einheitliches XML-basiertes Transportverfahren

eXTra und WebServices

Version 1.0
Ausgabestand 1.0.0

Final

Herausgeber:

AWV – Arbeitsgemeinschaft für wirtschaftliche Verwaltung e. V.
Düsseldorfer Str. 40
65760 Eschborn
Vereinsregister 73 VR 5158, Amtsgericht Frankfurt am Main
Telefon: 0 61 96/7 77 26-0
Fax: 0 61 96/7 77 26-51
Mail: info@awv-net.de
Web: www.extra-standard.de, www.awv-net.de.

Das vorliegende Dokument zum einheitlichen XML-basierten Transportverfahren „eXTra“ wurde von Mitarbeiterinnen und Mitarbeitern des AWV-Arbeitskreises 2.1 „Vereinheitlichung von Datenübermittlungssystemen“ im Fachausschuss 2 „Verwaltungsvereinfachung und Entbürokratisierung im personalwirtschaftlichen Umfeld“ erstellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

0. Motivation

Der eXTra-Standard beschreibt Formate zur Datenkommunikation zwischen Maschinen. Eine hervorragende Unterstützung dieser Maschine-zu-Maschine-Kommunikation bietet die Welt der WebServices, die zunehmend in einer serviceorientierten Architektur zum Einsatz kommen. Dieses Dokument beschreibt Nutzungsmöglichkeiten von eXTra-Nachrichten im Kontext von WebServices und gibt Empfehlungen.

1. Grundlagen

1.1. Serviceorientierte Architekturen und der SOAP-Webservices-Stack

Verstärkt wird der Ruf nach der Verwendung von WebServices im eXTra-Umfeld durch die vermehrte Entstehung von **serviceorientierten Architekturen (SOA)** laut:

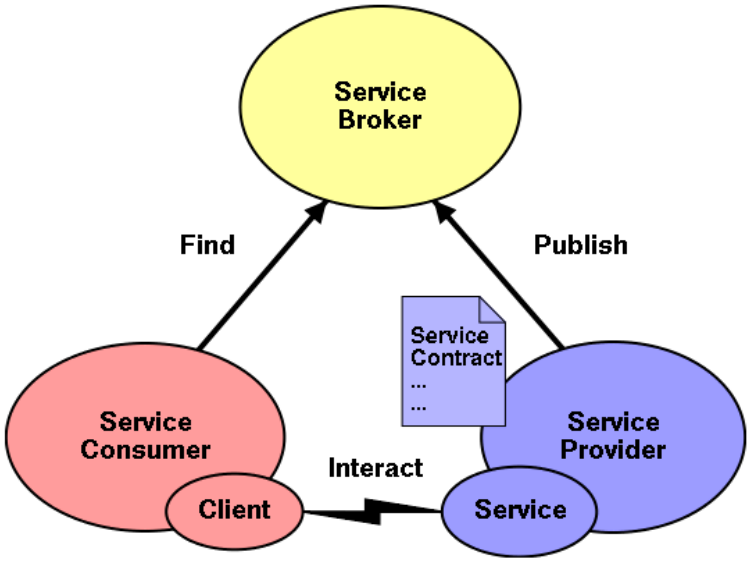

BILD 1: Beteiligte Komponenten und deren Interaktionen in einer serviceorientierten Architektur

Die damit verbundene direktere Unterstützung und Flexiblisierung von Geschäftsprozessen durch Software in Form von lose gekoppelten Services hat sich seit einiger Zeit in Unternehmen, aber auch in der Verwaltung verstärkt. Der **SOAP-Webservices-Stack (SOAP-WS.*)** als **Umsetzungstechnologie für eine SOA** hat sich hier etabliert, was zum Großteil daran liegt, dass sowohl die Open-Source-Community als auch große Softwarefirmen (z. B. Oracle, Microsoft und IBM) massiv in Tools und Frameworks für ihre Laufzeitumgebungen investiert haben.

Auf Basis von SOAP-WS.* ist auf offiziellen Standards (W3C, OASIS u.a) basierende plattformübergreifende, interoperable und sichere Kommunikation und Datenaustausch zwischen verschiedenen Kommunikationpartnern - sowohl unternehmensintern, aber vor allem unternehmensübergreifend - möglich. Voraussetzung ist allerdings, dass sich die Kommunikationspartner in einer SOA (sogenannte Consumer und Provider) bilateral auf die Nutzung bestimmter WS.*-Standards/Features sowie Transportprotokollen, die auch von

ihrer jeweiligen Entwicklungs- und Laufzeit-Infrastruktur unterstützt werden müssen, sowie an die Empfehlungen bzw. Spielregeln der WebServices-Interoperability (WS-I) Organsisation gehalten haben. Diese Spielregeln werden in Form von sogenannte **WS-I-Basic-Profiles** veröffentlicht und können auch durch Tools überprüft werden. Die Basisprofiles fassen ein Subset von ausgewählten WS.*-Standards zusammen.

