

Smart Clients, Teil 4: Web Services manchmal besser ohne SOAP?

Bremse oder Gaspedal?

VON CHRISTIAN CAMPO

Web Services auf der Basis von SOAP sind heute Standard, Smart Clients auf Basis von Web Services im Kommen. Liefert die Formel SOAP + Smart Client immer das beste Ergebnis? Oder verfolgen sie unterschiedliche Ziele? Und was sind die speziellen Anforderungen von Smart Clients an Web Services?

Smart Clients halten ihre Daten, im Gegensatz zu Fat Clients, nicht lokal. Sie greifen über Web Services auf zentral verfügbare Fachkomponenten und dahinter liegende Datenbanken zu. Dadurch haben Smart Clients besonders hohe Anforderungen an Web-Service-Implementierungen. Sie müssen effektiv zwischen lokalen und auf dem Server liegenden Komponenten vermitteln. Gleichzeitig müssen sie den hohen Benutzungskomfort mit seinen kurzen Reaktionszeiten in unterschiedlichen Online-Szenarien aufrechterhalten.

Sei es LAN, DSL, aber auch UMTS oder GPRS, Web Services dürfen nicht zur Bremse der Applikation werden. Außerdem darf man bei der Betrachtung von Web Services die Kosten, die zum Beispiel bei GPRS nach Volumen abgerechnet werden, nicht aus den Augen verlieren.

SOAP ist der Standard

Für die Umsetzung von Web Services hat sich SOAP als Standard durchgesetzt. Im

Smart Clients – die Serie

- Teil 1: Was ist ein Smart Client? Definition und Abgrenzung
- Teil 2: Aspekte einer Softwarearchitektur für Smart Clients
- Teil 3: Effektive UI-Entwicklung für Smart Clients
- Teil 4: Web Services manchmal besser ohne SOAP?
- Teil 5: Smarte Softwareaktualisierung

Open-Source- wie im kommerziellen Bereich gibt es eine große Auswahl an guten SOAP-Implementierungen. Für die Interoperabilität (eines der wichtigsten Ziele von SOAP) zwischen verschiedenen Implementierungen innerhalb und jenseits von Java ist SOAP unverzichtbar geworden. Das WS-I-Konsortium bemüht sich deshalb um weitere Verbesserung der Interoperabilität.

Smart Clients und SOAP

Die Nutzung von SOAP in einer Web-Service-orientierten Welt ist weit verbreitet und erscheint auch für Smart Clients nahe liegend. Die breite Unterstützung durch Frameworks und Tools und die vorhandenen Schnittstellen in bereits existierenden zentralen Anwendungen innerhalb und außerhalb des eigenen Unternehmens unterstützen eine Entscheidung in diese Richtung.

Allerdings spielt SOAP seine Vorteile gerade in heterogenen Umgebungen aus, in denen viele Komponenten aus verschiedenen Umgebungen und Plattformen integriert werden müssen. SOAP stellt jedoch kein schlankes Format dar. Es wurde weder auf Verarbeitungsgeschwindigkeit hin spezifiziert noch für geringe Bandbreiten optimiert.

Aber welche anderen Web-Service-Protokolle gibt es noch, die besser für Smart Clients geeignet sind? Möglicherweise ist es erst einmal hilfreich, einige Smart-Client-Anwendungen zu betrachten, die jeder kennt, und zu untersuchen,

wie dort die Kommunikation zwischen Smart Client und Server gelöst ist.

Exkurs: Bekannte Smart-Client-Anwendungen

Kennen Sie eine Smart-Client-Anwendung oder haben Sie schon eine benutzt?

SOAP ist ...

