SOAP (Simple Object Access Protocol)

Seminar: DB-Aspekte des E-Commerce, Schwerpunkt: Techniken

Christian Gebauer <gebauer@bigfoot.com>

Universität Kaiserslautern
Sommersemester 2001

Inhaltsverzeichnis

1 Einleitung **2**
1.1 Neue Technologien: XML und HTTP . . . 2
1.2 Was ist SOAP? . . . 3
1.3 Gliederung . . . 3

2 Die SOAP 1.1 Spezifikation **3**
2.1 Die Struktur einer SOAP Nachricht: Envelope, Header und Body . . . 4
2.1.1 Das *actor* Attribut . . . 4
2.1.2 Das *mustUnderstand* Attribut . . . 5
2.1.3 Das SOAP *Fault* Element . . . 5
2.2 Datenkodierung . . . 6
2.3 HTTP als Transportprotokoll . . . 6
2.4 RPC . . . 7

3 Typische Einsatzgebiete **8**
3.1 Client/Server-Anwendungen . . . 8
3.2 Transportprotokoll für komplexere APIs . . . 8
3.3 Integration inkompatibler RPC/RMI Mechanismen . . . 8

4 Beispiel **8**
4.1 Automatisierte Abwicklung einer Auktion . . . 9
4.2 Apache SOAP . . . 9
4.3 Eine Beispielsitzung . . . 12
4.4 Fazit . . . 14

5 Betrachtung der Vor- und Nachteile **14**
5.1 Allgemeine Vorteile XML-basierter Protokolle . . . 14
5.2 Reifegrad . . . 14
5.3 Leistungsverhalten . . . 15
5.4 Umfang des Standards . . . 15
5.4.1 Transaktionsmanagement . . . 15
5.4.2 Authentisierung, Verschlüsselung und sonstige Sicherheitsmerkmale 16
5.4.3 Fazit . . . 16

1 Einleitung

Seitdem die physikalischen Möglichkeiten zur Vernetzung von Computersystemen bestehen, werden Verfahren benötigt, die die darauf laufenden verteilten Softwaresysteme verknüpfen. Die einfachste Lösung dieses Problems besteht darin, eine "*rohe*" Netzwerkverbindung zum Zielsystem zu öffnen und darauf ein systemspezifisches Protokoll zur Informationsübertragung zu benutzen. Dieser Ansatz bietet optimale Performanz, stößt allerdings spätestens dann an seine Grenzen, wenn Drittsysteme integriert werden sollen, die ein eigenes Protokoll verwenden. Deshalb wurden schon früh standardisierte Protokolle eingeführt. Diese kann man aufteilen in:

1. Protokolle die eine bestimmte, eng definierte Aufgabe erfüllen
   Diese sind typischerweise textbasiert und vollständig dokumentiert. Ein typisches Beispiel ist der Transport von EMails mit SMTP und POP3.

2. universell einsetzbare Protokolle
   Diese sind meistens binär und häufig in komplette APIs zur Entwicklung von verteilter Software eingebettet.

Typische Vertreter der zweiten Kategorie sind:

- Sun RPC [6], [7]
- IIOP, das Basisprotokoll von CORBA [8]
- ORPC [9], das Basisprotokoll von DCOM [10]
- JRMP, ein mögliches Basisprotokoll von RMI [11]

Die allermeisten dieser Protokolle sind mehr oder weniger proprietär. Entweder die Spezifikation ist nicht offengelegt – „JRMP is a proprietary, stream-based protocol that is only partially specified“ [11] – oder das verwendete binäre Datenformat ist mit bestimmten Rechnerarchitekturen bzw. Programmiersprachen nur umständlich erzeugbar.

1.1 Neue Technologien: XML und HTTP

Seit der Entwicklung von XML existiert nun ein Datenformat, auf dessen Basis neue Protokolle entwickelt werden können, die interessante Vorteile gegenüber den bestehenden Angeboten haben. Entscheidend ist dabei vor allem die Fähigkeit von XML, beliebige Daten mehr oder weniger selbstbeschreibend darzustellen. D.h., zwei Kommunikationspartner können Daten austauschen obwohl sie nur minimale Kenntnisse über ihr Gegenüber haben. Im Gegensatz dazu müssen bei den oben aufgezählten Protokollen in der Regel zusätzliche Informationen vor dem Beginn der Kommunikation ausgetauscht werden, z.B. durch Installation von Stubs. Dies macht beide Seiten voneinander abhängig.

Als Transportmedium für ein solches neues Protokoll bietet sich HTTP an, da über den POST Befehl in einfacher Weise ein RPC-artiger Dialog realisiert werden kann. Außerdem steht eine umfangreiche Infrastruktur im Bezug auf Proxies und Firewalls zur Verfügung, mehr dazu in Abschnitt 3.

Ein Funktionsaufruf über ein fiktives solches Protokoll könnte z.B. wie folgt aussehen:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>orderPart</methodName>
  <params>
    <partNumber><integer>2345</integer></partNumber>
    <quantity><integer>5</integer></quantity>
    <customerID><string>C4523</string></customerID>
  </params>
</methodCall>
```

Diese Anfrage wird nun über den POST Befehl an den Server geschickt und dieser antwortet über die gleiche Verbindung. Falls die Bestellung angenommen werden kann, wird dieses Ergebnis übermittelt:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
     <price><money>50.45</money></price>
  </params>
</methodResponse>
```

