Transaktionen

Aus dem DB-Bereich ist zur Absicherung von Transaktionen das ACID-Prinzip bekannt :

- Atomarität : Operationen einer Transaktion werden entweder ganz oder gar nicht durchgeführt
- Consistency (Konsistenz) : immer Übergang von einem konsistenten Zustand in einen anderen konsistenten Zustand
- Isolation : isolierte (unabhängige) Ausführung der Transaktionen
- Durabilität : dauerhafte Speicherung der Daten nach Abschluss

Für die professionelle Anwendung von WS sind genau die gleichen Anforderungen zu erfüllen, jedoch ist ACID nicht immer optimal :

Probleme durch verteilten Charakter von WS :

- Atomarität : Rollback erfordert hohen Kommunikationsaufwand
- Konsistenzabsicherung erfordert ebenfalls viele Bestätigungs-Msg.
- Isolation führt zu hohen Sperr-Raten (durch WS-Laufzeiten werden die zugrunde liegenden DB deutlich länger als nötig gesperrt !)
- Durabilität in verteilten Systemen erfordert das Führen von entsprechend vielen Logdateien

Webservice – Transaktionen - spezielle Lösungen

Verteilte Transaktionen mit 2-Phasen Commit-Protokoll :

Ein zentraler Koordinator steuert die verteilte Transaktion in 2 Schritten:

Phase 1 :

- Vorbereitungsphase : Jeder Teilnehmer wird mit „Prepare“-Nachricht aufgefordert, seine Arbeitsschritte zu beenden, je nach Ergebnis antwortet der Teilnehmer mit „Prepared“ oder „Cancel“

Phase 2

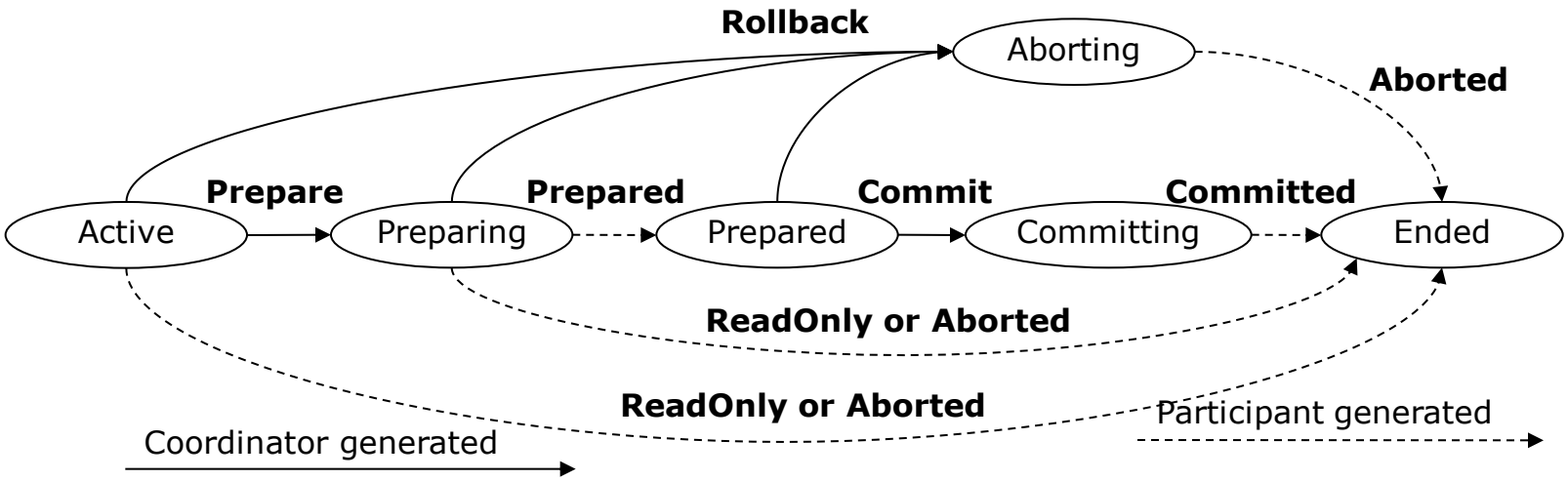
- Nach Vorliegen aller Rückmeldungen entscheidet der Koordinator über den Erfolg der Transaktion. Bei Erfolg wird an alle Teilnehmer ein „Commit“ versendet, womit die Transaktion als endgültig abgeschlossen definiert wird.
- Bei Misserfolg wird bei Forderung nach Atomarität ein „Abort“ gesendet.
- Falls keine Forderung nach Atomarität besteht, kann der Koordinator über eigene Geschäftsregeln über den Erfolg entscheiden (z.B. bei fehlendem Erfolg bei Statistikmodul trotzdem buchen ...)