SOAP [1] ist ein XML-basiertes Web-Service-Protokoll, das aktuell in der Version 1.2. beim W3C spezifiziert ist und eine breite Unterstützung gefunden hat. Es definiert unabhängig von der verwendeten Programmiersprache und Plattform ein Datenformat für den Nettodaten-austausch zwischen Programmen. SOAP ist durch die Verwendung von HTTP internetfähig. SOAP-Schnittstellendefinitionen werden in WSDL [2] formuliert. Dort können, neben den Web-Service-Methoden, die verwendeten Datenstrukturen mit XML-Schema definiert werden. Die Aufgabe der einzelnen Implementierungen ist es dann, die sprachneutralen Definitionen in WSDL bzw. im XML-Schema auf sprachabhängige Datentypen abzubilden oder dafür Konfigurationsmöglichkeiten zu schaffen. Es gibt fünf so genannte SOAP Encodings, mit denen SOAP-Daten kodiert werden können:

- *rpc/encoded*
- *rpc/literal*
- *document/encoded*
- *document/literal*
- *wrapped*

Während früher oft *rpc/encoded* verwendet wurde, haben sich in letzter Zeit *document/literal* und *wrapped* durchgesetzt, da beide eine viel präzisere Definition des serialisierten XML-Formats erlauben.

Wahrscheinlich schon und vielleicht ohne diese als solche zu erkennen. Apple iTunes Musicstore oder Google Earth sind prominente Beispiele von Smart Clients, die als lokale Anwendung auf zentral gespeicherte Daten über das Internet zugreifen.

iTunes lädt Metadaten aus seinem Musicstore, tätigt Einkäufe und lädt Songs in seine lokale Musikbibliothek. Die Schnittstelle zum zentralen Server ist nicht offen gelegt, es ist jedoch bekannt [3], dass HTTP und XML (aber nicht SOAP) benutzt werden. Aus der Sicht von Apple gibt es keine Notwendigkeit zur Benutzung von SOAP: weder für den eigenen Gebrauch noch als öffentliche Schnittstelle für andere.

Google Earth bietet zwar zahllose Möglichkeiten für die Erweiterung mit Plug-ins und der Verbindung mit eigenen Daten, aber die Schnittstelle zwischen dem Google Earth Smart Client und seinem Backend bleibt verborgen. Sicher ist nur, dass er HTTP und damit ein Standard-Internet-Protokoll benutzt.

Das sind zwei Smart Clients, die eng mit ihrem Backend integriert arbeiten und keinen Vorteil in der Benutzung eines offenen Protokolls wie SOAP sehen.

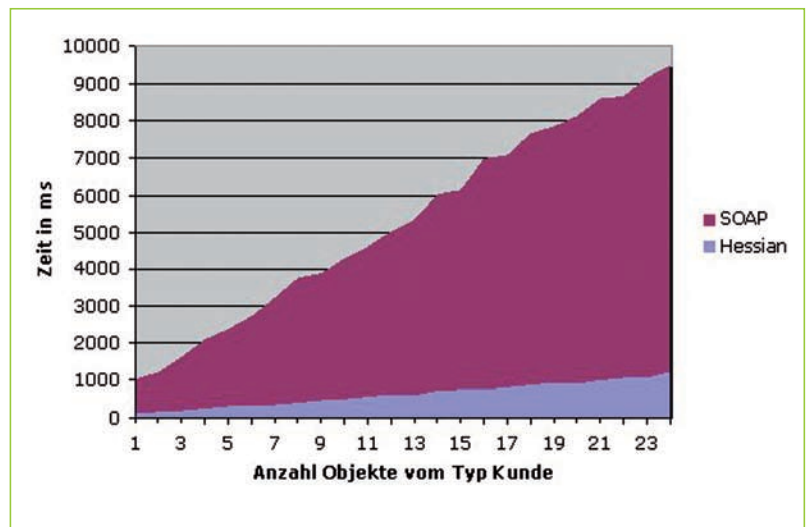
Alternative Ansätze zu SOAP

Um die Anforderungen eines Smart Clients an die Kommunikation mit dem Backend zu erfüllen, gibt es mehrere Ansätze für ein effektiveres, besser geeignetes Transportformat. Einige dieser Ansätze werden nun vorgestellt.