Falls aber die Bestellung nicht erfolgreich ist, wird dies dem Kunden über einen definierten Fehlermechnismus mitgeteilt:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
     <code><integer>21</integer></code>
     <reason><string>Part not in stock.</string><reason>
  </fault>
</methodResponse>
```

Dieses Beispiel ist dem XML-RPC Protokoll [12] nachempfunden. XML-RPC ist ein sehr einfaches XML basiertes RPC Protokoll und in gewisser Weise ein Vorläufer von SOAP, da einige der SOAP Entwickler vorher an Entwicklung von XML-RPC beteiligt waren.

1.2 Was ist SOAP?

SOAP (Simple Object Access Protocol) ist ein Übertragungsprotokoll zum Übermittlung von Nachrichten in verteilten Systemen und zur Durchführung von RPCs (Remote Procedure Call). Der Namensbestandteil "*Object*" ist nicht im Sinne von OOP zu verstehen. SOAP definiert kein objektorientiertes Programmiermodell, sondern lediglich eine Konvention zur Durchführung eines Funktionsaufrufs und selbst diese ist optional.

Alle Informationen werden als XML-Dokumente [1] übermittelt, wodurch SOAP weitestgehend technologie- und plattformneutral ist. Zur Übertragung wird auf etablierte Netzwerkprotokolle zurückgegriffen, wie z.B. HTTP [2] oder SMTP [3].

Die Entwicklung und Standardisierung von SOAP wird und wurde hauptsächlich von Privatunternehmen durchgeführt (Microsoft, IBM, Lotus und andere). Der derzeitig Entwicklungsstand ist SOAP 1.1 [4] das als Vorschlag beim W3C Konsortium [5] eingereicht worden ist. Eine offizielle Standardisierung steht noch aus.

1.3 Gliederung

Im nächsten Abschnitt werden die technischen Eigenschaften von SOAP beschrieben wie sie in der SOAP 1.1 Spezifikation definiert sind. Anschließend betrachten wir im dritten Abschnitt die typischen Einsatzgebiete dieses neuen Protokolls. Der vierte Abschnitt beschreibt beispielhaft eine konkrete SOAP Implementierung und eine darauf aufbauende einfache Client/Server-Anwendung. Der letzte Abschnitt enthält schließlich eine detaillierte Betrachtung der Vor- und Nachteile die durch den Einsatz von SOAP entstehen.

2 Die SOAP 1.1 Spezifikation

Die zur Zeit gültige SOAP Spezifikation [4] umfasst die folgende Elemente:

- die Struktur des XML-Dokumentes, das eine SOAP Nachricht enthält
- ein Mechanismus zur Kodierung von gängigen Datentypen in XML-Konstrukten

- der Einsatz von HTTP als Transportprotokoll für SOAP Nachrichten
- eine Konvention zur Durchführung von RPC

Als SOAP Nachricht bezeichen wir den Verbund von Nutzdaten und der umgebenden Struktur, also das gesamte XML-Dokument. Die folgenden Unterabschnitte werden nun die einzelnen Teilbereiche der Spezifikation näher betrachten.

2.1 Die Struktur einer SOAP Nachricht: Envelope, Header und Body

Der gesamte Inhalt einer SOAP Nachricht ist in einem einzigen (SOAP Nachrichten dürfen keine *Document Type* Deklarationen haben) Wurzelelement enthalten, dem SOAP *Envelope* Element. SOAP verwendet Namespaces [13] um Konflikte zwischen den im SOAP Standard definierten Elementen und fremden XML-Schemata zu vermeiden. Das *Envelope* Element liegt deswegen wie alle anderen Standardelemente im Namespace „http://schemas.xmlsoap.org/soap/envelope/“.

Das *Envelope* Element enthält als direkte Subelemente den optionalen SOAP *Header* und den SOAP *Body*. Die Subelemente des *Body* Elements repräsentieren die eigentliche Informationen der SOAP Nachricht die für den endgültigen Empfänger bestimmt sind. Der Header enthält Metainformationen die für den endgültigen Empfänger oder einen der Zwischenhändler bestimmt sein können. Mit Hilfe dieser Metainformationen kann SOAP um Funktionalitäten erweitert werden, die im eigentlichen Standard nicht vorgesehen sind, wie z.B. Authorisierungs- oder Transaktionsmanagmentverfahren. Zwei spezielle vordefinierte Attribute regeln dabei für wen die Metainformation bestimmt ist, und was passiert wenn der Empfänger dieses Element nicht verarbeiten kann.

Es ergibt sich also folgende Struktur:

```
 <SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Header>
     <!-- ..... -->
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
     <!-- ..... -->
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.1.1 Das *actor* Attribut

SOAP Nachrichten müssen nicht direkt an einen Empfänger geschickt werden, sondern können über eine Reihe von Zwischenhändlern (SOAP intermediaries) weitergeleitet werden. Wie dabei der Weg bestimmt wird den die Nachricht nimmt wird in der Spezifikation nicht festgelegt, genausowenig wie explizites Verfahren zu Adressierung des endgültigen Empfängers definiert wird. Eine mögliche technische Lösung wäre es die Adressierung über Metainformationen im SOAP Header zu realisieren die von den Zwischenhändlern ausgewertet werden. Typischerweise wird der Weg der Nachricht aber durch das eingesetzte Transportprotkoll bestimmt. Im Falle von HTTP wird der endgültige Empfänger durch die URL festgelegt über die der HTTP Server erreichbar ist. Ein möglicher Zwischenhandler wäre ein Proxy über den die HTTP Verbindung aufgebaut wird.

Nicht alle Teile der Nachricht müssen für den endgültigen Empfänger bestimmt sein. Deswegen leiten die Zwischenhändler eingehende Nachrichten nicht einfach blind weiter, sondern parsen die Nachricht und werten den für sie bestimmten Teil aus. Bevor sie die Nachricht weiterleiten, entfernen sie alle von ihnen bearbeiteten Teile (und fügen gegebenenfalls neue Elemente ein).

Jede am Transport einer SOAP Nachricht beteiligte Station und auch der endgültige Empfänger sind über eine URI identifizierbar. Das *actor* Attribut enthält eine solche URI und kann in jedem Subelement des SOAP Headers verwendet werden. Mit ihm wird festgelegt, wer dieses Element auswerten soll. Falls es fehlt, ist das Element für den endgültigen Empfänger der SOAP Nachricht bestimmt. Alle Elemente, die im SOAP *Body* enthalten sind, haben keine *actor* Attribute und sind somit automatisch für den endgültigen Empfänger bestimmt. Anstatt einer „echten" URI kann auch die spezielle URI "http://schemas.xmlsoap.org/soap/actor/next" verwendet werden. Diese bezeichnet den nächsten Empfänger in der Kette.

2.1.2 Das *mustUnderstand* Attribut

Das *mustUnderstand* Attribut bestimmt, ob das dazugehörige Element vom Empfänger zwingend verarbeitet werden muß. Es kann die Werte „0" und „1" annehmen und auf alle Subelemente des SOAP Headers angewandt werden. Die Voreinstellung ist „0" für Header Elemente, und unveränderlich „1" für alle Body Elemente. Trifft eine SOAP Implementierung auf ein Element mit gesetztem *mustUnderstand* Attribut, ist sie gezwungen die Verarbeitung abzubrechen, falls sie mit dem Element nichts anzufangen weiß. Dieser Mechanismus macht die dezentrale Erweiterung des SOAP Protokolls über zusätzliche Metadaten im Header erst richtig praktikabel, weil es vermeidet, dass kritische Inkompatibilitäten unbemerkt bleiben.