Infrastruktur-Impact durch Nutzung der SOAP-WS*-Plattform:

Meist wird bei Nutzung der SOAP-WS.* Plattform auf der Providerseite eine **Application-Server-Infrastruktur** benötigt, um die Services den Consumern skalierbar und ausfallsicher anbieten zu können.

Natürlich werden sowohl für die Consumer – als auch die Providerseite leistungsfähige Open-Source-Bibliotheken und Frameworks (inkl. Open-Source-Application-Server) angeboten, so dass hier nicht zwingend größere Investitionen erforderlich sind.

1.2. SOAP

SOAP definiert ein **Kommunikationsprotokoll** auf Basis eines XML-Standards und ist Plattform–, Hersteller- und Sprachenunabhängig. SOAP kann über verschiedene Transportprotokolle wie HTTP, SMTP u.a. übertragen werden.

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

BILD 2: Gerüst einer Soap-Nachricht

SOAP-Messages werden durch sogenannte **SOAP-Nodes** erzeugt, gesendet, verteilt, empfangen und verarbeitet.
Zwischen dem initialen Sender und dem endgültigen Empfänger können in komplexeren Szenarien auch spezielle SOAP-Nodes als **Intermediär** die SOAP-Messages weiterleiten, z.B. aufgrund von Informationen im SOAP-Header.

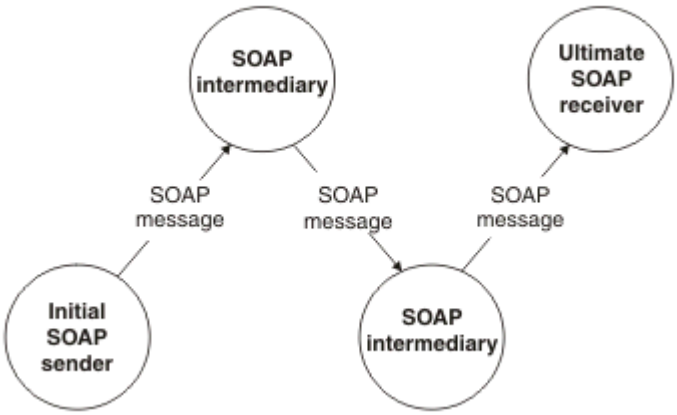

BILD 3: Routing einer Soap-Nachricht über sogenannte Intermediäre

1.3. Serviceinterface (WSDL)

Die Webservices Description Language (WSDL) ist wegen ihres formalen Charakters und der Abstraktionsmöglichkeit bei SOAP-WS.* der zentrale Dreh- und Angelpunkt in der Abstimmung zwischen den Kommunikationspartnern.

Hiermit wird eine weitgehend technologieneutrale Beschreibung des Interfaces eines SOAP-WebServices auf Basis von entsprechenden XML-Schemastandards ermöglicht. Will man vom einzusetzenden Kommunikationsprotokoll und konkreten Servicendpunkten abstrahieren, kann man die WSDL so modularisieren, dass man die Elemente <types>, <messages> und <porttype> in ein eigenständiges, komplett neutrales WSDL-Dokument ausgliedert und dieses in einer spezifischen WSDL mit den restlichen Elementen <binding> und <service> für einen zu erstellenden Webservice importiert.

Überblick WSDL 1.1 Aufbau

	<definitions>
XML-Schema der Datentypen	<types> <schema></schema> </types>
Parameter der Operationen	<message> <part></part> </message>
Service-Schnittstellen	<portType> <operation></operation> </portType>
Kommunikations-Protokoll Mapping: SOAP, HTTP, etc.	<binding> <operation></operation> </binding>
Service-Endpunkte (URL)	<service> <port></port> </service>
	</definitions>

BILD 4: Typischer Aufbau einer WSDL

Bereits die WSDL sollte den WS-I-Vorgaben bzw. dem ausgewählten WS-I-Basicprofile genügen, um die Interoperabilität zur Laufzeit nicht zu gefährden, da aus der WSDL der Consumer- und Providercode generiert wird.

1.4. Servicecontract (Policy)

Für jeden SOAP-WebService ist neben dem Serviceinterface (WSDL) ein Servicecontract notwendig. Der Servicecontract stellt eine informelle Spezifikation des Zwecks, der Funktionalität, der Beschränkungen sowie der Nutzung des Services bereit und geht damit über das Serviceinterface hinaus. Der Servicecontract sollte dabei sowohl detaillierte Informationen bzgl. der Funktionalität bzw. der Leistungen des Services bereitstellen als auch Randbedingungen und Parameter, die nicht Teil der WSDL sind, beschreiben.

