Hauptseminar Webservices

„SOAP und das Messaging Framework“

von Michael Buchzik,

7.5.2003

Betreuer: Thorsten Strufe

Inhalt

1. Einführung

1.1 Was ist SOAP und wofür ist es gedacht?

Die Grundidee hinter SOAP ist, dass Anwendungen ermöglicht wird, unabhängig von Betriebssystem, Programmiersprachen und allen anderen Implementierungsdetails untereinander Informationen austauschen zu können.
Zusätzlich soll es leichtgewichtig und strukturiert sein.
Daher hat man sich für einen XML-basierten Ansatz entschieden.

1.2 Verwandte Anwendungen und Vorgängerversionen

Den Wunsch, Daten und Anwendungen über ein Netzwerk zu nutzen, gibt es natürlich schon länger als die Idee von Webservices; das Buch „Webservice-Programmierung mit SOAP“ [SOAP1] nennt zwei „Vorläufer“ von SOAP:

RPC (Remote Procedure Call, verteiltes Rechnen): Methode, mit der ein Programm eine Prozedur (oder Funktion oder Methode) eines anderen Programms aufruft, dabei Parameter übergibt und Rückgabewerte erhält.
(bei XML-RPC bzw. „RPC-bastiertem SOAP“ ist XML die Darstellung von Parametern und Rückgabewerten).
EDI (Electronic Document Exchange): Standardformat, das die automatisierte Interpretation von finanziellen und kommerziellen Dokumenten und Nachrichten erlaubt. Es ist die Grundlage für automatisierte geschäftliche Transaktionen.
(bei „Dokument-basiertem“ SOAP ist XML ein anwendungsspezifisches Dokument bzw. Nachricht, wie z.B. Warenbestellung, o.ä.)

XML-RPC (TM)

XML-RPC [XMLRPC] stellt einen einfachen, XML-basierten Remote Procedure Call über HTTP 1.1 dar. Es wurde von der Firma *UserLand Software* 1999 entwickelt und könnte eine ursprüngliche Version von SOAP gewesen sein [SOAP1, S.174].

Vorteile: einfach (Spezifikation umfasst 5 Din A4 Seiten); es existieren viele Implementationen / Bibliotheken für die meisten Programmiersprachen; grundlegende Datentypen sind vorhanden (Integer, Boolean, String, Double, DateTime, base64-encodierte Binärdaten, Kombinationen wie Records/Structs und Arrays/Listen);
Firewalldurchlässigkeit auf Grund des Hypertext Transfer Protokolls.

Nachteile: Nicht universell genug, „nur“ über HTTP (HTTP-POST) nutzbar.

SOAP 1.1 (Acronym für „Simple Object Access Protocol“)

SOAP in der Version 1.1 ist eine **W3C-Notiz** aus dem Jahr 2000, sie entstand bereits durch mehrere beteiligte Firmen (UserLand Software, IBM, Microsoft, Lotus, DevelopMentor).
Es stellt ein XML-basiertes schlankes Protokoll zum Austausch von Informationen in einer dezentralisierten Umgebung dar und definiert grundlegende XML-Schema-basierte Kodierregeln zur Nachrichtenverarbeitung und für anwendungsspezifische Datentypen.

Im Vergleich zu XML-RPC ist SOAP 1.1 bereits deutlich variabler einsetzbar:
Nicht nur RPC, sondern beliebige anwendungsspezifische Nachrichten sind nun möglich.
SOAP 1.1 kann theoretisch in Verbindung mit beliebigen darunter liegenden Netzwerkprotokollen verwendet werden, in der Notiz selbst wird aber nur HTTP und das HTTP Extension Framework (RFC 2774) behandelt.

SOAP 1.2

SOAP 1.2 ist eine **W3C Recommendation** [SOAP3, SOAP4, SOAP8], und wurde von der W3C **XMLProtocol Working Group** umfangreich überarbeitet (Dokumentstruktur, zusätzliche bzw. geänderte Syntax, SOAP HTTP Bindung, RPC, SOAP Kodierungen). Und der Begriff „SOAP“ ist nun kein Acronym für „Simple Object Access Protocol“ mehr, sondern ein Name der für sich selbst steht. SOAP in der Version 1.2 ist die inzwischen fünfte Überarbeitung des Standards [SOAP1, S.174].

In produktiven Umgebungen kommen sowohl XML-RPC als auch SOAP in den Versionen 1.1 und 1.2 zum Einsatz.
In Zukunft soll SOAP 1.2 als Basis des **W3C XML Protocol Version 1.0** dienen [XMLProtocol].