2.1.3 Das SOAP *Fault* Element

Die Subelemente des SOAP Bodies werden erst durch die konkrete Anwendung/Implementierung von SOAP bestimmt, der Standard definiert nur ein Elementtyp, den SOAP *Fault*. Er dient zur Übermittlung von Fehler- und Statusinformationen. Folgende Subelemente sind standardisiert:

faultcode Dieses Element enthält einen Code, der es erlaubt die Fehlersituation algorithmisch zu identifizieren. Der SOAP Standard spezifiziert ein Schema zur Bildung dieser Codes [14].

faultstring In diesem Element ist eine Fehlerbeschreibung enthalten, die an Endbenutzer ausgegeben werden kann.

faultactor Dieses Element enthält die URI des Zwischenhändlers der den Fehler ausgelöst hat.

detail Falls der Fehler durch ein Element des SOAP Body ausgelöst wurde, müssen in diesem Element applikationsspezifische Detailinformationen zum Fehler enthalten sein.

Ein Beispiel für ein *Fault* Element, das während einer Verkaufstransaktion auftreten könnte:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client.UnknownPart</faultcode>
  <faultstring>The part number is unknown.</faultstring>
  <detail>
    <e:myfaultdetails xmlns:e="Some-URI">
      <message>Part not in stock.</message>
      <errorcode>1001</errorcode>
    </e:myfaultdetails>
  </detail>
</SOAP-ENV:Fault>
```

Abgesehen von diesem Mechanismus zur Signalisierung von Fehlern bei der Verarbeitung befasst sich die Spezifikation nicht mit bei der Übertragung möglicherweise auftretenden Fehlern, wie z.B. verlorenen Nachrichten. Die fehlerfreie Übertragung muß durch das unterliegende Transportprotokoll sichergestellt werden. Falls dies nicht der Fall ist (z.B. bei der Verwendung von SMTP), müssen diese Aufgaben von den Anwendungen die SOAP benutzen übernommen werden.

2.2 Datenkodierung

Die SOAP Spezifikation enthält auch ein speziell auf diesen Anwendungszweck zugeschnittenes Verfahren zur Abbildung von gängigen Datentypen auf XML-Konstrukte. Diese Kodierung ist allerdings nur eine Empfehlung. Das *encodingStyle* Attribut eines Elements kann eine Liste von URIs beinhalten, die die relevanten Regeln zur Dekodierung der Daten identifizieren. *http://schemas.xmlsoap.org/soap/encoding/* steht dabei für das vom Standard beschriebene Format. Das *encodingStyle* Attribut kann in jedem Element enthalten sein und ist für dessen Inhalt und alle Subelemente gültig.

Die Kodierung baut auf die W3C Empfehlung zu XML Schemata [15], [16] auf. Spezifiziert werden neben einfachen Typen wie Integer, Float, String und Enumerationen auch komplexere Typen wie (mehrdimensionale) Felder und Strukturen. Felder und Strukturen können beliebig ineinander verschachtelt werden.

2.3 HTTP als Transportprotokoll

Prinzipiell können beliebige Netzwerkprotokolle zur Übermittlung von SOAP Nachrichten verwendet werden. Die SOAP 1.1 Spezifikation beschränkt sich auf HTTP, üblich ist außerdem die Verwendung von SMTP.

Die Kommunikation über HTTP erfolgt indem der Client über den *POST* Befehl eine SOAP Nachricht, die die gewünschte Anfrage enthält, an den Server schickt. Dieser schickt anschließend über die selbe Verbindung eine SOAP Nachricht mit dem Ergebnis der Anfrage.

Der Header des *POST* Befehls enthält neben üblichen *Content-Type* und *Content-Length* Feldern ein zusätzliches Feld *SOAPAction.* Dieses Feld enthält eine URI die den Zweck der übermittelten SOAP Nachricht beschreiben soll. Dieses Feld hat keinen Einfluß auf die Interpretation der Nachricht, kann aber Dritten (Firewalls, Proxies) als Information über den Inhalt der Nachricht dienen.

Im Falle einer erfolgreichen Bearbeitung wird vom Server für die Antwort ein Statuscode aus dem Bereich 2xx benutzt. Im Fehlerfall muß der Fehlercode 500 zusammen mit einer SOAP Nachricht, die ein entsprechendes SOAP *Fault* Element enthält, übermittelt werden.

Die Beispieltransaktion aus der Einleitung würde wie folgt durchgeführt werden:

```
POST /Order HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://shop.com/part?action=order"

<SOAP-ENV:Envelope...

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope...
```

Zusätzlich wird auch noch die Verwendung des HTTP Extension Framework [17] spezifiziert, welches eine detaillierte Beschreibung der versandten Nachricht ermöglicht.

2.4 RPC

Ein RPC Methodenaufruf besteht aus einer SOAP Nachricht die den Namen der gewünschten Methode und ihre Parameter enthält und einer zweiten mit dem Ergebnis des Aufrufs. Falls HTTP verwendet wird, ergibt sich dieser Dialog automatisch, bei anderen Protokollen müssen äquivalente Lösungen gefunden werden (beispielsweise eine Antwortmail bei SMTP).

Der Methodenaufruf wird durch eine Struktur repräsentiert, die den Namen der Methode trägt. Als direkte Subelemente enthält sie die Parameter des Aufrufs, auch diese tragen den Namen der jeweiligen Methodenparameter. Die Nachricht enthält keine Information darüber, an welches Objekt bzw. Schnittstelle der Aufruf gerichtet ist. Diese Information wird über das unterliegende Transportprotokoll übermittelt (HTTP: Die URIs die im *POST* Befehl und im *SOAPAction* Header verwendet werden, SMTP: E-Mail-Adresse).

Das Ergebnis wird ebenfalls in einer Struktur übermittelt, die Subelemente für den Rückgabewert und alle weiteren Ausgabeparameter enthält. Der Name der Struktur und des Rückgabewerts sind dabei nicht relevant, es wird aber empfohlen, Methodenname+"Response" als Name für die Struktur zu verwenden. Falls ein Fehler bei der Bearbeitung aufgetreten ist, wird statt des Ergebnises eine SOAP *Fault* Element übertragen. Zusätzlich müssen in diesem Fall auch immer die Fehlerbehandlungsmechanismen des unterliegenden Transportprotokolls benutzt werden.

Natürlich können auch im RPC Kontext Nachrichten zusätzliche Informationen im SOAP Header enthalten, die beispielsweise zur Transaktionsverwaltung dienen können.

Hier nun zum letzten Mal die Beispieltransaktion aus der Einleitung, nun in korrektem SOAP Format:

```
POST /Order HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://shop.com/part?action=order"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
     <m:orderPart xmlns:m="http://shop.com/part?action=order">
       <partNumber>2345</partNumber>
       <quantity>5</quantity>
       <customerID>C4523</customerID>
     </m:orderPart>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
    <m:orderPartResponse xmlns:m="http://shop.com/part?action=order">
        <price>34.5</price>
    </m:orderPartResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Dieses RPC Verfahren ist ausdrücklich optional, falls seine Semantik also für ein konkrete Anwendung ungünstig ist, muß es nicht verwendet werden. Es wäre z.B. möglich einen Dienst zu implementieren der SOAP Nachrichten simultan per SMTP an viele verschiedene Empfänger verschickt und keine Antwort der Empfänger benötigt. Oder ein Empfänger kann eine Antwort zeitverzögert schicken nachdem er eine bestimmte Anzahl Nachrichten empfangen hat.

3 Typische Einsatzgebiete

Dieser Abschnitt beschreibt die zur Zeit üblichen Einsatzgebiete von SOAP.

3.1 Client/Server-Anwendungen

Aufgrund der inhärenten Leistungsverhaltens das im Abschnitt 5 im Detail behandelt werden, ist der Einsatz von SOAP vor allem bei Client/Server-Kommunikation über das Internet interessant:

- Die Übertragungswege weisen typischerweise recht hohe Latenzen auf, deshalb fällt die zusätzliche Verzögerung durch den erhöhten (De-) Kodierungsaufwand nicht allzu stark ins Gewicht.
- Die übertragenen Datenmengen sind typischerweise eher gering (im Vergleich zum LAN), deswegen ist das Einsparungspotential durch effizientere binäre Protokolle ebenfalls gering.

Der Hauptvorteil von SOAP liegt hierbei in der Kompatibilität mit Firewalls und Proxies. Selbst in stärker gesicherten Netzwerken wird in der Regel die Möglichkeit gegeben, über HTTP auf externe Rechner zuzugreifen. Und somit können von allen Rechnern in einem solchen LAN mehr oder weniger problemlos externe SOAP Dienste genutzt werden. Andere Protokolle würden die Freischaltung eines zusätzlichen, schwer zu kontrollierenden Ports auf der Firewall erfordern. Natürlich ist diese Lösung kein Ersatz für eine wirksame Sicherheitsstrategie. Ohne diese könnten über SOAP unkontrolliert Informationen aus dem gesicherte LAN gelangen und unerwünschte Zugriffe von Außen getätigt werden. Eine wirksame Firewallkonfiguration würde also alle HTTP Pakete stoppen müssen, die SOAP Nachrichten enthalten. Das textuelle Format von SOAP und Konstrukte wie das *SOAPAction* Header Feld ermöglichen es aber, SOAP Nachrichten sinnvoll zu filtern.

3.2 Transportprotokoll für komplexere APIs

SOAP kann als Transportprotokoll für .NET und CORBA eingesetzt werden. Durch die zusätzlichen Abstraktionsschichten kann je nach Anwendungsfall entschieden werden, ob SOAP oder ein binäres Protokoll verwandt wird, ohne dass die Programme angepaßt werden müssen. Der von Iona angebotene CORBA Orb [18] kann beispielsweise sowohl über IIOP als auch SOAP kommunizieren.

3.3 Integration inkompatibler RPC/RMI Mechanismen

Weiterhin kann SOAP mit Hilfe geeigneter Middleware dazu dienen, die Zusammenarbeit von inkompatiblen Softwarekomponenten (z.B. DCOM und CORBA) zu ermöglichen. Dazu werden alle Nachrichten und Funktionsaufrufe zunächst in äquivalente SOAP Nachrichten umgewandelt und beim Empfänger wieder in das passende native Format umgewandelt.

4 Beispiel

Die vorangegangen Abschnitten sollen nun durch ein umfassendes Beispiel illustriert werden. Dieses besteht aus einer Client/Server Anwendung, die alle wesentlichen Eigenschaften von SOAP ausnutzt:

- Technologieunabhängigkeit
  Der Client wird in C++ ohne vorgefertigte SOAP-Bibliotheken implementiert, der Server in Java unter Zuhilfenahme einer passenden fertigen SOAP Implementierung.

- Erweiterung der SOAP Funktionalität durch zusätzliche Einträge im SOAP Header
- Benutzung von HTTP als Transportprotokoll
- Verwendung des vorgegeben RPC Verfahrens

4.1 Automatisierte Abwicklung einer Auktion

Die Beispielanwendung soll die automatische Abwicklung von Auktionen durch ein Internet-Auktionshaus simulieren. Der Server wird mit Hilfe von Apache SOAP [19] in Java implementiert. Er verwaltet folgende Daten:

- eine Liste der registrierten Benutzer mit Name und Passwort
- eine Liste der laufenden Auktionsangebote mit Id, Beschreibung und einer Liste der zugehörigen Gebote

Über SOAP bietet der Server seinen Clients folgende Schnittstelle:

String login(String userName, String pass)	Mit Hilfe dieser Methode meldet sich der Client zu Beginn einer Sitzung am Server an. Im Falle einer erfolgreichen Anmeldung wird ein String zurückgegeben, mit dem sich der Client für den Rest der Sitzung ausweisen kann. Diesen String fügt er im SOAP Header jeder weiteren Anfrage ein.
void logout()	Hiermit beendet der Client die aktuelle Sitzung.
Vector getItemList()	Diese Methode liefert eine Liste der laufenden Auktionen, jeweils mit Id, Beschreibung und dem aktuellem Höchstgebot.
double getCurrentBid(int itemId)	Über diese Methode kann der Client das aktuelle Höchstgebot für einen Auktionsgegenstand erfragen.
void makeBid(int itemId, double amount)	Hiermit gibt der Client ein neues Gebot ab.

