SOAP

© Klaus Schild, 2004

Lernziele

- Aufbau und Verarbeitung von SOAP-Nachrichten
- SOAP über HTTP
- SOAP-Engines
- Vor- und Nachteile von SOAP

© Klaus Schild, 2004

Was ist SOAP?

- XML-basiertes Nachrichtenformat zur Übertragung von strukturierten und typisierten Daten:
  - Dokumente (Messaging)
  - entfernte Prozeduraufrufe (RPCs)
- einfach erweiterbar
- zustandslose Einweg-Kommunikation
- mit allen gängigen Internet-Protokollen übertragbar

© Klaus Schild, 2004

Aufbau von SOAP-Nachrichten

© Klaus Schild, 2004

Prinzipieller Aufbau

- Envelope: umschließt die eigentliche Nachricht (Body) und den Briefkopf (Header)
- Body: Nachrichteninhalt
- Header: optionaler Briefkopf enthält Zusatzinformationen

© Klaus Schild, 2004

XML-Syntax

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
      xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    Zusatzinformationen
  </env:Header>
  <env:Body>
    Nachrichtinhalt
  </env:Body>
</env:Envelope>
```

- Wurzel-Element: Envelope aus dem SOAP-Namensraum
- Beachte: kein W3C-Namensraum
- Header: optional
- Body: obligatorisch

© Klaus Schild, 2004

Eine SOAP-Anfrage an Google

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
        …
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- doGoogleSearch(key, q, start, maxResults,...)
- doGoogleSearch aus Google-Namensraum
- Datentypen aus XML-Schema

Und die Antwort von Google

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope …>
  <env:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" …>
      <return xsi:type="ns1:GoogleSearchResult">
        …
      </return>
    </ns1:doGoogleSearchResponse>
  </env:Body>
</env:Envelope>
```

- Antwort: doGoogleSearchResponse(return(…))
- Datentyp ns1:GoogleSearchResult *nicht* in SOAP-Nachricht, sondern in WSDL-Beschreibung definiert.

Envelope-Namensraum = SOAP-Version

- SOAP 1.1
- W3C-Note (2000)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env=" http://schemas.xmlsoap.org/soap/envelope">
  …
</env:Envelope>
```

- SOAP 1.2
- W3C-Standard (Juni 2003)

aktuelle Version

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/envelope">
  …
</env:Envelope>
```


Versionen

SOAP 1.1

- kein offizieller W3C-Standard, aber heute noch weit verbreitet
- Google benutzt diese Version

SOAP 1.2

- einzige Version, die vom W3C als Standard offiziell akzeptiert wurde
- Vorlesung benutzt diese Version

Zum Unterschied:

- knapp: http://www.w3.org/TR/soap12-part0, Kapitel 6.
- ausführlich: http://www.hadleynet.org/marc/whatsnew.html

Nachrichteninhalt

```
<env:Envelope …">
        …
  <env:Body xmlns:ns="URI">
    <ns:Nachrichtinhalt-Teil-1>…</ns:Nachrichteninhalt-Teil-1>
        …
    <ns:Nachrichteninhalt-Teil-n>…</ns:Nachrichteninhalt-Teil-n>
  </env:Body>
</env:Envelope>
```

- beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt: z.B. durch speziellen Namensraum oder WSDL-Beschreibung

Briefkopf

```
<env:Envelope …>
        …
  <env:Header xmlns:ns="URI" >
    <ns:Zusatzinformation-1>…</ns:Zusatzinformation-1>
        …
    <ns:Zusatzinformation-n>…</ns:Zusatzinformation-n>
  </env:Header>
  <env:Body>…</env:Body>
</env:Envelope>
```

Header Blocks

- beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt

Erweiterbarkeit durch Header Blocks

```
<env:Envelope …>
  <env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol">
      <priority>1</priority>
      <expires>2003-10-12T14:00:00-05:00</expires>
    </alertcontrol>
  </env:Header>
  <env:Body>
    <alert-msg xmlns="http://example.org/alert">
      Pick up Mary at school at 2pm!
    </alert-msg>
  </env:Body>
</env:Envelope>
```

Zusatz-information (*Header Block*)