Im SOAP-WS.*-Umfeld gibt es als Teil des Servicecontracts den **WS-Policy-Standard**, der dazu dient, ergänzende Vereinbarungen zum Interface mit WSDL formalisiert abzubilden. Schwerpunkt dieser Policies sind Quality of Service- (QoS) - Anforderungen wie Security, Transactions, Reliable-Messaging u.a. in Form von Zusicherungen (Assertions). Auch eigendefinierte Custom-Policies sind im Rahmen dieses Standards abbildbar.

Über den speziellen **WS-Security Policy Standard** können spezielle Security-Policies definiert werden, die natürlich von der verfügbaren Infrastruktur auf Consumer und Providerseite entsprechend unterstützt werden müssen.

1.5. WS.*-Universum

Neben SOAP und WSDL als XML-Standards für WebServices stehen weitere Standards, die in Summe den gesamten SOAP-WebServices-Stack ausmachen. Natürlich müssen nicht alle dieser Standards in konkreten Kommunikationsszenarien genutzt werden.

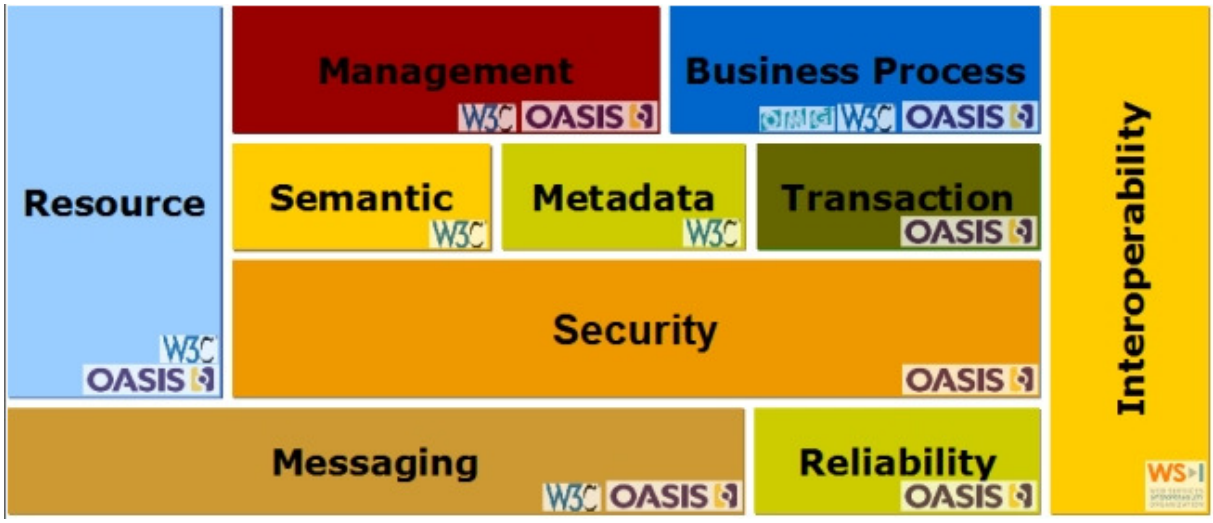

BILD 5: Verschiedene Bereiche des WS*-Universums

1.6. Message Exchange Patterns

SOAP kennt bei der Kommunikation unterschiedliche Message Exchange Patterns (MEP), deren Auswahl sowohl beim Design der WSDL als auch bei der Auswahl der zu nutzenden Transportprotokolle zwischen Consumer und Provider über entsprechende SOAP-Bindings zu berücksichtigen ist.
Zu unterscheiden sind bei den MEP grundlegend der fachliche und der technische Aspekt des Kommunikationsprotokolls.

Im Rahmen einer Webservice-Kommunikation mit SOAP ist aus technischer Sicht zwischen Consumer und Provider zu klären, ob

a) ein Service-Call dazu führt, dass der Consumer beim Aufruf blockiert wird bis eine Antwort vom Provider kommt, oder
b) der Service-Consumer sofort nach Absetzen des Requests ohne Blockierung, jedoch auch ohne Server-Antwort, weiterarbeiten kann

Message Exchange-Pattern "In-Only" (One-Way)

Bei diesem sehr einfachen Message-Exchange-Pattern werden die Nachrichten nur in eine Richtung versendet. Der Sender kümmert sich nicht darum, ob die Nachricht tatsächlich angekommen und korrekt verarbeitet wurde.

Die Fire-and-Forget-Mimik führt dazu, dass der Absender nach Wegschicken der Nachricht nicht auf eine Response warten muss.

BILD 6: Darstellung "In-Only"

Eine Spielart von In-Only ist das „Robust In-Only", bei dem der Erhalt der Nachricht vom Service-Provider quittiert wird (entweder mit einem positiven Status-Code oder mit einer Fehlernachricht).