Zusätzlich zur öffentlichen Schnittstelle bietet der Server eine interne Schnittstelle an, die es erlaubt den aktuellen Auktionsverlauf über JSP zu visualisieren.

Der Client soll eine grafische Benutzeroberfläche bieten und wird in C++ auf Basis von Qt/KDE implementiert. Er gibt dem Benutzer die Möglichkeit, Name und Passwort einzugeben, diese Daten werden dann zur Anmeldung beim Server genutzt. Der Client bezieht die Liste der laufenden Auktionen vom Server und stellt sie in einer Tabelle dar. Über einen gesonderten Dialog ist es nun möglich, den Client anzuweisen, das Gebot automatisch periodisch zu erhöhen.

4.2 Apache SOAP

Mit die wichtigste SOAP Implementierung für Java ist Apache SOAP [19], welches ursprünglich von IBM entwickelt wurde und nun Teil des Apache Projekts ist. Zur Verfügung steht sowohl eine Bibliothek zur Programmierung von Clients, als auch die nötige serverseitige Infrastruktur zur Implementierung von SOAP Diensten. Die Serverkomponenten von Apache SOAP sind als Servlets implementiert, im Einsatz als Server setzt Apache SOAP deshalb auf einen Servlet Container wie z.B. Tomcat auf. Dabei werden sowohl Nachrichten- als auch RPC-basierte Dienste unterstützt. Das Beispiel wird als RPC Dienst

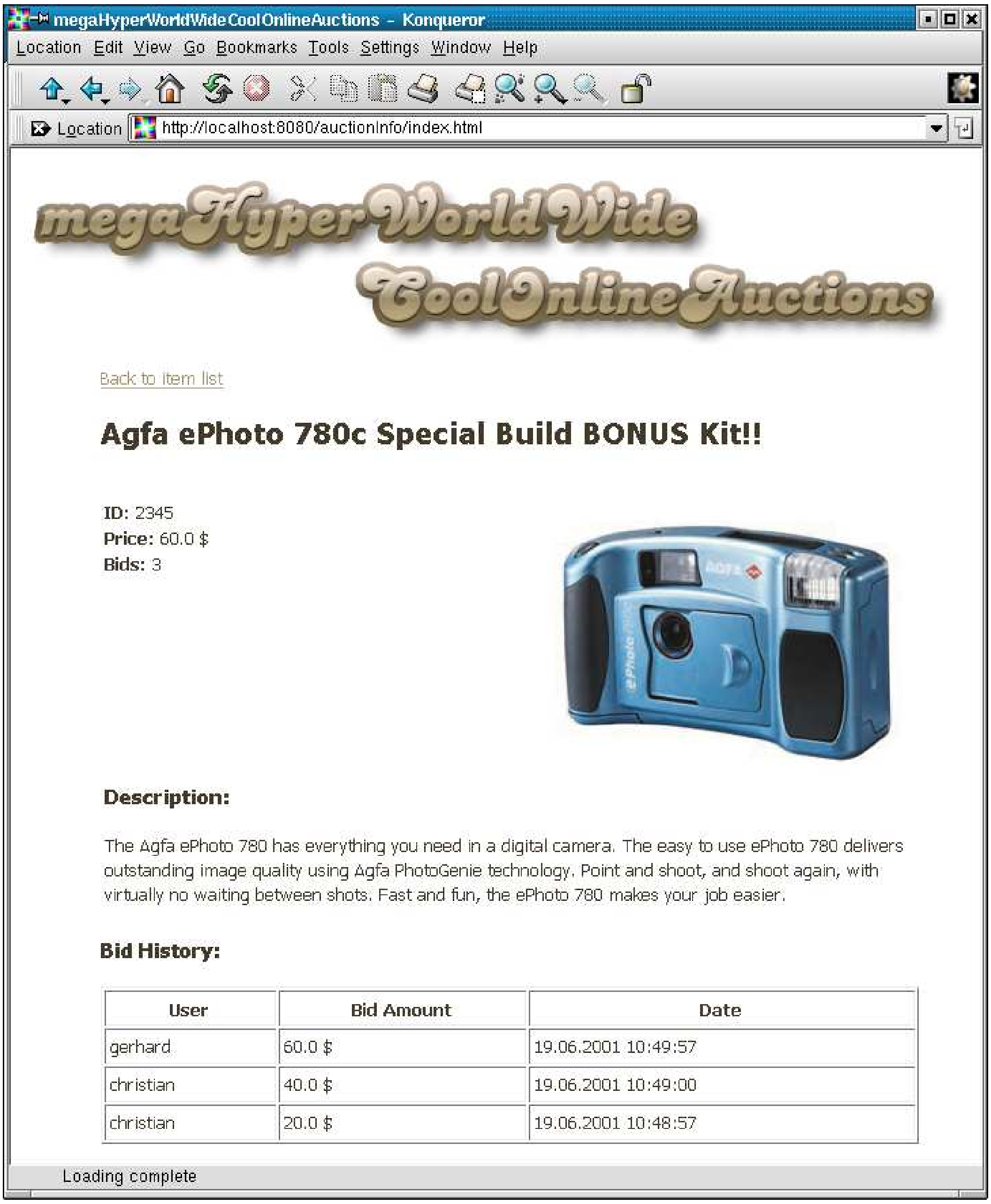

Abbildung 1: Darstellung des Auktionsverlaufs im Webbrowser

implementiert werden. Als Übertragungsprotokolle können HTTP und SMTP verwendet werden.

Die Installation eines neuen Dienstes erfolgt, indem dem Administrationswerkzeug eine XML-Datei übergeben wird, die die Eigenschaften des neuen Dienstes beschreibt (alternativ gibt es ein webbasiertes Frontend, das die Informationen über ein Formular entgegen nimmt). Die Datei enthält unter anderem folgende Informationen:

- die URI über die der neue Dienst erreichbar sein soll
- der Name der Java-Klasse, die den Dienst implementiert
- eine Liste der Methoden die von außen zugänglich seinen sollen