Binäres XML

Die erste Möglichkeit besteht darin, die einzelnen semantischen Teile einer XML-Instanz in einer nur maschinenlesbaren Form und damit deutlich kompakter zu kodieren – als binäres XML. Man behält die Semantik wie Tags, Attribute und Namespaces bei, verzichtet aber auf die ressourcenintensive Textform. Der Inhalt und die Metadaten bleiben erhalten, lassen sich jedoch deutlich effektiver übertragen und, viel wichtiger, wesentlich schneller verarbeiten. Die Kodierung betrifft nur einen speziellen Teil der Serialisierung und Deserialisierung. Für die Anwendung ist die geänderte Kodierung

Abb. 1:
Geschwindigkeitsvergleich
SOAP/
Hessian
(~ Faktor
8)



transparent, sie betrifft nur den XML-Parser bzw. den XML-Serialisierer.

Diese Idee ist nicht neu, sondern wurde bereits vom W3C in der XML Binary Characterization Working Group 2003 [4] thematisiert. Der Auslöser war auch damals die „mangelnde Knappheit“ des XML-Formats (engl. *“Terseness in XML markup is of minimal importance”* [5]) in Verbindung mit großzügigem Verbrauch von Bandbreite, I/O- und CPU-Ressourcen. Einige Formate wie ASN.1 (Abstract Syntax Notification 1) [6] oder MPEG-7 [7] wurden diskutiert, aber es entstand keine W3C Recommendation. Eine Einigung auf eine binäre XML-Alternative scheiterte, weil man das XML-Format nicht in zwei syntaktische Varianten spalten wollte.

Fast Web Services

Einen an obige W3C-Initiative [4] angelehnten Ansatz verfolgte etwas später der Java-Erfinder Sun Microsystems mit seinem Fast-Infoset-Format [8]. Dort werden die Schema- und Instanzdaten in einer neuen Syntax kodiert. ASN.1 übernimmt die Rolle der XML Schemas und Packed Encoding Rules (PER) [9] die Rolle der XML-Instanzdaten.

ASN.1 ist keine Neuerfindung, sondern wurde vor über 15 Jahren definiert. Zusammen mit PER wird es hier geschickt von Sun eingesetzt, um die XML-Semantik kompakter zu kodieren. Dabei bleiben alle Informationen aus SOAP erhalten, d.h., es existiert eine SOAP Envelope und

auch die Java-Typen werden auf XML-Schema-Typen abgebildet.

Sun kündigt hierzu an [10], bei kleineren XML-Dokumenten die Größe des Datenstroms auf bis zu einem Drittel gegenüber SOAP reduzieren zu können. Die Zeit beim Einlesen (Parsen) reduziert sich um den Faktor 3 bis 5, beim Serialisieren um den Faktor 5 bis 10. Dabei bleibt – so die Theorie – die Umgebungs- und Plattformunabhängigkeit von SOAP gewahrt, da ASN.1 und PER keine Java-Abhängigkeiten in ihren Formaten enthalten. In der Praxis gibt es bisher nur für Java eine Web-Service-Implementierung.

Hessian

Ein weiterer Ansatz besteht darin, die Java-Objektinstanzen zusammen mit ihren Typ-Informationen zu serialisieren – vergleichbar mit RMI. Diesen Datenstrom überträgt man dann mit einem internetfähigen Protokoll wie HTTP. Das Verfahren, Java-Objekte zu serialisieren, ist bekannt und einfach zu implementieren. RMI selbst ist wenig geeignet, da es kein HTTP unterstützt und eine stehende Verbindung voraussetzt – im Gegensatz zu Web Services, die zustandslos pro Aufruf eine Verbindung aufbauen. Die Erweiterung RMI over IIOP unterstützt zwar HTTP, wird aber von Sun nur bis Java 1.4. unterstützt.