2. Das Messaging Framework

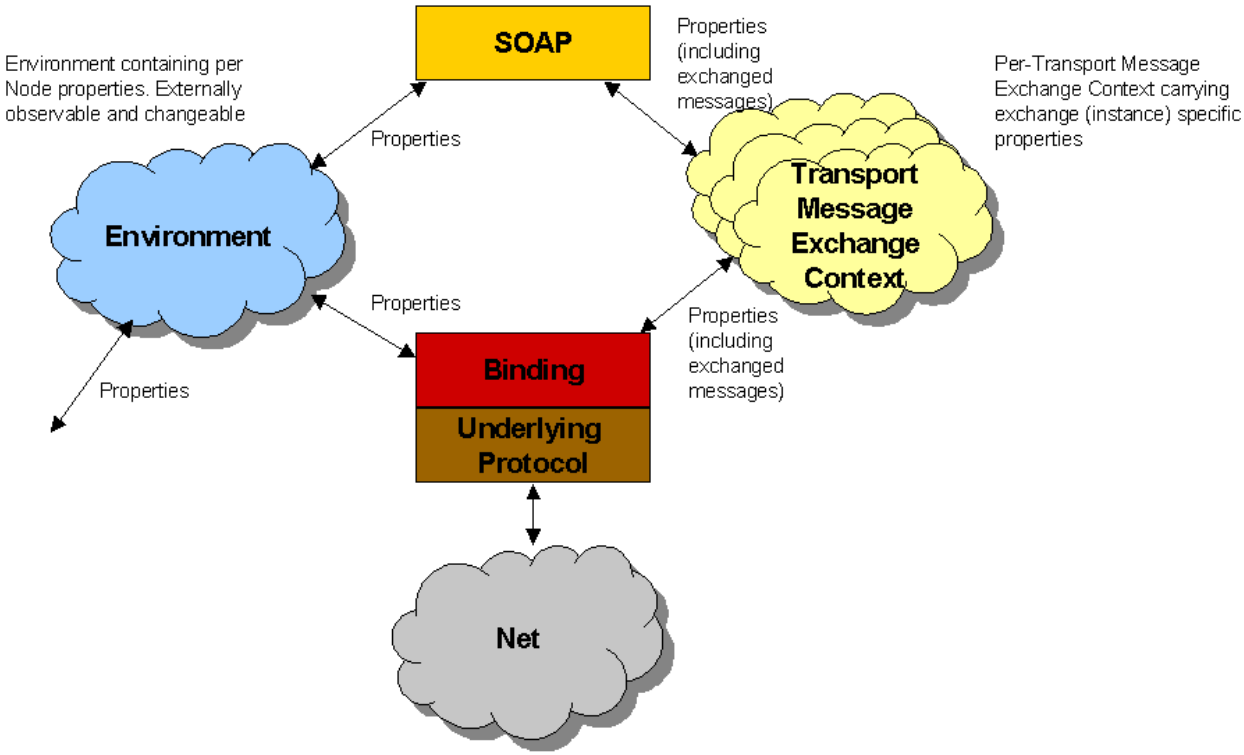

Abb. 1: Das SOAP Messaging Framework

Das SOAP Messaging Framework besteht aus vier Teilen:

- Der Umgebung (Environment), auf der ein SOAP-Knoten läuft,
- dem SOAP-Prozessor, der die XML-/SOAP-Nachrichten bearbeitet und eine Schnittstelle (Proxy) zur darunter liegenden Softwareschicht darstellt,
- dem Message Exchange Context, dem Schema nach dem der Nachrichtenaustausch stattfindet
- und der Anbindung an ein Netzwerkprotokoll, dem „SOAP Protocol Binding“

Hier werden die letzten drei Punkte betrachtet, da diese unabhängig von der Umgebung funktionieren sollen und die Anpassung an die Hard- und Softwareumgebung von der verwendeten SOAP-Bibliothek bewerkstelligt werden muss!