Nachricht

© Klaus Schild, 2004

Das mustUnderstand-Attribut

```
<env:Envelope …>
  <env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol"
                  env:mustUnderstand="true">
      …
    </alertcontrol>
  </env:Header>
  <env:Body>…</env:Body>
</env:Envelope>
```

- mustUnderstand="true": Empfänger muss Header Block verstehen, ansonsten Fehlermeldung
- Standard-Wert ist "false": Empfänger muss Header Block *nicht* unbedingt verstehen

© Klaus Schild, 2004

Orthogonale Erweiterungen

```
<Body>
  <Nachricht>
    …
  </Nachricht>
</Body>
```

Nachrichtenformat bleibt unverändert

```
<Header>
  <Zusatz-1>
    …
  </Zusatz-1>
  …
</Header>
```

…

```
<Header>
  …
  <Zusatz-n>
    …
  </Zusatz-n>
</Header>
```

Erweiterungen (optional oder obligatorisch) sind unabhängig voneinander

© Klaus Schild, 2004

Anfrage-Antwort-Muster

- SOAP selbst unterstützt nur Einweg-Kommunikation
- Anfrage-Antwort-Muster können auf verschiedene Weise realisiert werden:
  - mit eindeutiger Referenz im Briefkopf
  - mit HTTP

© Klaus Schild, 2004

Referenz im Briefkopf

- Anfrage enthält eindeutige Referenz („Mein Zeichen"), worauf die Antwort anschließend verweisen kann:

```
<Header>
  <wsu:identifier xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility/">
        URI als eindeutige Referenz
  </wsu:identifier>
</Header>
```

- auf diese Weise beliebig komplexe Interaktionen möglich
- oft verwendet, aber noch *kein* etablierter Standard
- Alternative zu wsu:identifier:
  MessageID aus Namensraum von WS-Addressing

© Klaus Schild, 2004

Struktur entfernter Prozeduraufrufe

Procedure(In-Parameter-1,…,In-Parameter-n)

```
<env:Envelope …>
  <env:Body>
    <m:Procedure xmlns:m="URI">
      <m:In-Parameter-1>…</m:In-Parameter-1>
        …
      <m:In-Parameter-n>…</m:In-Parameter-n>
    </m:Procedure>
  </env:Body>
</env:Envelope>
```

- Name der Prozedur Kind-Element von Body
- Eingangsparameter Kind-Elemente der Prozedur (ungeordnet)
- Beachte: grundsätzlich *Call-by-Value*

© Klaus Schild, 2004

Struktur entfernter Prozeduraufrufe

physischer Aufenthaltsort (URI) der Prozedur:

- *nicht* in SOAP-Nachricht repräsentiert
- außerhalb von SOAP im Transportprotokoll
- kann aber im Briefkopf repräsentiert werden:

```
<wsa:EndpointReference
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        xmlns:gs"urn:GoogleSearch ">
  <wsa:Address>http://api.google.com/search/beta2</wsa:Address>
  <wsa:PortType>gs:GoogleSearchPort</wsa:PortType>
</wsa:EndpointReference>
```

- WS-Addressing *kein* etablierter Standard

Struktur der Antwort

```
<env:Envelope …>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI">
      <m:Out-Parameter-1>…</m:Out-Parameter-1>
        …
      <m:Out-Parameter-n>…</m:Out-Parameter-n>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- *ProcedureResponse* Kind-Element von Body
- Name für *ProcedureResponse* irrelevant
- Rückgabewerte Kind-Elemente von *ProcedureResponse*: Ausgangsparameter und eigentliches Ergebnis
- In-Out-Parameter erscheinen im Aufruf und der Antwort

Ausgezeichneter Ergebnistyp

public Out-Parameter-i Procedure(…)

```
<env:Envelope …>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI"
          xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
      <rpc:result>m:Out-Parameter-i</rpc:result>
      <m:Out-Parameter-1>…</m:Out-Parameter-1>
        …
      <m:Out-Parameter-n>…</m:Out-Parameter-n>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```


Fehlermeldungen

```
<env:Envelope …>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Code und Reason obligatorisch
- Code: zur maschinellen Verarbeitung
- Reason: zusätzliche Information, nicht zur maschinellen Verarbeitung

Fehlermeldungen: Reason