Message Exchange-Pattern "In-Out"

Das üblichste und mittels HTTP sehr einfach abbildbare Kommunikations-Verfahren ist das Request-Response-Pattern (im SOAP-Umfeld als „In-Out" bezeichnet).

Ein Service-Consumer stellt einen Request an einen Server, dieser antwortet mit einer Response (oder ggf. mit einer Fehlermeldung).

Die Kommunikation verläuft synchron, der Consumer wartet auf die Antwort des Service-Providers.

Das Transportprotokoll bietet eine implizite Assoziierung zwischen Request und Response.

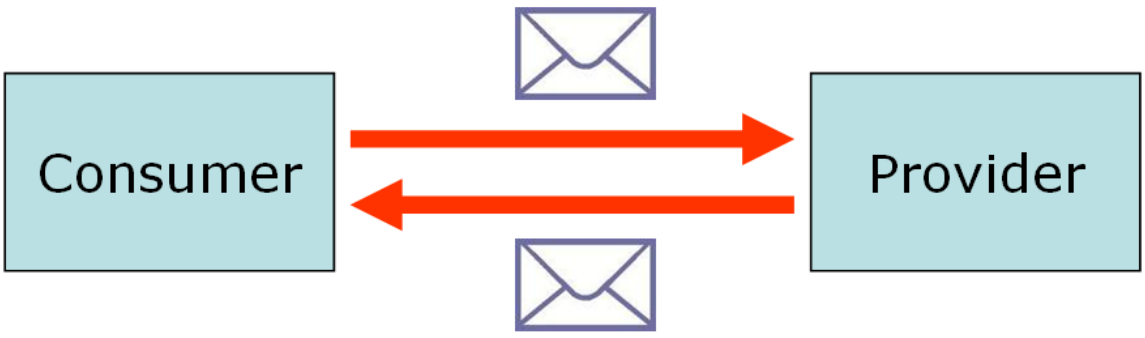

BILD 7: Darstellung "In-Out"

Asynchrone Message Exchange-Patterns

Bei der technisch asynchronen Kommunikation wird ein Webservice-Call abgesetzt, dessen serverseitige Abarbeitung relativ lange dauern kann. Ein Blockieren des Clients soll hierbei vermieden werden, weshalb der Consumer nach Absetzen des Requests lediglich eine Statusmeldung (zumeist mit einer Correlation-ID) erhält.

Der Service-Provider arbeitet den Request – ggf. über einen längeren Zeitraum – ab. Das Ergebnis wird dann in einem neuen, dedizierten Webservice-Call an den ursprünglichen Consumer zurückgegeben:

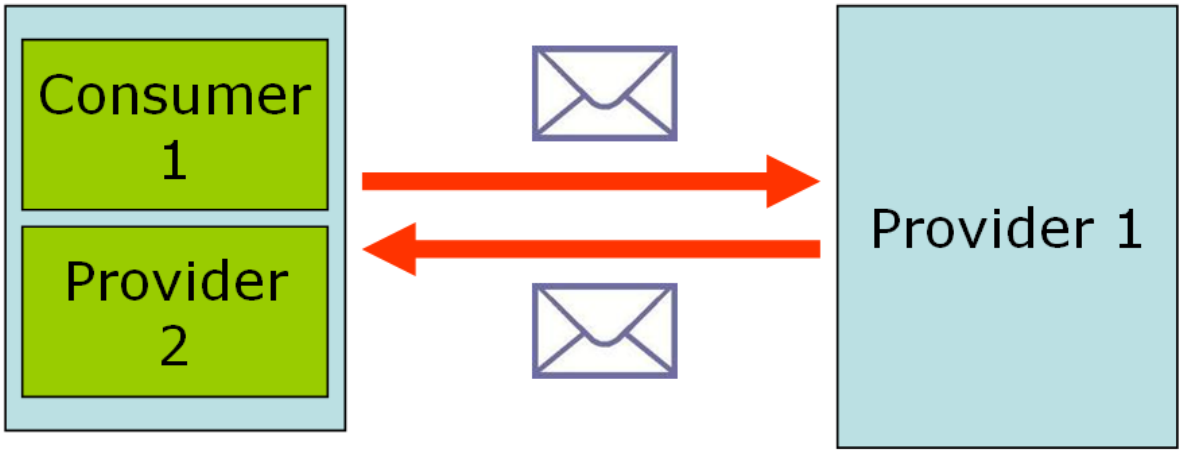

BILD 8: Darstellung eines asynchronen Kommunikations-Szenarios

In diesem Szenario sind also technisch zwei Webservices involviert, die technisch unabhängig voneinander kommunizieren – dies macht einen Mechanismus zum Zuordnen der richtigen Nachrichten notwendig. Ferner ist die Aufrufbarkeit eines Services beim Endanwender, z.B. aufgrund der noch nicht reifen Infrastruktur, derzeit nicht angeraten.