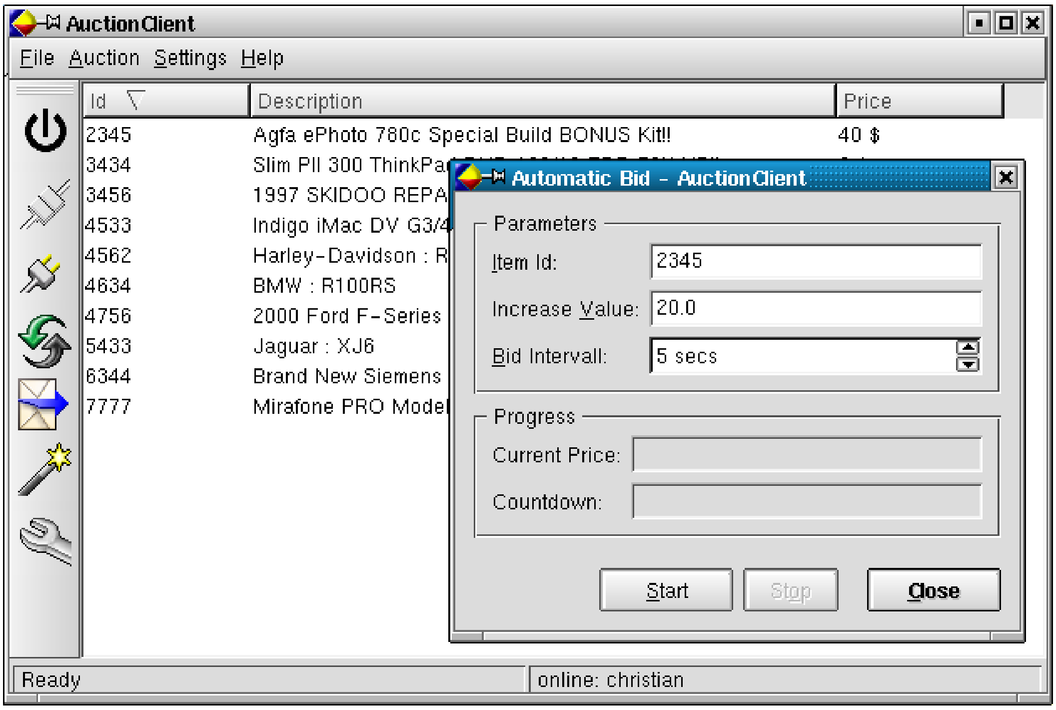

Abbildung 2: Der Client des Auktionssystems

- eine oder mehrere Klassen die die Fehlerbehandlung übernehmen
- Deklarationen die die automatische Abbildung von Java-Typen auf XML-Konstrukte ermöglichen

Konkret hat diese Datei für den Beispieldienst folgenden Inhalt:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
                   id="auctionServer">
  <isd:provider type="auction.AuthProvider"
                scope="Application"
              methods="login logout getItemList getCurrentBid makeBid">
    <isd:java class="auction.Server" static="true"/>
  </isd:provider>

  <isd:faultListener>org.apache.soap.server.DOMFaultListener
  </isd:faultListener>