2.1 Der grundlegende Aufbau einer SOAP-Nachricht (SOAP Message Construct)

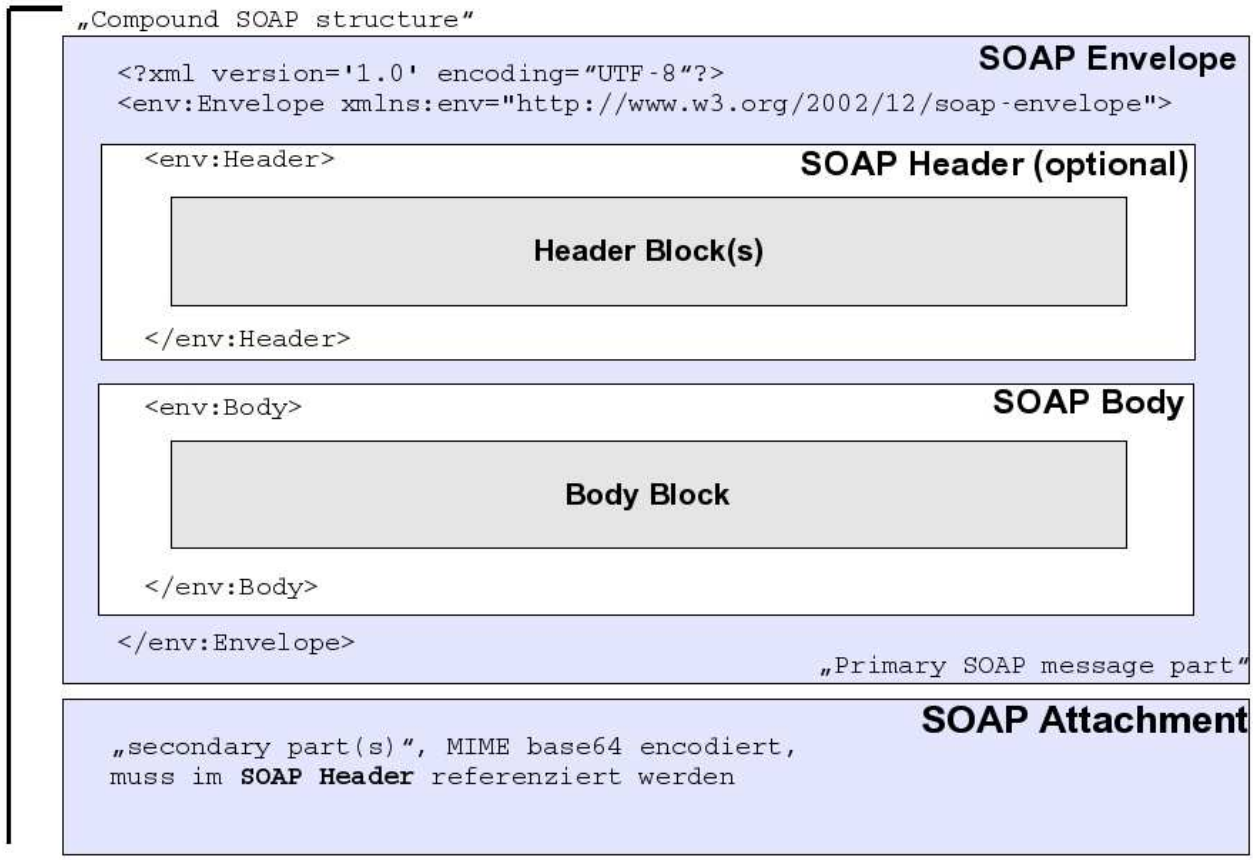

Abb. 2: das SOAP Message Construct

Eine SOAP-Nachricht hat ein ganz bestimmtes XML-Schema
(SOAP 1.2: http://www.w3.org/2002/12/soap-envelope), das strikt eingehalten werden muss.
Die Wurzel des XML-Baumes stellt das Tag „Envelope" dar. Es umschließt die Nachricht und muss vorhanden sein.
Optional kommen als nächstes ein oder mehrere Header-Blöcke, umschlossen von einem Header Tag. Mit den optionalen Kopfdaten werden sogenannte SOAP-Features realisiert, extra Funktionalität die im SOAP-Standard gar nicht definiert wird.
Die eigentliche Nachricht wird allerdings im Body-Block übertragen, der daher auch immer vorhanden sein muss.
Das XML in den Datenblöcken (Header, Body) kann entweder dem SOAP-Encoding-Schema (http://www.w3.org/2002/12/soap-encoding) oder einem selbst definiertem XML-Schema entsprechen. Weitere Schemata können mit dem Attribut `encodingStyle` referenziert werden.
Die einzige Einschränkung gilt bei SOAP RPC: Das erste Kindelement im Body-Block muss der anzusprechende Methodenname sein!

2.2 Das SOAP 1.2 Attachment Feature

Obwohl vom Encoding-Schema zulässig, waren Daten nach dem SOAP-Envelope bei SOAP 1.1 in der Spezifikation nicht vorgesehen. Bei SOAP 1.2 existiert nun ein eigener „Working Draft" [SOAP5] der definiert, wie man Anhänge neben SOAP-Envelopes erstellt und wie man diese Anhänge aus dem Envelope heraus referenziert.
Werden SOAP-Nachrichten mit Anhängen verschickt, nennt man den Teil des SOAP-Envelope „Primary SOAP message part" oder einfach „primary part", und den Anhang „secondary part", bzw. mehrere Anhänge „secondary parts". Der Primary Part und alle Secondary Parts zusammen heißen „Compound SOAP structure".

2.3 Beispiel: Selbst erstellter SOAP RPC, der die Methode „Ping“ realisiert:

Zuerst der Request:

```
<?xml version='1.0' encoding='UTF-8'?>
<s:Envelope xmlns:s="http://www.w3.org/2002/12/soap-envelope">
  <s:Body>
    <m:Ping xmlns:m="http://meinhost.de/soap-ping"
        p:encodingStyle="http://www.w3.org/2002/12/soap-encoding">
      <host xsi:type="string">www.heise.de</host>
      <count xsi:type="integer">1</count>
    </m:Ping>
  </s:Body>
</s:Envelope>
```

Abb. 3: SOAP RPC Request

Hier im Beispiel-Request gut ersichtlich: die Header-Tags werden nicht benötigt und fehlen daher völlig. Die Kindelemente der angesprochenen Methode „Ping“ sind in einem eigenen Schema definiert, das am URL „http://meinhost.de/soap-ping“ zu finden ist. Diese Elemente dürfen zusätzlich auch noch dem SOAP-Encoding entsprechen. Die beiden Kindelemente im Beispiel erklären sich fast von selbst, das „host“-Tag soll einen String mit einem Hostnamen oder einer IP enthalten und ein optionales Tag namens „count“ kann einen ganzzahligen Wert der Anzahl der durchzuführenden Pings enthalten.
Eine mögliche Antwort (Response) könnte so aussehen:

```
<?xml version='1.0' encoding='UTF-8'?>
<s:Envelope xmlns:s="http://www.w3.org/2002/12/soap-envelope">
  <s:Body>
    <m:PingResponse xmlns:m="http://meinhost.de/soap-ping"
        p:encodingStyle="http://www.w3.org/2002/12/soap-encoding">
      <host xsi:type="string">www.heise.de</host>
      <count xsi:type="integer">1</count>
      <ip xsi:type="string">193.99.144.71</ip>
      <status xsi:type="string">alive</status>
      <avgtime xsi:type="float">7.493</avgtime>
    </m:PingResponse>
  </s:Body>
</s:Envelope>
```

Abb. 4: SOAP RPC Response

Einziger Unterschied zum Request: an den Methodennamen als erstes Tag im Body wurde das Suffix „Response“ angehängt. Alle weiteren Kindelemente stellen das Ergebnis, die Ausgabe der Prozedur dar.