Fachlich asynchrone Aufrufe können natürlich durch technisch synchrone Aufrufe abgebildet werden (z.B. auch mittels Polling).

2. eXTra und die SOAP-WS*-Plattform

eXTra als einheitliches XML-basiertes Transportverfahren kann - rein technisch betrachtet - im Rahmen einer WebService-Kommunikation mit SOAP zum Einsatz kommen bzw. in eine SOAP-Kommunikation integriert werden. Letztlich werden bei der WebService-Kommunikation mit SOAP auch „nur“ XML-Nachrichten und etwaige binäre Anhänge, im Falle von MTOM in einem Standard-Format, über ein Transportprotokoll (wie z.B. HTTP) versendet.

BILD 9: Einbettung von eXTra-Nachrichten in eine SOAP-Nachricht

Vom **Designaspekt** her ist in jedem Falle bei der Integration von eXTra in einen Service der WSDL-First-Ansatz gesetzt. Der WSDL-First-Ansatz ist ein Top-Down-Ansatz, der im Webservices-Umfeld auch als Contract-First bezeichnet wird. Beim **WSDL-First-Ansatz** wird die WSDL i. d. R. nicht durch Tools - ausgehend von bestehendem Sourcecode - generiert, sondern manuell erstellt. Dabei werden die XML-Schemata von eXTra für die auszutauschenden Datenstrukturen entsprechend inkludiert. Aufgrund der vorhandenen eXTra Nachrichten-Strukturen (ebenso nach Profilierung) kommt somit ausschließlich der **Document-Literal-Message-Style** in Frage.

eXTra kennt drei eXTra Ebenen: Fach-Nachrichten, Paket- und Transportebene. Vom **benötigten eXTra Datenumfang** her gesehen kann die WebService-Schnittstelle eine weite Spanne vom Transport einer gesamten eXTra-Nachricht bis hin zu einem speziellen eXTra-

WebService-Schema (z.B. nur Nachrichten auf Basis des Transport-Request-/Reponse-Types) abdecken – dies obliegt dem Designer der WebService-Schnittstelle, der diese im Kontext eines Fachverfahrens für die gesamte eXTra-Topologie entwickelt.

Je nach eingesetzten Laufzeitumgebungen besteht die Möglichkeit, dass die eigentlichen Nutzdaten der eXTra-Nachricht mit dem effizienten Verfahren „**MTOM - Übermittlung von Binärdaten“** übertragen werden. Dadurch erreicht man, dass eine Übertragung ohne (die durch die Base64-Codierung bedingte) nennenswerte Vergrößerung des Datenvolumens erfolgen kann.

Zu beachtende Abgrenzungen und Verträglichkeitsgesichtspunkte beim Zusammenspiel von eXTra und SOAP-WS.*

Hier gibt es folgende Themen:

- **Message-Exchange-Patterns**
  Technisch gesehen wird das Request/Response-Pattern im Zusammenspiel mit eXTra wohl das häufigst genutzte sein. Die passende technische Abbildung der SOAP-MEPs auf passende eXTra – MEPs ist zu beachten. Es könnte z.B. ein In-Only auf SOAP-WS.* Basis kein Request/Response auf eXTra-Ebene abbilden. Die Verträglichkeit von SOAP und eXTra ist der folgenden Tabelle zu entnehmen (WSDL 2.0. wurde gegenüber 1.1 weiterentwickelt, wobei die Namensgebung der MEPs geändert wurde):

MEP		eXTra		
WSDL 1.1	WSDL 2.0	Fire-and-forget	Request-with-Acknowledgement	Request-with-Response
One-Way	In-Only	**X**		
	Robust In-Only			
Request-Response	In-Out		**X**	**X**
	In-Optional-Out			

- **Security**
  Bei Nutzung von **WS-Security-Standards** sind ähnlich wie bei eXTra Signaturen / Verschlüsselungen für Elemente des SOAP-Headers und des Payloads im SOAP-Body möglich sowie unterschiedliche Authentifizierungverfahren (WS-Trust/WS-Federation etc) auf Basis von XML-Signature und XML-Encrpytion. Diese könnten alternativ zu den Möglichkeiten des eXTra-Standards auf der Transport-Ebene bei der Umsetzung in ein lauffähiges Datenübermittlungssystem auf der DFÜ-Ebene

verwendet werden (siehe Dokument „Sicherheit und Verfügbarkeit in einem eXTra-spezifischen Datenübermittlungsverbund“).