```
<env:Envelope …>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- mindestens ein Text-Element
- xml:lang-Attribut obligatorsich

Fehlermeldungen: Code

```
<env:Envelope …>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Value obligatorsich
- env:Sender: Anfrage nicht korrekt, nochmalige korrigierte Anfrage erwartet

Fehlermeldungen: Subcode

```
<env:Envelope … xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>rpc:BadArguments</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Subcode optional
- Subcode genauso strukturiert, wie Code: Value obligatorisch, Subcode optional
- rpc:BadArguments: standardisierter Fehlercode

© Klaus Schild, 2004

Datenkodierung

© Klaus Schild, 2004

Datenkodierung

- SOAP überträgt Daten
- Daten müssen auf irgendeine Weise dargestellt werden:

  Wie z.B. Listen und mehrdimensionale Arrays darstellen?
- verwendete Datenkodierung kann als Zusatzinformation angegeben werden:

  env:encodingStyle="*URI*"
- "*URI*": Bezeichner für das verwendete Kodierungschema
- verschiedene Kodierungsschemata (encodingStyle-Attribute) innerhalb einer SOAP-Nachricht erlaubt

© Klaus Schild, 2004

Kodierungsschemata

- Kodierungsschema kann anwendungsspezifisch sein:

  "http://www.google.com/gs-encoding" (fiktiv)
- in SOAP vordefiniertes Kodierungsschema:
  - **"http://www.w3.org/2003/05/soap-encoding"**
  - legt Darstellung von folgenden Dingen fest:

    Referenzen: mit id und ref

    Arrays: mit enc:itemType und enc:arraySize

© Klaus Schild, 2004

Arrays

- kommen in RPCs häufig vor
- können z.B. mit xsd:sequence definiert werden:

```
<xsd:complexType name="ArrayOfTwoIntegers">
  <xsd:sequence>
    <xsd:element name="number" type="xsd:int" minOccurs="2" maxOccurs="2" />
  </xsd:sequence>
</xsd:complexType>
```

- Nachteile:
  - *keine* standardisierte Darstellung
  - *keine* kompakte Schreibweise für mehrdimensionale Arrays

© Klaus Schild, 2004

SOAP-Arrays

```
<numbers xmlns:enc="http://www.w3.org/2003/05/soap-encoding"
        enc:itemType="xsd:int" enc:arraySize="2">
  <number>1</number>
  <number>2</number>
</numbers>
```

- Array mit zwei Elementen vom Typ xsd:int.
- enc:arraySize="*": Array mit beliebig vielen Elementen

© Klaus Schild, 2004

Mehrdimensionale SOAP-Arrays

```
<numbers enc:itemType="xsd:int" enc:arraySize="2 2">
  <number>1</number>      → a1 b1
  <number>2</number>      → a2 b1
  <number>3</number>      → a1 b2
  <number>4</number>      → a2 b2
</numbers>
```

b

a

- 2x2-Matrix mit Elementen vom Typ xsd:int.
- enc:arraySize="* 2": nx2-Matrix
- enc:arraySize="* *": nxm-Matrix *nicht* erlaubt!

Verarbeitungsmodell

Verarbeitung einer SOAP-Nachricht

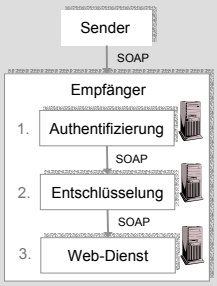
Empfänger muss verarbeiten:

- Body
- Header Blocks mit mustUnderstand="true"

Empfänger kann ignorieren:

- Header Blocks mit mustUnderstand="false"
- Header Blocks *ohne* mustUnderstand-Attribut

Schrittweise Verarbeitung



SOAP-Nachrichten können schrittweise verarbeitet werden, z.B.:

1. Authentifizierung: Verifizierung einer digitalen Signatur in einem Header Block
2. Entschlüsselung des Body
3. Aufruf des eigentlichen Web-Dienstes

Vorteile

Aufgabenteilung:

- spezialisierte Server
- z.B. Authentifizierungs-Server und Server, auf dem der eigentliche Dienst läuft
- Authentifizierungs-Server kann *vor* der Firewall liegen, die anderen Server *hinter* der Firewall

Lasterverteilung:

- Verteilung auf verschiedene Server

SOAP-Knoten

SOAP-Knoten: Rechner, die jeweils Teil einer SOAP-Nachricht verarbeiten

Zwischenknoten (*intermediary*)

Endknoten (*ultimate receiver*)

Beachte: nächster SOAP-Knoten jeweils *nicht* festgelegt

Bezeichnung von SOAP-Knoten

SOAP-Knoten: werden durch URIs identifiziert

Zwischenknoten (*intermediary*):

- anwendungspezifische URI
- http://www.w3.org/2003/05/envelope/role/**next**

Endknoten (*ultimate receiver*):

http://www.w3.org/2003/05/envelope/role/**ultimateReceiver**

© Klaus Schild, 2004

Festlegung der Zuständigkeiten

```
<env:Header>
  <p:oneBlock xmlns:p="http://example.com" env:role="URI">
    ...
  </p:oneBlock>
  <q:anotherBlock xmlns:q="http://example.com"
      env:role="http://www.w3.org/2003/05/envelope/role/next">
    ...
  </q:anotherBlock>
  <r:aThirdBlock xmlns:r="http://example.com">
      ...
  </r:aThirdBlock>
</env:Header>
```

- role: zuständiger SOAP-Knoten (URI)
- fehlt role-Attribut, dann ist der Endknoten zuständig

© Klaus Schild, 2004

Aufgabe eines Zwischenknoten

1. verarbeitet Header Blocks mit
   - role=" http://www.w3.org/2003/05/envelope/role/**next**"
   - role="*URI*", wobei "*URI*" den betreffenden Zwischenknoten bezeichnet
2. löscht alle verarbeiteten Header Blocks
3. fügt eventuell neue Header Blocks hinzu
4. leitet die so modifizierte SOAP-Nachricht an den nächsten SOAP-Knoten weiter

Beachte: Header Blocks ohne mustUnderstand="true" müssen *nicht* verstanden werden.

© Klaus Schild, 2004

Aufgabe des Endknoten

1. verarbeitet folgende Header Blocks:
   - role="http://www.w3.org/2003/05/envelope/role/**ultimateReceiver**"
   - role=" http://www.w3.org/2003/05/envelope/role/**next**"
   - ohne role-Attribut
2. verarbeitet Body

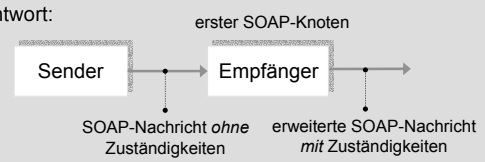
Beachte: Body muss *immer* verstanden werden.

© Klaus Schild, 2004

Festlegung der Zuständigkeiten in SOAP

- Vorteile der schrittweisen Verarbeitung klar:
  Aufgabenteilung und Lastverteilung
- Warum aber Zuständigkeiten in einer *SOAP-Nachricht* festlegen?
- Zuständigkeiten sollten doch für Sender *transparent* (nicht sichtbar) sein!

Antwort:



© Klaus Schild, 2004

Protokoll-Bindungen

© Klaus Schild, 2004

Protokoll-Bindungen (Bindings)

- Spezifikation, wie SOAP-Nachrichten mit einem bestimmten Protokoll übertragen werden
- Beschreibt, wie abstrakte SOAP-Nachrichten in konkrete Nachrichten umgewandelt (serialisiert) werden.
- nicht vorgeschrieben, *wie* eine Protokoll-Bindung spezifiziert wird
- Beachte: innerhalb einer Kette von SOAP-Knoten unterschiedliche Protokoll-Bindungen möglich

© Klaus Schild, 2004

Protokoll-Bindungen

- konkrete Nachricht meist XML, kann aber auch ein beliebig anderes Format sein:

  z.B. komprimiertes Binärformat
- wichtig: Umwandlung in konkrete Nachricht ohne Informationsverlust, Umwandlung also symmetrisch