2.4 Dokument-basiertes oder RPC-basiertes SOAP

Während bei Dokument-basiertem SOAP beliebige XML-Daten innerhalb des SOAP-Bodys stehen können und bei RPC-basiertem die entfernte Prozedur eindeutig angesprochen werden soll, wird in den Spezifikationen nicht verraten, wann eigentlich welche Methode vom Entwickler verwendet werden sollte!
Die Zeitschrift iX gibt eine etwas unsicher als Frage formulierte Antwort: *„Geht es im Kern um die zu verarbeitenden Daten oder handelt es sich um eine typische Prozedur, wie z.B. das Überprüfen eines Artikellagerbestandes?“* [SOAP10]. Das ist natürlich etwas unbefriedigend, da SOAP immer einen Nachrichtenaustausch darstellt und die übermittelten Daten immer auch verarbeitet werden müssen, sprich: beide Varianten sind bei jeder Problemlösung anwendbar!

2.5 Der Weg einer SOAP Nachricht

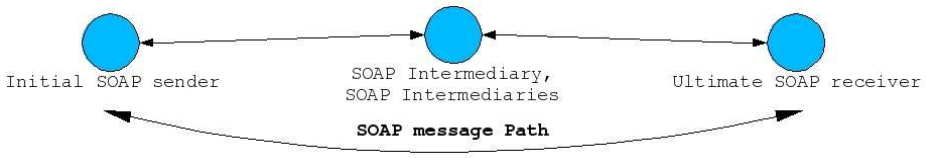

Abb. 5: SOAP message path

Nun kann es vorkommen, dass eine SOAP Nachricht nicht direkt vom Sender zum Empfänger gelangt, sondern erst über einen oder mehrere Zwischenknoten zum Zielknoten findet. Das kann aus den verschiedensten Gründen passieren, z.B. um alle Nachrichten von einem Zwischenknoten mitloggen zu lassen.
SOAP-Knoten können folgende Stellungen im Nachrichtenpfad (SOAP message path) einnehmen:

- `SOAP sender`: Jeder SOAP-Knoten, der Nachrichten verschickt (also auch weiterleitet).
- `SOAP receiver`: Jeder SOAP-Knoten, der Nachrichten empfängt.
- `Initial SOAP sender`: Der erste SOAP-Knoten im Nachrichtenpfad, der Ersteller der Nachricht.
- `SOAP Intermediary`: Zwischenknoten, der die Nachricht weitergeben muss. Steht der nächste SOAP-Knoten bereits fest, spricht man von einem „passiven Zwischenknoten“. Muss der nächste Knoten jedoch durch Analyse und evtl. sogar Modifikation der SOAP-Nachricht erst ermittelt werden, spricht man von einem „aktiven Zwischenknoten“.
- `Ultimate SOAP receiver`: SOAP-Knoten, für den die Nachricht (Body) eigentlich bestimmt ist.

2.6 SOAP Header Blöcke

SOAP Header sind optionale Datenblöcke, die der Erweiterung von SOAP dienen (SOAP Features, z.B. das „SOAP Attachment Feature“ [SOAP5]).
Werden die Syntax und die Semantik von einem oder mehreren Header-Blöcken spezifiziert, spricht man von einem „SOAP module“. SOAP module können sich auch aus einem oder mehreren SOAP Features zusammensetzen.
Nachdem ein Header-Block von einem SOAP-Knoten abgearbeitet wurde, muss er aus der SOAP-Nachricht entfernt werden.
Die Daten innerhalb eines Header-Blocks können durch ein eigenes XML-Schema referenziert sein, oder dem SOAP-Encoding entsprechen.
Verschiedene Attribute können die Abarbeitung von Header-Daten beeinflussen:

Zuständigkeit: das SOAP role-Attribut

Als erstes stellt sich die Frage: „Wer ist überhaupt für die Abarbeitung eines Header Blocks zuständig?“ Dafür ist das `role`-Attribut zuständig (Typ: xs:anyURI)! Ist es nicht vorhanden, ist einzig der eigentliche Empfängerknoten, der „ultimate Receiver“, für einen Header Block zuständig. Ansonsten steht im `role`-Attribut ein URI (Uniform Resource Identifier). Entweder der, der den SOAP Knoten eindeutig identifiziert, oder einer von drei möglichen, in [SOAP3] vorgegebenen:

- http://www.w3.org/2002/12/soap-envelope/role/none („none“):kein SOAP-Knoten darf diesen Header-Block beachten.
- http://www.w3.org/2002/12/soap-envelope/role/next („next“): der im Nachrichtenpfad nächstgelegene SOAP Knoten ist für diesen Header Block zuständig. Andersherum betrachtet: ist in einer eingehenden Nachricht in einem Header das `role`-Attribut auf diesen URI für „next“ gesetzt, ist der SOAP-Knoten für diesen Header zuständig.