  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:x="auctionServer" qname="x:ItemInfo" javaType="auction.ItemInfo"
java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
  </isd:mappings>
</isd:service>
```

Eingehende Nachrichten werden von Apache SOAP geparst und in Methodenaufrufe umgesetzt. Standardmäßig können dabei eine große Anzahl von einfacheren Datentypen (int, float, boolean, byte, Arrays, String, Date, Vector, Map, etc.) automatisch dekodiert werden. Für fehlende Typen können eigene Parser geschrieben und an entsprechender Stelle registriert werden. Tritt während des Methodenaufrufs ein Fehler auf, kann die Methode eine spezielle *SOAPException* auslösen. Diese wird von den registrierten Fehlerbehandlungsklassen in einen SOAP *Fault* umgesetzt. Ansonsten wird der Rückgabewert der Methode an den Empfänger gesandt.

4.3 Eine Beispielsitzung

Zunächst muß sich der Client mit Benutzername und Passwort authentifizieren, anschließend gibt der Server ihm den Authorisierungs-String für die neue Sitzung:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  xmlns:xsi="http://www.w3.org/1999/
instance"    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <ns1:login xmlns:ns1="auctionServer">
    <loginName xsi:type="xsd:string">christian</loginName>
    <password xsi:type="xsd:string">gebauer</password>
  </ns1:login>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:xsi="http://www.w3.org/1999/XMLSchema-
instance"  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <ns1:loginResponse xmlns:ns1="auctionServer"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return xsi:type="xsd:string">christian2058954590</return>
  </ns1:loginResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die automatische Dekodierung von Parametern kann Apache SOAP nur dann bewerkstelligen, wenn die Parameter über das *type* Attribut selbstbeschreibend dargestellt werden. Falls diese Attribute fehlen muß für jeden Parameter der Typ über das Konfigurationswerkzeug spezifiziert werden. Eine weitere spezifische Eigenart dieser SOAP Implementierung ist, dass der Namespace des Methodenaufrufs das Zielobjekt bestimmt. Hier wird der Namespace „auctionServer“ verwendet, korrespondierend zu der Definition des Dienstes „auctionServer“ über das Konfigurationswerkzeug.

Anschließend kann der Client nun eine Liste der Auktionsgegenstände abrufen:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Header>
  <ns1:SessionAuthorizationToken
   SOAP-ENV:mustUnderstand="1"
   xmlns:ns1="auctionServer"
   xsi:type="xsd:string">christian2058954590</ns1:SessionAuthorizationToken>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <ns1:getItemList xmlns:ns1="auctionServer">
  </ns1:getItemList>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Im SOAP Header kann man den Authorisierungs-String erkennen. Das Element das den eigentlichen Methodenaufruf symbolisiert hat keine Subelemente, da keine Parameter benötigt werden.

Die Liste der Auktionsgegenstände die der Server nun zurückliefert verdeutlicht, dass mit SOAP auch komplexere Datenstrukturen übermittelt werden können. Der Rückgabewert ist ein Vektor, der für jeden Gegenstand eine Struktur mit Id, Beschreibung und den aktuelle Gebot enthält:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-
instance"  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <ns1:getItemListResponse xmlns:ns1="auctionServer"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <return xmlns:ns2="http://xml.apache.org/xml-soap"
           xsi:type="ns2:Vector">
     <item xsi:type="ns1:ItemInfo">
       <description xsi:type="xsd:string">Agfa ePhoto</description>
       <price xsi:type="xsd:double">60.0</price>
       <id xsi:type="xsd:int">2345</id>
     </item>

<!-- ..... --!>

     <item xsi:type="ns1:ItemInfo">
       <description xsi:type="xsd:string">Slim PII 300</description>
       <price xsi:type="xsd:double">0.0</price>
       <id xsi:type="xsd:int">3434</id>
     </item>
   </return>
  </ns1:getItemListResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nun kann der Benutzer ein Gebot abgeben:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Header>
  <ns1:SessionAuthorizationToken
   SOAP-ENV:mustUnderstand="1"
   xmlns:ns1="auctionServer"
   xsi:type="xsd:string">christian2058954590</ns1:SessionAuthorizationToken>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <ns1:makeBid xmlns:ns1="auctionServer">
    <itemId xsi:type="xsd:int">2345</itemId>
    <amount xsi:type="xsd:double">50</amount>
  </ns1:makeBid>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Falls dieses angenommen werden kann, sendet der Server eine leere Nachricht als Bestätigung, da die Methode keinen Rückgabewert hat. Ist das Gebot aber ungültig, weil z.B wie in diesem Fall der Betrag des Gebots zu gering ist, antwortet der Server mit einem SOAP Fault:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <SOAP-ENV:Fault>
    <faultcode>AuctionServer.bidAmountTooLow</faultcode>
    <faultstring>Bid amount too low.
                 You have to make a higher bid</faultstring>
    <faultactor>/soap/servlet/rpcrouter</faultactor>
    <detail>
      <stackTrace>
```

```
[SOAPException: faultCo-
de=AuctionServer.bidAmountTooLow; msg=Bid amount too low. You ha-
ve to make a higher bid]
        at auction.Server.makeBid(Server.java:304)
        at java.lang.reflect.Method.invoke(Native Method)
        at org.apache.soap.server.RPCRouter.invoke(RPCRouter.java:146)

 ....

      </stackTrace>
    </detail>
  </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.4 Fazit

Alle aufgestellten Anforderungen konnten in zwei Tagen implementiert werden. Der Aufwand hielt sich dabei in erträglichen Grenzen: der Server inklusive Visualisierung über JSP umfasst etwa 700 Zeilen Quellcode, der Client etwa 1000 Zeilen. Dieses Ergebnis zeigt, dass die vorhanden Implementierungen praktisch einsetzbar sind, zumindestens für Probleme ähnlicher Größenordnung. Weiterhin zeigt sich, dass die angestrebte Unabhängigkeit von konkreten Programmiersprachen und anderen Technologien erreicht worden ist. Einen Client für ein Protokoll wie z.B. IIOP zu implementieren hätte den zeitlichen Rahmen höchstwahrscheinlich gesprengt.

5 Betrachtung der Vor- und Nachteile

Im letzen Abschnitt werden wir nun die Vor- und Nachteile die sich durch den Einsatz von SOAP ergeben können betrachten.

5.1 Allgemeine Vorteile XML-basierter Protokolle

Eine Reihe von Vorteilen die SOAP bietet, sind allen XML-basierten Protokollen gemein:

- Ein selbstbeschreibendes Datenformat wie XML ist ein hervorragendes Mittel zur Minimierung von unerwünschter Koppelung zwischen Softwarekomponenten
- Reine ASCII-Formate wie XML erleichtern die Fehlersuche während der Entwicklung und können von Proxies und Firewalls eher verstanden werden als binäre Protokolle.
- XML wird in zunehmenden Maße als natives Format zur Datenhaltung verwendet. Ist dies der Fall entsteht bei der Verwendung von SOAP kein zusätzlicher Konvertierungsaufwand. Inkompatible Schemata können dabei mit Werkzeugen wie XSLT [20] recht einfach in passende Formate umgewandelt werden.

5.2 Reifegrad

SOAP 1.1 ist kein fertiger Standard. Es ist deshalb möglich, dass sowohl an SOAP als auch an dessen Implementierungen noch größere Änderungen durchgeführt werden. Unter Umständen müssen also heute entwickelte SOAP Programme in naher Zukunft bereits umgeschrieben werden.

Das W3C hat eine Arbeitsgruppe [21] zur Schaffung eines einheitlichen XML Protokolls eingesetzt. Die veröffentlichten Anforderungsdokumente [22] lassen darauf schließen, dass der neue Standard im Wesentlichen eine Erweiterung von SOAP 1.1 sein wird.