© Klaus Schild, 2004

Protokoll-Bindungen

- HTTP-Binding bisher als *einzige* Protokoll-Bindung für SOAP standardisiert.
- zwei unterschiedliche Bindungen:
  - für HTTP-POST
  - für HTTP-GET

© Klaus Schild, 2004

HTTP

- Anfrage-Antwort-Muster
- zustandsloses Protokoll
  - jedes Anfrage-Antwort-Paar eine isolierte, abgeschlossene Einheit
  - Daten von vorherigen Anfragen oder Antworten stehen später nicht mehr zur Verfügung

© Klaus Schild, 2004

Die wichtigsten HTTP-Formate

HTTP GET
- fragt Web-Ressource mit einer URI ab
- Parameter können in eine URI kodiert werden, z.B.:

  http://myserver.com/search?q=hello%20there

HTTP POST
- fragt Web-Ressource ab, übermittelt gleichzeitig Daten

HTTP HEAD
- fragt Datum und Zeit einer Web-Ressource ab

© Klaus Schild, 2004

HTTP-POST-Anfrage an Google

```
POST /search/beta2/doGoogleSearch HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope …>
 <env:Body>
   <doGoogleSearch xmlns="urn:GoogleSearch">
     <key xsi:type="xsd:string">3289754870548097</key>
     <q xsi:type="xsd:string">Eine Anfrage</q>
       …
   </doGoogleSearch>
 </env:Body>
</env:Envelope>
```

© Klaus Schild, 2004

Antwort von Google

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope …>
    <env:Body>
      <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch"
            env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <return xsi:type="ns1:GoogleSearchResult">…</return>
    </ns1:doGoogleSearchResponse>
    </env:Body>
</env:Envelope>
```


HTTP-GET-Anfrage

```
GET /Reservations/itinerary?reservationCode=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Accept: application/soap+xml
```

- ruft getItinerary(reservationCode) auf
- Vorteil: entspricht Grundsatz, dass jede Web-Ressource über eine URI identifiziert wird
- gebuchte Reise ist eine Web-Ressource, URI sollte daher Reservierungsnummer (reservationCode) enthalten
- HTTP-GET empfohlen, wenn *alle* Parameter Web-Ressourcen identifizieren
- Nachteil: offen, wie aus SOAP-Nachricht eine URI generiert wird

SOAP-Engines

SOAP-Engines

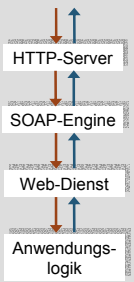
Problem:

- Wie sende ich SOAP-Nachrichten von A nach B?
- Wer empfängt die Nachrichten und gibt sie an den Web-Dienst weiter?
- Wie bekomme ich die Antwort zurück?

Lösung:

- SOAP-Engine, d.h. eine spezielle Software zur Verarbeitung von SOAP-Nachrichten

SOAP-Engines



- typischerweise mit einem HTTP-, SMTP oder FTP-Server betrieben
- bekannte SOAP-Engines:

Apache Web Services Project axis

➔ http://ws.apache.org/axis/

Microsoft .net

SOAP::Lite for Perl

➔ http://www.soaplite.com/

Vor- und Nachteile

Vorteile

+ etablierter Standard, wird u.a. in .net verwendet
+ sowohl RPC als auch Messaging werden unterstützt
+ RPC über Firewalls hinweg möglich
+ einfach erweiterbar
+ Erweiterungen unabhängig voneinander

© Klaus Schild, 2004

Nachteile

- RPCs über Firewalls hinweg nicht unbedingt erwünscht
- zusätzlicher Verarbeitungsaufwand
- für viele notwendige Erweiterungen noch kein etablierter Standard

  Beispiel: wsu:identifier vs. wsa:MessageID
- Protokoll-Bindungen können unterschiedliche Semantik haben

  Beispiel: SMTP-Binding asynchron, HTTP-Binding synchron
- daher heutzutage nicht vollständig kompatibel

© Klaus Schild, 2004

Literatur

SOAP 1.2:

- http://www.w3.org/TR/soap12-part0/

SOAP 1.1:

- http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

Übersicht über Erweiterungen:

- http://www-306.ibm.com/software/solutions/webservices/pdf/SecureReliableTransactedWSAction.pdf

© Klaus Schild, 2004