Webservice – Transaktionen - verfügbare Standards

- Vorarbeiten von IBM, Microsoft und BEA am WSTF (WS Transaction Framework)
- im April 2009 wurde durch OASIS der Web Services Transaction (WS-TX) TC - Standard verabschiedet

Bestandteile :

- WS-Coordination v1.2 – definiert Kommunikationskanäle zur Steuerung und Koordination der verteilten Transaktion
- WS-AtomicTransaction v1.2 – definiert Zustände und Nachrichten
- WS-BusinessActivity v1.2 – noch feinere Untersetzung der Zustände



Quelle : WS-AtomicTransaction v1.2 Standard - Completion Protocol – OASIS

UDDI

Verfügbare Webservices werden verwaltet durch :

Universal Description, Discovery and Integration (UDDI)

- Entwicklungsbeginn September 2000
- Begründer : Microsoft, Ariba, IBM etc. (heute über 200 Firmen)
- Achtung : 2005 haben diese Firmen ihre bislang öffentlichen UDDI-Server wieder abgeschaltet und es ist bisher auch keine neuer Server verfügbar
  (vgl. http://uddi.xml.org/public-uddi-registry)
- aktuell (2008) Liste von Webservices : http://seekda.com/ (kein UDDI)

UDDI-Überblick

- eine Programmers-API spezifiziert ca. 30 Methoden zum Beschreiben, Suchen und Publizieren von Web-Services
- Kommunikation komplett über SOAP-Nachrichten
- UDDI-Operationen sind sehr einfach gehalten:
  - Keine logischen Operatoren
  - Keine kombinierten Anfragen
  - Rudimentäre Infrastruktur
  - Entwickler gehen von einer Integration in intelligentere Systeme aus

UDDI-Infrastruktur

Ziel ist Schaffung eines unabhängigen Firmen- und Diensteverzeichnisses :

Gegenwärtig 3 Teile:

1. **” white pages“** - mit Adresse, Kontaktinformationen und Ansprechpartnern
2. **” yellow pages“** mit einer Einordnung des Unternehmens nach einheitlichen Richtlinien gemäß Geschäftsbereich
3. **” green pages“** mit technischen Informationen über die angebotenen Dienstleistungen und Verweisen zu genauen Spezifikationen angebotener Web Services oder anderen Informationsquellen, soweit erforderlich

UDDI-Geschäftsverzeichnis ist

- logisch zentralisiert, aber physikalisch verteilt auf verschiedene Knotenpunkte, die sich ständig gegenseitig auf dem Laufenden halten.
- soll jederzeit für jeden frei zugänglich sein.
- Bis ca. Ende 2006 wurden 2 zwei Knotenpunkte von Microsoft und IBM betrieben, danach nur noch dezentrale (Firmen-) Verzeichnisse

UDDI-Daten

Die UDDI-Daten untergliedern sich in :