- **Fehlerbehandlung**

  SOAP sieht eine spezielle Fehlerbehandlung durch **SOAP-Faults** vor, die schon beim Design der WSDL an entsprechenden Stellen zu berücksichtigen ist. Unterschieden werden müssen dabei sowohl SOAP-Faults mit Standardfehlern, die automatisch von der Laufzeitumgebung/SOAP-Engine am Server erzeugt und dem Consumer immer gesendet werden, als auch explizit **modellierte SOAP-Faults**, die entsprechend in die Datenstrukturen der WSDL integriert werden können. Sollte dieser Mechanismus bei eXTra genutzt werden, würden sich hierfür die vorhandenen Fehlerdatenstrukturen aus dem Schema **extra-error-*.xsd** anbieten. Auf welche Punkte hierbei zu achten ist, wird im Anhang A1 „SOAP-Faults“ dargestellt. Wie eine WSDL mit SOAP-Fault aussehen kann, ist im Anhang A2 „Beispiel einer WSDL-Datei eines eXTra-WebService“ dargestellt.

- **Service-/Message-Versionierung**

  Üblicherweise wird in serviceorientierten Umgebungen auf Basis von SOAP-WS.* die WSDL inklusive der integrierten XML-Schemata für die Message-/Datenstrukturen auf XML-Namespaceebene versioniert. In eXTra gibt es bereits eine Versionskennung in den entsprechenden Schemata. Die Versionierung bzw. das komplette Lifecyclemanagement / Governance eines SOAP-WebServices ist aber unabhängig von der Versionierung in den eXTra-Schemata zu sehen.

- **Einsatz eines Enterprise Service-Busses (ESB)**

  In serviceorientierten Umgebungen mit vielen Services und Consumern kommt sehr oft ein ESB als zentrale Governance-Infrastruktur zum Einsatz. Dieser kann verschiedenste Aufgaben wie Routing, Tranformation von Nachrichten, Umsetzung von Kommunikationsprotokollen und MEPs sowie aspektorientierte Connectivy (Logging, Security, etc.) übernehmen. Letztlich ist zu berücksichtigen, dass bei Einsatz eines ESB Consumer und Provider eines eXTra basierten SOAP-WebServices entkoppelt und über den ESB miteinander verbunden werden. Inbesondere die Themen Security und MEPs sind auch hier im Kontext von eXTra hinsichtlich Abgrenzung und Verträglichkeit zu berücksichtigen (siehe auch Dokument „Sicherheit und Verfügbarkeit in einem eXTra-spezifischen Datenübermittlungsverbund“).

3. Konformität

Konformität ist sowohl bei SOAP WebServices als auch bei eXTra ein sehr wichtiges Thema.

Bei SOAP WebServices wird eine Realisierung empfohlen, die WS-I konform ist und mit Hilfe der veröffentlichten WS-I-Basic-Profiles überprüft werden kann.

Bei eXTra sind folgende Grundsätze zu beachten:

- Für die eXTra-Konformität eines SOAP-WebServices ist die Grundvoraussetzung, dass es keine Einschränkung der eXTra-Features geben darf. Die Spezifikation des SOAP WebServices darf also nicht so restriktiv sein, dass eXTra-Sprachmittel nicht verwendet werden können.
- Die eXTra-Namespaces dürfen nicht verändert werden.
- Die originalen bzw. profilierten eXTra Schemadateien dürfen nicht verändert werden.

4. Effizienz – MTOM

MTOM ist völlig transparent für die eXTra-Schemata. Alle eXTra-Schema-Elemente, die auf der **Base64CharSequence bzw. Data** (siehe unten) basieren, können potentiell per Attachment per Referenz außerhalb des SOAP-Bodys übertragen werden, sofern beide Kommunikationspartner diese Optimierung in ihrer SOAP-WebServices-Runtime unterstützten.

• Aktivierung von MTOM

Die Aktivierung von MTOM findet aus Sicht des WebServices auf Service- oder Operation-Ebene statt und betrifft die Message-Parameter der Operation des Porttypes (WSDL):

```
<xs:element name="Data" type="DataType"/>
   <xs:complexType name="DataType">
      <xs:choice>
         <xs:element ref="CharSequence" minOccurs="1" maxOccurs="1"/>
         <xs:element ref="Base64CharSequence" minOccurs="1" maxOccurs="1"/>
         <xs:element ref="ElementSequence" minOccurs="1" maxOccurs="1"/>
         <xs:element ref="AnyXML" minOccurs="1" maxOccurs="1"/>
      </xs:choice>
   </xs:complexType>

<xs:element name="Base64CharSequence" type="Base64CharSequenceType"/>
   <xs:complexType name="Base64CharSequenceType">
      <xs:simpleContent>
         <xs:extension base="xs:base64Binary"/>
      </xs:simpleContent>
   </xs:complexType>
```

Hinweis: Da die Element Data und Base64CharSequence genau in dieser Form im eXTra Standard definiert sind, werden diese Definitionen durch den Import der eXTra Namespaces in der WSDL verfügbar. Die explizite Definition in der WSDL kann also im Zusammenhang mit dem eXTra Standard entfallen.