Die Softwarefirma Caucho Technology [11] nun stellt mit ihrem Web-Service-Kit Hessian eine Open-Source-Bibliothek zur Verfügung. Die Implementierung ist

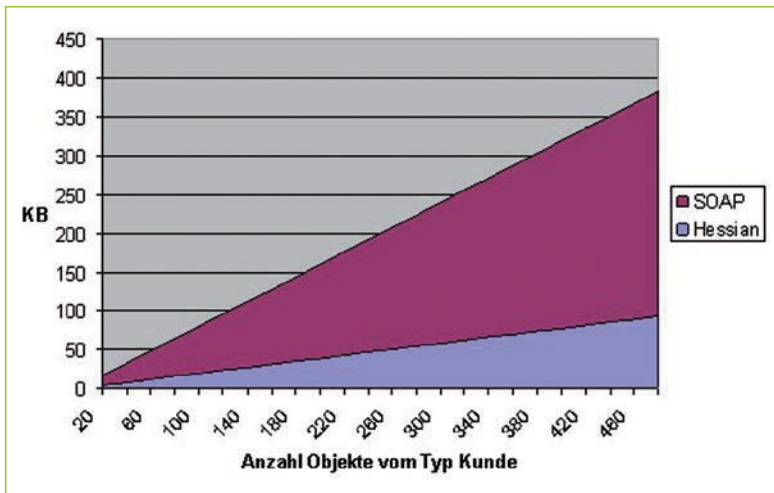


Abb.2: Größenvergleich SOAP/Hessian (~ Faktor 4)

äußerst schlank und erlaubt die Umsetzung von Web Services auf Client- und Serverseite mit geringem Aufwand. Im Unterschied zu RMI benutzt Hessian eigene Serialisierer, sodass auch Objekte, die nicht das Serializable Interface implementieren, einfach übertragen werden können. Zusätzlich werden häufig vorkommende Datentypen wie int, boolean, float, char, byte, String, Date, Calendar, Collections und Arrays optimiert serialisiert. Neben den reinen Nutzdaten müssen, ähnlich wie bei der Java-Serialisierung, auch Metadaten wie Paket-, Klassen- und Feldnamen in den serialisierten Daten gespeichert werden.

Um die reine Antwortzeit von SOAP und Hessian vergleichen zu können, wurde auf einem handelsüblichen PC ein Test für Web Services aufgesetzt. Gemessen wurden die Antwortzeiten abhängig von der Anzahl der zu übertragenden Objekte. Der Einfachheit halber waren Client und Server auf demselben Rechner installiert. Das Ergebnis ist in Abbildung 1 dargestellt.

Der Performancegewinn für Hessian liegt bei einem Faktor 7 bis 9. Die Verzögerung durch das Netzwerk ist vernachlässigbar. Die Zeitunterschiede ergeben sich durch die Unterschiede beim Serialisieren und Deserialisieren der zwei Protokolle. Bei einem Test über ein vergleichsweise langsames Kommunikationsmedium wie ISDN oder gar GPRS wird dieser Geschwindigkeitsvorteil von Hessian nicht mehr entscheidend für die Antwortzeit sein. Dafür zahlt sich dort dann die deutlich geringere Größe der Datenpakete aus (siehe Beispiel im nächsten Kapitel).

SOAP versus Hessian im Beispiel

In Java ist der Aufruf des XML-Parsers besonders ressourcenintensiv. Das wirkt sich auf die Performance einer Applikation aus, wenn diese häufig den Parser aufruft. Nicht nur der Anstieg der CPU-Auslastung ist deutlich messbar, es entstehen auch viele temporäre Java-Objekte, die später wieder vom Garbage Collector entfernt werden müssen. Auch das kostet CPU-Zeit. Bei Benutzung eines binären Formats kann man die Objekte bereits

so ablegen, dass sich später einfach wieder Java-Objekte erzeugen lassen. Dabei entstehen keine temporären Objekte. Die Deserialisierung geht in einem Rutsch und dadurch deutlich schneller vonstatten.