- **businessEntity:** alle bekannten und relevanten Informationen über ein Unternehmen und dessen angebotene Leistungen
- **businessService**: Beschreibung eines bestimmten Service
- **bindingTemplate**: Informationen über den Zugriffspunkt eines Service (z.B. URL) und den Verbindungsaufbau
- **tModel**: genaue Beschreibung einer technischen Spezifikation, die für einen bestimmten Service gebraucht wird
- **publisherAssertion**: Informationen über die Beziehung zwischen zwei Unternehmen, die im UDDI Verzeichnis eingetragen sind (optional, z.B. Mutter - Tochter)
- Details und Beispiele unter : **http://uddi.xml.org/**

Alternative Transportoptionen - XML –RPC

- SOAP ist aufgrund seiner vielen Freiheiten speziell für kleinere Kommunikationsaufgaben nicht immer die optimale Lösung
- mit XML-RPC (XML-Remote-Procedure-Call) können insbesondere entfernte Methodenaufrufe schneller implementiert werden :
- entwickelt 1998 von Dave Winner als „einfachere“ Alternative zu SOAP
- generell synchroner Versand über http (keine anderen Verfahren)

- im Gegensatz zu den ca. 40 Datentypen von SOAP (aus XML-Schema) unterstützt XML-RPC nur 6 Datentypen (string / integer / double / boolean / date / binary) + davon abgeleitete Strukturen und Array´s
  - dies wird jedoch auch wieder kritisiert, da damit automatische Konvertierungen komplexerer Datentypen wie z.B. URL nicht möglich sind -> nur über Strings abbildbar und damit Verlust an Information
- Transportcodierung ähnlich wie SOAP

Alternative Abfrageoptionen - REST

REST - (REpresentational State Transfer)

- nach einer Dissertation von Roy Fielding von 2000 als neues Architekturkonzept definiert (keine Technologie)
- **jede Web-Ressource soll eindeutig über eine URI identifiziert werden**
- **Es können außer XML-Daten auch andere Formate verwendet werden !**
- durch Aufrufe unterschiedlicher URL´s ändert quasi auch der Client seinen Zustand -> „State Transfer“ - jedoch ohne Gedächtnis !
- auch Änderungen werden per POST (oder ggf. auch PUT) durchgeführt
- Beispiel: Online-Angebot von Amazon per WS: http://aws.amazon.com/

```
url = "http://webservices.amazon.de/onca/xml? Service=
    AWSECommerceService&AWSAccessKeyId=".APIKEY .
    AssociateTag=".PARTNERID "&Operation=ItemLookup&ItemId=".$isbn .
    "&ResponseGroup=Large";
```

Vorteile:

- REST ist eine interessante Alternative zu SOAP speziell für sehr kleine Services (z.B. für Börsenkurse / Wetterdaten etc.)
- REST kann als eine Art Rückbesinnung auf grundlegende Web-Techniken verstanden werden : „einfacher Aufruf per Link“

Alternative Abfrageoptionen - REST - Beispiel

Allgemeines Format von REST-Anfragen

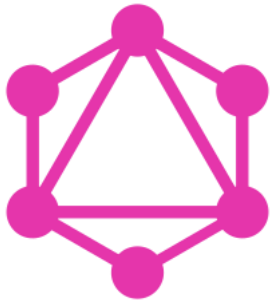
- **Protocol://ServiceName/ResourceType/ResourceID**
- Protocol: in der Regel http
  mit den Methoden GET, POST, PUT, DELETE
- Servicename = Serveradresse
- ResourceType oder Operation – möglichst verständliche Bezeichnung
- ResourceID- eindeutige ID für die Ressourceninstanz
- Beispiel: http://server123/customerread/322
  - **der Request entspricht einem http-GET**
  - Mit der gleichen Syntax können auch weitere Operationen definiert werden:
    http://server123/customerdelete/322
    http://server123/customerrename/324?nachname=Schulze

  - Alternativ auch http://server123/customers?customerid=322
  - **Optional können auch Formatanweisungen angegeben werden**
    http://MyService/Persons/1?format=xml&encoding=UTF8
    http://MyService/Persons/1?format=json&encoding=UTF8