• Integration der speziellen MTOM-WS-Policy

Weitere Auswirkungen auf die WSDL bzw. Schnittstelle sind jedoch gegeben. Siehe dazu http://www.w3.org/Submission/WS-MTOMPolicy/.

Diese spezielle MTOM-WS-Policy muss entweder direkt oder per Policyattachment in die WSDL integriert werden. Dabei ist folgendes zu beachten:

Integration der erforderlichen Namespaces

```
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"

xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
```

Einfügen der eigenen Policy

```
<wsp:Policy wsu:Id="eXTraMTOMPolicy">
     <wsoma:OptimizedMimeSerialization />
</wsp:Policy>
```

Einfügen der Referenz im Binding

```
<wsdl:binding name="extraSOAP" type="extraws:extra">
     <wsp:PolicyReference URI="#eXTraMTOMPolicy" wsdl:required="true" />
```

Hinweis: in reinen Java-Java-Umgebungen oder auch in reinen .NET-.NET Umgebungen ist dies obsolet. Im Allgemeinen, bzw. in anderen Umgebungen ist auf diese Auswirkungen jedoch zu achten.

5. Empfehlungen für eXTra Implementierungen

5.1. Empfehlungen für bestehende eXTra Implementierungen

Wenn es bereits ein eXTra-spezifisches Datenübermittlungssystem z.B. auf Basis von https gibt und ein zusätzlicher Transport-Eingang/Kanal in Form eines SOAP WebServices geschaffen werden soll, dann ist eine naheliegende Möglichkeit, den bestehenden eXTra-Server unverändert zu lassen und die über den neuen, zusätzlichen Transport-Eingang vom SOAP WebService empfangenen eXTra-Dokumente an den eXTra-Server zur weiteren Bearbeitung weiter zu reichen.

Ein puristischer Lösungsansatz, bei dem eXTra in der WSDL nicht in Erscheinung tritt, ist so gestaltet, dass ein eXTra-Dokument/Nachricht im SOAP-Body als Attachment in einem base64binary Element übermittelt wird. Die SOAP-Engine auf Empfängerseite leitet dann das empfangene Dokument wie skizziert an den nachgelagerten eXTra-Server zur weiteren Verarbeitung weiter. Diese sehr einfache und schnelle Lösung kann jedoch nicht davon profitieren, dass auf der SOAP-Ebene das eXTra Dokument auf syntaktische Korrektheit mittels XML-Parser geprüft werden könnte, dies muss weiterhin der nachgelagerte eXTra-Server übernehmen.

Will man jedoch vom XML-Parser der SOAP-Engine profitieren und somit erreichen, dass der nachgelagerte eXTra-Server nur syntaktisch korrekte eXTra-Dokumente erhält, muss man die eXTra Strukturen und Namespaces in die WSDL importieren. Ein Beispiel mit zugehöriger WSDL ist im Anhang A2 „Beispiel für eine WSDL eines eXTra WebServices mit SOAP-Faults und MTOM“ zu finden.

5.2. Empfehlungen für neue Implementierungen auf Basis von eXTra

Hier ist man freier, was die Nutzung der Features einer SOAP-Engine oder eines ESB angeht.

Die Gestaltungsmöglichkeiten mittels WSDL ermöglichen prinzipiell eine weit feinere Definition der eXTra-Schnittstelle mit WSDL Sprachmitteln. Es wird jedoch empfohlen, darauf aus mehreren Gründen zu verzichten. Es gilt nicht nur, eine möglichst große Übereinstimmung mit den eXTra-XML-Nachrichten beizubehalten, sondern darüber hinaus eine möglichst große Entkopplung von SOAP und eXTra bzw. eine minimale gegenseitige Beeinflussung zu erreichen, weil nur so eine weitgehend konfliktfreie Weiterentwicklung beider Standards sichergestellt werden kann.

6. Referenzen

6.1. Referenzen zu SOA und dem SOAP-WebServices-Stack

[1] SOAP http://www.w3schools.com/soap/default.asp
[2] WSDL http://www.w3schools.com/wsdl/default.asp
[3] WS-I http://www.ws-i.org/
[4] WS*-Universum http://www.ws-overview.com/
[5] MTOM
http://de.wikipedia.org/wiki/SOAP_Message_Transmission_Optimization_Mechanism
[6] SOA http://www.itwissen.info/definition/lexikon/service-oriented-architecture-SOA-SOA-Architektur.html