- http://www.w3.org/202/12/soap-envelope/role/ultimateReceiver („ultimateReceiver“): entspricht dem nichtgesetzten `role`-Attribut, nur derjenige SOAP Knoten, der auch für die Bearbeitung des SOAP Body zuständig ist, ist für diesen Header Block zuständig.

Die folgende (Zuständigkeits-)Tabelle stammt aus [SOAP2], und soll das `role`-Attribut noch einmal veranschaulichen:

Role	fehlend	„none“	„next“	„ultimateReceiver“
Node				
initial sender	nicht anwendbar	nicht anwendbar	nicht anwendbar	nicht anwendbar
intermediary	nein	nein	ja	nein
ultimate receiver	ja	nein	ja	ja

Abb.6

Verständnisfrage: das SOAP mustUnderstand-Attribut

Generell ist eine korrekte Abarbeitung von Header Blöcken nicht zwingend vorgeschrieben. Kann ein SOAP-Knoten einen Header-Block nicht korrekt abarbeiten, nimmt er ihn aus der SOAP-Nachricht heraus und fährt mit der Abarbeitung der SOAP-Nachricht fort.
Befinden sich aber wichtige Daten in einem Header-Block, die zum Beispiel für die weitere Abarbeitung oder den weiteren Weg der Nachricht von Bedeutung sind, muss dieser SOAP-Knoten diesen korrekt verarbeiten!
Dafür ist das Attribut „`mustUnderstand`“ (vom Typ xs:boolean) gedacht. Ist dieses auf „true“ gesetzt und der SOAP-Knoten kann den Header-Block nicht interpretieren, muss er einen SOAP Fault erzeugen (SOAP mustUnderstand Fault [SOAP3, 5.4.8])!

Header-Weiterleitung: das SOAP relay-Attribut

Normalerweise wird ein Header-Block nach der Bearbeitung durch einen SOAP-Knoten aus der SOAP-Nachricht entfernt.Dies kann man durch das `relay`-Attribut (Typ: xs:boolean) verhindern. Da das SOAP Processing Model [SOAP3, 2.7.1] aber zwingend vorschreibt, bearbeitete Header Blöcke aus dieser Nachricht zu entfernen bedeutet das, dass der selbe, identische Header Block in die Nachricht wieder neu eingefügt wird.

2.7 Fehlerbehandlung: SOAP Faults

Natürlich können bei der Verarbeitung von SOAP-Nachrichten auch Fehler auftreten.
Um diese den anderen beteiligten SOAP-Knoten mitteilen zu können, ist im Envelope-Schema (http://www.w3.org/2002/12/soap-envelope) der SOAP-Fault vorgesehen [SOAP3, Abs. 5.4].
Dieser wird innerhalb des Body-Elements transportiert, und hat mindestens zwei Kindelemente:

- Ein `Code`-Element (muss vorhanden sein)
- Ein `Reason`-Element (muss vorhanden sein)
- Ein `Node`-Element (optional)
- Ein `Role`-Element (optional)
- Ein `Detail`-Element (optional)

Das SOAP (Sub-)Code Element

Das `Code`-Element besitzt zwingend ein `Value`-Element als Kindelement, und optional ein `Subcode`-Element.
Ein `Subcode`-Element besitzt wiederum ein `Value`-Element und optional ein `Subcode`-Element.

Diese dienen als maschinenlesbarer Teil der Fehlernachricht. Folgende Fault Codes sind im Standard-Schema (http://www.w3.org/2002/12/soap-envelope) bereits definiert:

- `VersionMismatch`: Die im Envelope referenzierte SOAP-Version wird vom fehlererzeugenden SOAP-Knoten nicht unterstützt.

- `MustUnderstand`: Der SOAP-Knoten der den Fehler erzeugt, kann für ihn bestimmte und als zwingend interpretierbar markierte Header-Informationen nicht fehlerfrei abarbeiten.

- `DataEncodingUnknown`: Ein Kindelement im SOAP Header-Block oder Body ist in einer Kodierung angegeben, die der fehlererzeugende SOAP-Knoten nicht unterstützt.

- `Sender`: Die Nachricht wurde inkorrekt geformt oder enthielt nicht die benötigten Informationen, um sie weiterverarbeiten zu können. In der Regel ein Zeichen dafür, dass vor erneutem Senden Änderungen an der SOAP-Nachricht vorgenommen werden sollten.

- `Receiver`: Die Nachricht konnte nicht verarbeitet werden, aber im Gegensatz zum vorherigen Punkt eher wegen der Abarbeitung als wegen Inhalten der Nachricht selbst. Die selbe Nachricht könnte eventuell zu einem späteren Zeitpunkt erfolgreich verarbeitet werden (z.B. weil ein weiterer benötigter SOAP-Knoten im Moment nicht erreichbar ist).

Zusätzlich können noch eigene, anwendungsspezifische Fehlercodes in weiteren XML-Schemas definiert werden.