Alternative Abfrageoptionen - GraphQL

Motivation für weitere Ansätze

- sowohl WS wie auch REST definieren genau eine ganz spezifische Anfrage
- Schnelle Adhoc-Anfragen im Rahmen der Entwicklung mit leicht geänderten Parametern sind kaum möglich oder müssen bereits bei der Implementierung genau geplant werden



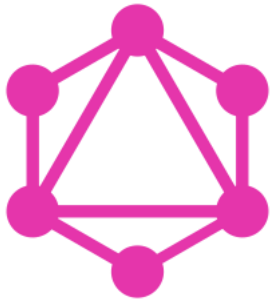
GraphQL als neuer flexiblerer Ansatz

- gestartet als interne Entwicklung von Facebook zur effizienten Entwicklung mobiler Apps
- Sourcecode öffentlich verfügbar gemacht 2015
- im November 2018 Gründung einer eigenen GraphQL foundation, (gehostet durch die non-profit Linux Foundation

- **Homepage** graphql.org
- **unter GitHub: github.com/facebook/graphql**

Alternative Abfrageoptionen - GraphQL

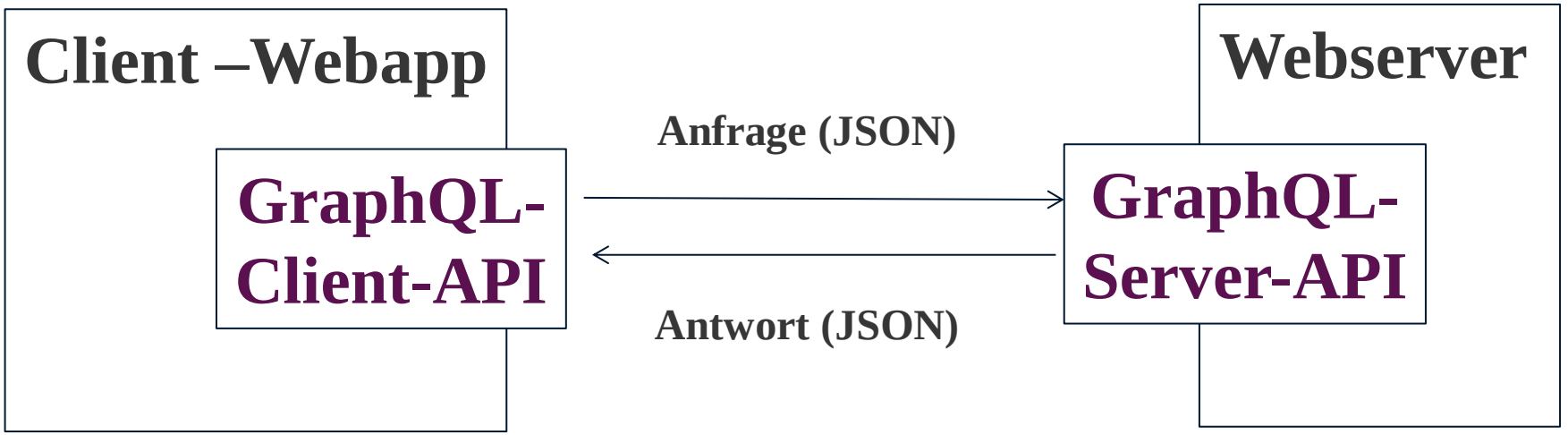
GraphQL – Grundansatz



- GraphQL ist eine API-Implementierung, welche sowohl für den Client (mit JS) und auch den Server in verschiedenen Implementierungen bereitgestellt wird
- Serverseitig werden fast alle bekannten Programmiersprachen unterstützt (C/C++ / Java / JavaScript (NodeJS) :
  - aktueller Stand siehe https://graphql.github.io/code/