Zur Veranschaulichung der beiden Protokolle folgt nun ein Beispiel. Für das Interface *IKundenSuche* wie folgt

```
public interface IKundenSuche {

    KundenAkte getKundeByName(String kundenName);

}
```

zeigt Listing 1 einen mit Axis erzeugten SOAP Request mit RPC Encoding. Das SOAP Encoding besteht aus 462 Byte für einen Web-Service-Aufruf mit einem 5-Byte-Parameter ("Campo"). Demgegenüber benötigt der Hessian Web Service Request `c**m**getKundeByNameS**Campoz` nur 29 Byte im binären Format von Hessian (*c*, *m* und *S* sind hier Kürzel für call, method und String, *z* ist das Endezeichen, **** steht für eine binäre Längenangabe).

Ein so großer Unterschied zwischen SOAP und Hessian wie in diesem Beispiel mit Faktor 15 besteht allerdings selten. Meist pendelt sich das Verhältnis zwischen Faktor 3 bis Faktor 5 zugunsten von Hessian ein. Es sei noch erwähnt, dass in der SOAP-Welt statt RPC Encoding meist die Typen *wrapped* und *document/literal* verwendet werden (Kasten SOAP). Damit lässt sich die XML-Abbildung der Daten (also in diesem Fall der String-Parameter) anpassen. Hessian zeigt in diesem Beispiel seine Stärke, indem es für häufig auftretende Felder wie Strings optimierte Serialisierungsformate benutzt. So wird *java.lang.String* als *S* mit einer Längenangabe abgekürzt. Bei komplexen Business-Objekten muss Hessian zusätzlich Metadaten wie Klassen- und Feldnamen mit in den Datenstrom aufnehmen.

Die Erfahrung zeigt, dass Hessian Daten deutlich kompakter übertragen kann. Zur Bestätigung wird nun die Web Service Response des obigen Beispiels betrachtet. Hier lag das Größenverhältnis zwischen SOAP (1.300 Zeichen) und

Listing 1

SOAP-Beispiel Request für Kundesuche

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getKundeByName
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <arg0 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        Campo
      </arg0>
    </getKundeByName>
  </soapenv:Body>
</soapenv:Envelope>
```

Hessian (300 Zeichen) bei Faktor 4. Die Abbildung 2 zeigt eine Grafik, die das sich entwickelnde Verhältnis zwischen SOAP und Hessian für eine wachsende Anzahl von Objekten des gleichen Typs darstellt.

Entwicklungsaufwand

Ein wesentlicher Aspekt bei verteilten heterogenen Systemen ist die Plattformunabhängigkeit von SOAP: Ein großer Vorteil bei der Integration einerseits, ein Mehraufwand bei der Implementierung andererseits. Alle Java-Fachobjekte müssen dazu auf neutrale sprachunabhängige Typen abgebildet werden. Dieser Zusatzaufwand entsteht regelmäßig während der Entwicklung und Erweiterung der Anwendung, zahlt sich aber im Smart-Client-Umfeld nach meiner Erfahrung nicht aus. Dort ist das Umfeld homogen, Smart Client und Server werden gemeinsam entwickelt und teilen sich die bei einem Web Service übertragenen Fachobjekte.

Binäre Frameworks wie Hessian zeigen sich da etwas entwicklerfreundlicher, indem sie von sich aus beliebige Java-Typen ohne Definitionen vom Client zum Server und zurück übertragen. Das geht nur so einfach, weil man sich innerhalb der homogenen Java-Objekt-Welt bewegt.

Als einzige Definition genügt auf der Client-Seite die Angabe des Interfaces und

eines URL, während auf der Serverseite Interface und implementierende Klasse ausreichen.

Sind Smart Clients ohne SOAP besser dran?

Bisher wurde nur festgestellt, dass Smart-Client-Anwendungen, die kein fertiges SOAP Backend vorfinden, von einer Library wie Hessian profitieren können. Möglicherweise wird ein Smart Client auch Web Services aufrufen, die in SOAP implementiert sind und entweder von Dritten beigesteuert werden oder nicht in Java implementiert sind. In solchen Fällen ist es richtig, SOAP zu verwenden. Bei existierenden eigenen Java-Web-Services besteht jedoch die Möglichkeit, die Implementierung sowohl mit einer SOAP-Schnittstelle als auch mit einer Hessian-Schnittstelle anzubieten. Beide Typen lassen sich leicht über den URL unterscheiden. Die Definition des Web-Service-Endpunkts findet in der Regel über die Konfiguration statt, ohne dass der eigentliche Programmcode verändert werden muss.

SOAP und Hessian transparent benutzen

Ist die Entscheidung gefallen, SOAP und andere Protokoll gemischt, aber transparent zu verwenden, hilft ein Framework als Wrapper bei der Umsetzung. Im Idealfall ist in diesem eine besondere Unterstützung für Web Services eingebaut, mit der bei der Definition eines Web Service neben dem URL auch das Protokoll festgelegt sowohl SOAP als auch Hessian unterstützt werden. Das spirit Framework von compeople z.B. erzeugt abhängig von der Konfiguration unterschiedliche Proxy-Objekte, die dann entsprechend SOAP- oder Hessian-Aufrufe erzeugen. Ein Beispiel für eine Konfiguration findet sich in Listing 2.

Die Anwendung kann nun über eine Factory den Web Service *spirit.sample.kundensuche* aufrufen, ohne sich um das verwendete Protokoll zu kümmern. Das Protokoll kann damit jederzeit gewechselt werden, indem man den *type*-Parameter verändert. Ein Beispiel für einen trivialen Aufruf einer Factory sehen Sie hier:

```
IKundenSuche kundensuche = (IKundenSuche)Service
FactoryAccessor.getService("spirit.sample.kundensuche");
```

Fazit

Für die Anbindung von Smart Clients an die zentral verfügbaren fachlichen Web Services und die Datenhaltung sind Geschwindigkeit, niedriger Bandbreitenverbrauch und geringe CPU-Belastung vordringliche Ziele, die in SOAP jedoch keine

Hessian zeigt sich im Vergleich zu SOAP entwicklerfreundlicher, da es von sich aus beliebige Java-Typen ohne Definitionen vom Client zum Server und zurück überträgt.

Priorität besitzen. Alternative Protokolle bieten deutlich mehr Performance und sind sehr einfach und für die Anwendung transparent benutzbar. Mit Frameworks kann ein Smart Client SOAP und Hessian gemischt benutzen. Vergleichbar einfach ist es möglich, Web Services auf dem Server je nach Anwendung unterschiedlich anzubieten, Hessian für den Smart Client und SOAP für alle anderen Nutzer des Services.



Christian Campo ist IT-Consultant bei der compeople AG in Frankfurt. Schwerpunktmäßig beschäftigt er sich mit Client-Server-Frameworks, Web Services und verteilten Serveranwendungen auf Basis von Tomcat.

Links & Literatur

- [1] SOAP: www.w3.org/TR/soap
- [2] WSDL: www.w3.org/TR/wsdl
- [3] Notes about itMS-4-all: hcssoftware.sourceforge.net/jason-rohrer/itms4all/
- [4] XML Binary Characterization Working Group: www.w3.org/XML/Binary
- [5] www.w3.org/2003/08/binary-interchange-workshop/Report.html
- [6] Abstract Syntax Notation number One: de.wikipedia.org/wiki/Abstract_Syntax_Notation_One
- [7] MPEG-7: de.wikipedia.org/wiki/MPEG-7
- [8] Fast Infoset and the Pragmatic SOA Approach: java.sun.com/developer/technicalArticles/WebServices/soa2/fastinfoset-soa.html
- [9] Packed Encoding Rules
- [10] Fast Infoset (incl. Performance results): java.sun.com/developer/technicalArticles/xml/fastinfoset/
- [11] www.caucho.com
- [12] www.compeople.de/spirit/

Listing 2

Konfiguration eines Web Service auf dem Client

```
// mit HESSIAN
<webservice
  name="spirit.sample.kundensuche"
  url="http://localhost/sample/hessian/KundenSuche"
  type="HESSIAN"/>

// oder Alternativ

// mit SOAP
<webservice
  name="spirit.sample.kundensuche"
  url="http://localhost/sample/services/
                                KundenSuche"
  type="SOAP"/>
```