Das SOAP Reason Element

Das zwingend vorhandene `Reason`-Element ist dazu gedacht, eine „human readable explanation“, also eine für den Anwender verständliche Fehlermeldung zu beinhalten.
Es besitzt mindestens ein `Text`-Kindelement, das als Attribute Sprache und (optional) Zeichensatz, als Inhalt die eigentliche Fehlernachricht in der zuvor angegebenen Sprache enthält.

Das SOAP Node Element

Das optionale `Node`-Element ist dazu gedacht die Information zu stellen, welcher Knoten auf dem Nachrichtenpfad die (ursprüngliche) Fehlernachricht erzeugt hat.

Das SOAP Role Element

Das `Role`-Element enthält den URI der Rolle, die er zum Zeitpunkt des Auftretens des Fehlers im Nachrichtenpfad gespielt hat.

Das SOAP Detail Element

Das optionale `Detail`-Element ist dazu gedacht, anwendungsspezifische Fehlermeldungen bezüglich des SOAP-Bodys zu transportieren.
Es ist voll variabel und hat optional Attribute und Kindelemente, sog. „SOAP detail entrys“.
Diese Daten müssen allerdings nicht vorhanden sein.

Beispiel SOAP Fault:

```
<?xml version='1.0'?>
<s:Envelope xmlns:s="http://www.w3.org/2002/12/soap-envelope"
            xmlns:rpc='http://www.w3.org/2002/12/soap-rpc'>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
        <s:Subcode>
          <s:Value>rpc:BadArguments</s:Value>
        </s:Subcode>
      </s:Code>
      <s:Reason>
       <s:Text xml:lang="en-US">Unknown Host!</s:Text>
       <s:text xml:lang="de">Unbekannter Rechnername!</s:Text>
      </s:Reason>
      <s:Detail>
        <e:myFaultDetails
          xmlns:e="http://meinhost.de/ping-faults">
        <e:message>Couldn't resolve IP to hostname</e:message>
        <e:errorcode>23</e:errorcode>
        </e:myFaultDetails>
      </s:Detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

Abb. 7: SOAP RPC Fault

Um zum ersten Beispiel (Ping als SOAP RPC) zurückzukommen, hier ein möglicher Fehler: den zu pingenden Host gibt es nicht.
Als zusätzliches Encoding-Schema wird bereits im Envelope http://www.w3.org/2002/12/soap-rpc referenziert. Dieses stellt beim Fehlercode die Option „`rpc:BadArguments`“. Das `Reason`-Element enthält Fehlermeldungen in englisch und deutsch, und das `Detail`-Element wurde von der imaginären Anwendung in einem eigenem Schema selbst definiert.

2.8 Das SOAP Protocol Binding Framework

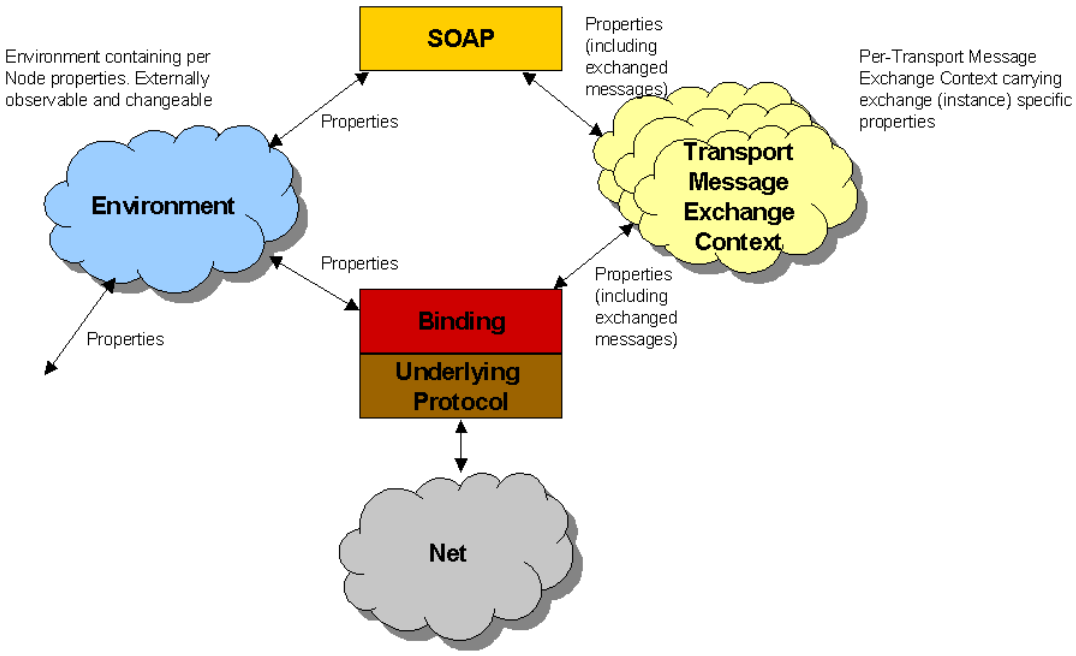

Abb. 8: Das SOAP Messaging Framework

Die Anbindung eines SOAP Prozessors an ein (beliebiges) Netzwerkprotokoll nennt man SOAP Protocol Binding, sie ist Teil des SOAP Messaging Frameworks.
Als Standard wird zwar das HTTP Protokoll vorgeschlagen, aber die Auswahl des Netzwerkprotokolls hängt von den Anforderungen an dem Transportweg (Sicherheit, Zuverlässigkeit, Geschwindigkeit, etc.) und von dem Schema des Nachrichtenaustausches, den „Message Exchange Patterns“ ab (z.B. Request-Response-MEP, Request-multi-Response-MEP, Response-MEP, Fire and Forget, etc.).

2.9 Das SOAP HTTP Binding

Das SOAP HTTP Binding benutzt das „SOAP Web Method Feature“, die Identifikation des SOAP Knotens erfolgt über den Request-URI. HTTP POST stellt das „SOAP Request-Response message exchange pattern“ dar.
Das SOAP HTTP Binding ist bei den sogenannten „Web Methoden“ (GET, POST, PUT, DELETE) jedoch nur auf *POST* und *GET* beschränkt. Wegen Content-Negotiation sollte übrigens HTTP/1.1 verwendet werden. Der Mime-Typ lautet nun bei SOAP in der Version 1.2 „application/soap+xml“ (bis SOAP 1.1: text/xml).

HTTP GET entspricht dem „SOAP Response message exchange pattern“ und sollte bei ausschliesslicher Abholung von Informationen benutzt werden („Information Retrieval“).
Die GET-Methode sollte sicher und idempotent sein (sicher: der Zustand der Maschine bleibt unverändert; Idempotenz: n identische Aufrufe erzeugen n gleiche Ergebnisse).

Beispiel HTTP-GET-Request:

```
GET /postleitzahl?code=98693 HTTP/1.1
Host: soap.postleitzahl.de
Accept: text/html, application/soap+xml
```

Abb. 9

Das Beispiel einen imaginären SOAP Knoten, der in einer Datenbank nach Ortsnamen zu gegebenen Postleitzahlen sucht. Parameter werden im URI übergeben (URI: http://soap.postleitzahl.de/postleitzahl?code=98693). Zusätzlich ist hier in der Zeile „Accept“ noch angegeben, in welchen Formaten der SOAP-Knoten das Ergebnis der Anfrage zurückliefern darf.

Beispiel SOAP HTTP-POST-Request

```
POST /postleitzahl?code=98693 HTTP/1.1
Host: soap.postleitzahl.de
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: n

<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Header>
  ...
 </env:Header>
 <env:Body>
  <plz:PLZ xmlns:plz="http://soap.postleitzahl.de/plz">
    <plz:Code>98693</plz:Code>
  </plz:PLZ>
 </env:Body>
</env:Envelope>
```

Abb. 10

Im Gegensatz zur HTTP-GET-Methode wird beim HTTP-POST eine vollständige SOAP-Nachricht an den SOAP-Knoten geschickt, daher wird dies auch im Sinne von SOAP als vollständiger Request gesehen und HTTP-GET nicht!

Dass hier noch einmal im Teil des HTTP-Requests ein Parameter auftaucht, wird in der SOAP-Recommendation als *„Web Architecture compatible SOAP usage“* bezeichnet, wobei der Parameter keine praktische Verwendung (im Sinne der Parameterübergabe) mehr findet!
(In anderen Beispielen in der W3C SOAP Recommendation kann man z.B. Transaktions-IDs an gleicher Stelle finden, die dann noch einmal in Header-Blöcken vorkommen.)
Solch ein Request wird allerdings in den Server-Logs mitprotokolliert, was meines Erachtens als eventueller Beweis bei nachträglich aufkommenden juristischen Fragen recht praktisch sein kann (z.B. Inanspruchnahme des Dienstes).

Beispiel SOAP HTTP-Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: n

<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Header>
  ...
 </env:Header>
 <env:Body>
  <plz:PLZ xmlns:plz="http://soap.postleitzahl.de/plz">
    <plz:Ort>Ilmenau</plz:Ort>
    <plz:Region>Thüringen</plz:Region>
    <plz:Code>98693</plz:Code>
  </plz:PLZ>
 </env:Body>
</env:Envelope>
```

Abb. 11

Die HTTP-Response sieht bei beiden Request-Methoden gleich aus. Nach den HTTP-Kopfdaten kommt dann die (übrigens Dokument-basierte) SOAP-Response.

Mitgenutzt werden auch die HTTP Response Codes, z.B. 2xx (erfolgreich/successful), 3xx (Weiterleitung/redirection), 4xx (Client Fehler/Error) und 5xx (Server Fehler/Error). So wird bei einem SOAP Fault zum Beispiel immer `HTTP/1.1 500 Internal Server Error` zurückgeliefert.

2.10 Andere SOAP Bindings

Das HTTP-Protokoll kann aber auch ungünstig sein und nicht so recht zum gewünschten Message Exchange Pattern passen: Was soll man bei „Fire and Forget“ z.B. mit der HTTP-Response anfangen, wo man doch nur Daten versenden und nicht empfangen möchte?

Dafür könnte sich aber z.B. das SOAP Email Binding [SOAP6], oder – noch experimenteller - ein Binding an das Jabber-Protokoll [SOAP11] eignen.