Insofern ist zu erwarten, dass heutige SOAP Anwendungen mit vertretbarem Aufwand an den kommenden Standard angepaßt werden können.

5.3 Leistungsverhalten

Ungenügende Leistungsfähigkeit eines Protokolls kann in vielen Anwendungsbereichen ein Ausschlußkriterium sein, sobald gewisse kritische Grenzen übertreten werden. Das System einer Bank zur Abwicklung von Überweisungen muß beispielsweise in der Lage sein, die pro Tag übliche Anzahl an Transaktionen innerhalb von 24 Stunden zu bearbeiten.

Leider sind nur wenige Untersuchungen zum Leistungsverhalten von SOAP im Vergleich zu anderen Protokollen verfügbar [23] [24] [25]. Die detaillierteste Studie ist an der Universität von Indiana durchgeführt worden und beschäftigt sich mit der Eignung von SOAP zur Übermittlung von Messdaten für wissenschaftliche Zwecke [23]:

- Der erreichbare Datendurchsatz über SOAP ist um den Faktor 10 geringer als bei binären Protokollen. Im Vergleich zu einer direkten Übertragung per Socket ohne Protokoll ergibt sich der Faktor 100. Dieser Unterschied ist vor allem in den Größenunterschieden der kodierten Daten begründet, und wird sich deshalb durch effizientere Implementierungen kaum verringern lassen.
- Der Rechenaufwand zur Kodierung und Dekodierung der Daten in SOAP Nachrichten ist um den Faktor 100 höher als bei einer binäre Repräsentation wie sie von Java-RMI verwendet wird. Dieser Aspekt kann durch effizientere Implementierungen optimiert werden. Beispielsweise soll der XML-Parser „Electric XML“ [26] in der Lage sein, SOAP Nachrichten mehr als 10 mal schneller zu parsen als Apache Xerces [27]. Dementsprechend hat die darauf aufbauende SOAP Implementierung „GLUE“ [28] nach Herstellerangaben ein recht interessante Leistungsverhalten: „100,000+ messages/second when service and invoker are in the same VM. 600+ messages/second when service and invoker are both on local LAN“

Im wesentlichen werden aber XML basierende Protokolle wohl immer „teurer“ sein als binäre Protokolle. Befürworter von SOAP behaupten [29] aber mit Recht, dass sich dieser Nachteil im Anbetracht der stetig steigenden Hardware-Ressourcen in Zukunft relativieren wird.

5.4 Umfang des Standards

Eines der wesentlichen Merkmale von SOAP ist der relativ geringe Umfang des Standards. Dies ist prinzipiell zu begrüßen:

- SOAP ist leicht erlernbar, sofern man gewisse Vorkenntnisse über XML besitzt.
- Es ist zu erwarten, dass die angebotenen SOAP Implementierungen mehr oder weniger vollständig und korrekt sein werden. Dies ist bei komplexen Standards wie z.B. ISO C++ oder SQL99 offensichtlich nicht der Fall.

Gleichzeitig ist diese Designentscheidung natürlich auch ein großer Nachteil, denn so werden viele wichtige Aspekte wie z.B. Transaktionsmanagment und Sicherheitsfunktionen nicht durch den Standard abgedeckt. Dieses Manko wird teilweise dadurch ausgeglichen, dass in Form des SOAP Headers eine relativ flexible Möglichkeit zur Nachrüstung dieser Funktionalitäten angeboten wird.

5.4.1 Transaktionsmanagement

Fertige Lösungen zum Transaktionsmanagement werden zur Zeit wohl nicht angeboten. Falls SOAP nur als Transportprotokoll für komplexere APIs wie CORBA oder .NET benutzt wird, sollte es möglich sein die entsprechenden Dienste dieser APIs zu nutzen.

5.4.2 Authentisierung, Verschlüsselung und sonstige Sicherheitsmerkmale

Die SOAP 1.1 Spezifikation beschäftigt sich nicht mit Sicherheitsaspekten. Da auch die Anforderungsdokumente [22] zum zukünftigen XML Protokoll Standard dieses Thema kaum behandeln, wird sich dieser Zustand wohl auch nicht ändern.

Zur Lösung dieses Mißstandes gibt es mehrere Ansätze. Eine Strategie besteht darin, die Übertragung möglichst sicher zu gestalteten indem die Sicherheitsfeatures des unterliegenden Übertragungsprotokolls ausgenutzt werden, wie z.B. HTTPS und SSL.

Desweiteren kann natürlich auch das eigentliche XML Dokument verschlüsselt bzw. signiert werden. Aufbauend auf die entsprechenden Bemühungen [30], [31] des W3C gibt es einen Vorschlag [32] für ein neues SOAP Header Element <*SOAP-SEC:Signature*>.

Generell ist aber festzustellen, dass Verantwortung für die Sicherheit eines SOAP Dienstes momentan eindeutig beim Designer bzw. Entwickler liegt. Er muß sehr genau abwägen welche Funktionen er zur Verfügung stellt und welche Authorisierungsmechanismen er implementiert.

5.4.3 Fazit

Meiner Meinung nach wird der zukünftige Erfolg von SOAP entscheidend davon abhängen, ob eine ausreichende Zusammenstellung dieser Erweiterungen etabliert werden kann. Sollte dies nicht der Fall sein, wird in der Praxis dann wohl auf proprietäre Erweiterungen zurückgegriffen werden (die z.B. in .NET enthalten sein könnten). Dies würde die erhoffte Interoperabilität stark einschränken.

Literatur

[1] Extensible Markup Language (XML), W3C Recommendation -
http://www.w3.org/TR/REC-xml

[2] Hypertext Transfer Protocol (HTTP), RFC2616
http://www.faqs.org/rfcs/rfc2616.html

[3] Simple Mail Transfer Protocol (SMTP), RFC2821
http://www.faqs.org/rfcs/rfc2821.html

[4] Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[5] The World Wide Web Consortium (W3C)
http://www.w3.org/

[6] Remote Procedure Call Protocol Specification Version 2 (RPC), RFC1057
http://www.faqs.org/rfcs/rfc1057.html

[7] External Data Representation Standard (XDR), RFC1014
http://www.faqs.org/rfcs/rfc1014.html

[8] CORBA/IIOP Specification
http://www.omg.org/technology/documents/formal/corbaiiop.htm

[9] Object RPC (ORPC)
http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm#dcomarch_objrpc

[10] Distributed Component Object Model (DCOM)
http://www.microsoft.com/com/tech/DCOM.asp

[11] Fundamentals of RMI - Short Course
http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html

[12] XML-RPC Specification
http://www.xmlrpc.org/spec

[13] Namespaces in XML, W3C Recommendation
http://www.w3.org/TR/REC-xml-names/

[14] SOAP Fault Codes
http://www.w3.org/TR/SOAP/#_Toc478383510

[15] XML Schema Part 1: Structures, W3C Recommendation
http://www.w3.org/TR/xmlschema-1/

[16] XML Schema Part 2: Datatypes, W3C Recommendation
http://www.w3.org/TR/xmlschema-2/

[17] HTTP Extension Framework, RFC2774
http://www.faqs.org/rfcs/rfc2774.html

[18] IONA
http://www.iona.com

[19] Apache SOAP
http://www.apache.org/soap/

[20] XSL Transformations (XSLT), W3C Recommendation
http://www.w3.org/TR/xslt

[21] XML Protocol Working Group
http://www.w3.org/2000/xp/Group/

[22] XML Protocol (XMLP) Requirements, W3C Working Draft
http://www.w3.org/TR/xmlp-reqs/

[23] Requirements for and Evaluation of RMI Protocols for Scientific Computing
http://www.extreme.indiana.edu/soap/sc00/paper/index.html

[24] Ein weiterer Benchmark
http://www.vbpj.com/upload/free/features/vbpj/2000/10oct00/ys0010/ys0010-6.asp

[25] xmlBlaster Performance
http://www.xmlBlaster.org/performance.html

[26] Electric XML
http://www.themindelectric.com/products/xml/xml.html

[27] Electric XML Benchmarks
http://www.themindelectric.com/products/xml/benchmarks.html

[28] The GLUE Platform
http://www.themindelectric.com/products/glue/glue.html

[29] Performance of SOAP
http://scriptingnews.userland.com/backissues/2000/06/27#performanceOfSoap

[30] XML Encryption Working Group
http://www.w3.org/Encryption/2001/

[31] XML-Signature Syntax and Processing, W3C Candidate Recommendation
http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/

[32] SOAP Security Extensions: Digital Signature, W3C NOTE
http://www.w3.org/TR/SOAP-dsig/