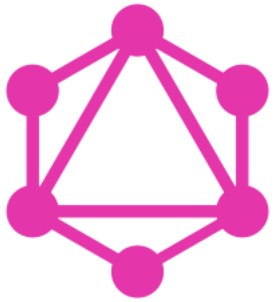
Grundprinzip:

- Über eine JavaScript-Client-Api werden Anfragen im JSON-Format gestellt und die Antwort ebenfalls im JSON-Format zurückgegeben



Alternative Abfrageoptionen - GraphQL

GraphQL – Client API



Neben der Code-basierten Abfrage bietet die GraphQL-Client-API auch eine vordefinierte grafische Oberfläche an, welche zum Austesten der API und zum Aufbau von Requests verwendet werden kann

GraphiQL ▶ Prettify History

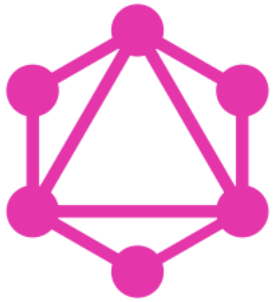
```
query {
  SIMGEN(f:"(x1-2)*(x2-4)*(x3-5)")
    {
    Rpoint
    Rgoalvalue
    }
  MonteCarlo(f:"(x1-2)*(x2-4)*(x3-5)")
    {
    Rpoint
    Rgoalvalue
    }

}
```

QUERY VARIABLES

```
{
  "data": {
    "SIMGEN": {
      "Rpoint": [
        10,
        9.91,
        0.02
      ],
      "Rgoalvalue": -235.7852
    },
    "MonteCarlo": {
      "Rpoint": [
        9.29,
        8.53,
        0.85
      ],
      "Rgoalvalue": -137.2334
    }
  }
}
```

GraphQL - Abfrageoptionen



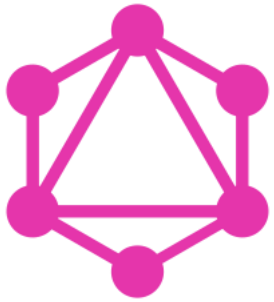
Verfügbare GraphQL – Abfragetypen

```
#  einfache Query auf alle Daten
query {  countries {
                code
                name   }
}
```

```
#  Query mit Argument
query {   country (  code : "DE" )
  {  code
    name
    phone
  }
}
```

In ähnlicher Weise können mit Query-Fragmenten mehrere Abfragen gleichzeitig gestartet werden und mit Mutations auch schreibende Aktionen durchgeführt werden.

GraphQL - Serviceangebote



Verfügbare GraphQL – Services

Analog zum WS –UDDI-Verzeichnis gibt es auch erste offizielle GraphQL-Angebote
(z.B. unter http://apis.guru/graphql-apis/)

- **Abfrage von Bahnhofsdaten bei der Deutschen Bahn**
  https://bahnql.herokuapp.com/graphql
- **Abfrage von Länderinformationen**
  https://countries.trevorblades.com/
- **GitHub – Abfagen**
  https://developer.github.com/v4/explorer/

Web-Services und alternative API´s – Zusammenfassung

WebServices -

- große Flexibilität, relativ gut beherrschbar
- gut lesbar, aber relativ umfangreiche Dateien , ggf. Performanceprobleme durch XML-Klartextdateien (Ausweg Binär-XML)
- breites Angebot an XML-Werkzeugen (XML-DB, freie Parser)
- gute Unterstützung durch die meisten Programmiersprachen (fertige Java- und C++-Bibliotheken)
- sehr große Anzahl von Erweiterungsstandards (vgl. WS-Security)
- WebServices und SOAP sind Industriestandard

REST –Web-Services

- geringerer Datenoverhead
- Schnelleres Testen und Implementieren möglich

GraphQL

- Leichtes Austesten der Schnittstelle online möglich
- sehr flexibel und effizient (nur die benötigten Daten anfordern)

Langfristig wahrscheinlich Übergang zu GraphQL-ähnlichen API´s