Beispiel SOAP Email Binding

```
From: soap@soapnode.de
TO: recipient1@meinhost.de
CC: recipient2@meinhost.de
Subject: Sitenews www.soapnode.de
Date: Wed, 19 Jun 2003 15:50:00 CET
Message-Id: <3EB64CA920AC106@soapnode.de>
Content-Type: application/soap+xml

<?xml version='1.0'>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Body>
  <s:SiteInfo xmlns:s="http://soapnode.de/info">
    <s:newTutorial>
     <s:Title>PHP für Anfänger</s:Title>
     <s:Addr>http://www.soapnode.de/tutorial/123/</s:Addr>
    </s:newTutorial>
    <s:newArticle>
      <s:Title>Lassen sich PHP und ASP überhaupt vergleichen?</s:Title>
      <s:Addr>http://www.soapnode.de/article/2048/</s:Addr>
    </s:newArticle>
    <s:News>
      <s:Titlle>PHP 5.0.0 Beta1</s:Title>
      <s:Date>20030714</s:Date>
      <s:Addr>http://www.soapnode.de/news/100465/</s:Addr>
    </s:News>
  </s:Info>
 </env:Body>
</env:Envelope>
```

Abb. 12

Man nehme einmal an, man würde keine sofortige Rückmeldung benötigen, ob eine SOAP Nachricht angekommen ist oder fehlerfrei verarbeitet wurde. Außerdem könnte man die selbe SOAP-Nachricht an viele Empfänger senden.
Im obigen Beispiel könnte ein SOAP-Knoten andere informieren, dass auf einer Webseite neue Artikel und Neuigkeiten erschienen sind. Partnerwebseiten könnten auf diese Art und Weise automatisch eine Verlinkung auf die Seite des SOAP Knotens mit neuen Daten erzeugen.

3. Fazit

Messaging mittels SOAP hat eindeutig Zukunft!
Man sollte aber nicht die Hoffnung hegen, SOAP und Web Services (in deren Stapel sich SOAP befindet) würden die universelle Allzweckmethode werden!
Die bestehenden Konzepte für verteilte Anwendungen werden durch die Web Services nur ergänzt, aber nicht ersetzt.

Die größte Stärke von SOAP ist meines Erachtens auch gleichzeitig die größte Gefahr:
die Flexibilität und Erweiterbarkeit durch SOAP Features und SOAP Module!
Sollten die Softwarehersteller (was sich im Moment bereits abzeichnet) in diesem Bereich versuchen, Alleinstellungsmerkmale für ihre Produkte zu erzeugen, geht die geforderte grundsätzliche Interoperabilität zwischen verschiedenen Platformen verloren!

Anhang A: Literatur, Resourcen:

[SOAP1] **„Webservice-Programmierung mit SOAP“**,
O'Reilly Verlag, 1.Auflage 2002
ISBN 3-89721-159-9

[SOAP2] **„SOAP Version 1.2 Part 0: Primer“**,
W3C Candidate Recommendation, 19. Dezember 2002
(zum Zeitpunkt des Hauptseminars!)
http://www.w3.org/TR/soap12-part0

[SOAP3] **„SOAP Version 1.2 Part 1: Messaging Framework“**,
W3C Candidate Recommendation, 19. Dezember 2002
(zum Zeitpunkt des Hauptseminars!)
http://www.w3.org/TR/soap12-part1

[SOAP4] **„SOAP Version 1.2 Part 2: Adjuncts“**,
W3C Candidate Recommendation, 19. Dezember 2002
(zum Zeitpunkt des Hauptseminars!)
http://www.w3.org/TR/soap12-part2

[SOAP5] **„SOAP 1.2 Attachment Feature“**,
W3C Working Draft, 14. August 2002
http://www.w3.org/TR/soap12-af

[SOAP6] **„SOAP Version 1.2 Email Binding“**,
W3C Note, 26. Juni 2002
http://www.w3.org/TR/soap12-email

„Developing Webservices, Part 3: SOAP interoperability“,
Bilal Siddiqui, IBM developerWorks, 2003
http://www-106.ibm.com/developerworks/webservices/library/ws-intwsdl3.html

[SOAP7] **„Simple Object Access Protocol (SOAP) 1.1**,
W3C Note, 8. Mai 2000
http://www.w3.org/TR/SOAP

[SOAP8] heise Newsticker, 25.6.2003: **„SOAP 1.2 als Recomendation verabschiedet“**
http://www.heise.de/newsticker/data/ola-25.06.03-002

[SOAP9] **Artikelreihe der Zeitschrift iX über Web Services**
iX 9/2002, S.121: **„Dienste leisten“**
(http://ww.heise.de/ix/artikel/2002/09/121)

[SOAP10] **Artikelreihe der Zeitschrift iX über Web Services**
iX 10/2002, S.134: **„Nehmen und geben“**
(http://www.heise.de/ix/artikel/2002/10/134/)

[XMLRPC] **„XML-RPC Specification“**,
Dave Winer von UserLand Software, 15. Juni 1999
http://www.xmlrpc.com/spec

[XMLProtocol] **XML Protocol Working Group**,
http://www.w3.org/2000/xp/Group/

„Uniform Resource Identifiers“,
RFC 2396, August 1998
http://www.ietf.org/rfc/rfc2396.txt

[SOAP11] **SOAP::Lite**
http://www.soaplite.com

Anhang B: Abbildungsverzeichnis