6.2. Referenzen zur Dokumentation des eXTra Standards

Kurzname	Quelle
DSIG	*eXTra Design Guidelines*, zu finden unter www.extra-standard.de
EINF	*Einführung in den eXTra Standard,* zu finden unter www.extra-standard.de
EMSG	*eXTra Standardnachrichten, Schnittstellenbeschreibung,* zu finden unter www.extra-standard.de
EXSEC	*Sicherheit und Verfügbarkeit in einem eXTra spezifischen Datenübermittlungsverbund,* zu finden unter www.extra-standard.de
EXWS	*eXTra und Webservices,* zu finden unter www.extra-standard.de
IFACE	*eXTra Transport Schnittstellenbeschreibung,* zu finden unter www.extra-standard.de
IMPL	*eXTra Implementierung*
KOMP	*eXTra Kompendium,* zu finden unter www.extra-standard.de
RFC2119	*Request for Comments: 2119*, S. Bradner, Harvard University, March 1997, http://www.ietf.org/rfc/rfc2119.txt
PROF	*eXTra Profilierung,* zu finden unter www.extra-standard.de
XENC	*XML Encryption*, http://www.w3.org/TR/xmlenc-core/
XML	*XML Recommendation 1.0, 3rd Edition*, http://www.w3.org/XML
XSD	*XML Schema Definition*, http://www.w3.org/TR/xmlschema-0/
XSIG	*XML Signature*, http://www.w3.org/TR/xmldsig-core/
XSL	*XML Stylesheet Language*, http://www.w3.org/TR/1999/REC-xslt-19991116, http://www.w3.org/TR/xslt20/

Anhang A1 „SOAP-Faults“

Entscheidend bei der Integration in eine WSDL sind folgende Aspekte:

• Integration des Schemas und des Namepace

```
  xmlns:extraerror="http://www.extra-standard.de/namespace/service/1"
...
  <xsd:import namespace="http://www.extra-standard.de/namespace/service/1"
schemaLocation="eXTra-error-1.xsd" />
```

• Definition einer passenden Message

```
  <wsdl:message name="ExtraFault">
    <wsdl:part element="extraerror:ExtraError" name="fault" />
  </wsdl:message>
```

• Aufnahme des Faults bei der Operation sowohl beim Porttype als auch beim zugehörigen Binding

```
  <wsdl:portType name="extra">
    <wsdl:operation name="execute">
      <wsdl:input message="extraws:executeRequest" />
      <wsdl:output message="extraws:executeResponse" />
      <wsdl:fault name="fault" message="extraws:ExtraFault"></wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>

...
  <wsdl:binding name="extraSOAP" type="extraws:extra">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="execute">
      <soap:operation
        soapAction="http://www.extra-standard.de/namespace/webservice/execute" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="ExtraFault">
        <soap:fault name="ExtraFault" use="literal" />
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
```

Anhang A2 „Beispiel einer WSDL-Datei eines eXTra-WebService mit SOAP-Faults und MTOM“

• Anmerkungen
Die WSDL wurde gemäß den Empfehlungen in Kapitel 5.2 erstellt, um – wie erwähnt - eine möglichst große Entkopplung von SOAP und eXTra bzw. eine minimale gegenseitige Beeinflussung zu erreichen, weil nur so perspektivisch eine weitgehend konfliktfreie Weiterentwicklung beider Standards sichergestellt werden kann.

• Erläuterungen
Wie oben im Kapitel 2 erwähnt, wurde die WSDL-Datei gemäß WSDL-First-Ansatz nicht mit Tools, sondern händisch erstellt.

Für das Beispiel wurde SOAP 1.1 verwendet, da dieser am weitesten verbreitet ist. Zu erkennen ist die Version am Parameter des Elements soap:binding.

Als Style für In- und Output wurde „literal“ verwendet (siehe auch Kapitel 6 oben). Dieses wird ebenfalls häufig verwendet und vereinfacht so die Umsetzung.
Der Name der Operation wurde mit „execute“ angegeben, da es in diesem Bereich keine weiteren Vorgaben gibt. Der Name „executeResponse“ wurde beim generieren der WSDL automatisch vergeben.
Als In- und Out-Parameter wurde der TransportRequestType bzw. TransportResponseType aus dem eXTra-Standard-Schema 1.2 gewählt. Der eXTra-Standard Version 1.2 wurde deshalb genommen, weil erst mit dieser Version die Fehlernachricht ExtraError zur Verfügung steht.

Die abgebildete WSDL-Datei und die dazugehörigen eXTra-Schemadateien sind WS-I BP 1.1 konform. Die Interoperabilität sollte somit gewahrt sein.
Eine Unterstützung und Konformität nach WS-I BP 1.2 oder auch 2.0 hängt davon ab, ob der Dienstanbieter diese Art der Schnittstellenbeschreibung bereitstellt.
Je nach eingesetzten Laufzeitumgebungen besteht die Möglichkeit, dass die eigentlichen Nutzdaten der eXTra-Nachricht mit MTOM, dem effizienten Verfahren zur Übermittlung von Binärdaten, eingesetzt werden kann. Dadurch erreicht man, dass eine Übertragung ohne nennenswerte Vergrößerung des Datenvolumens umgesetzt werden kann (siehe http://de.wikipedia.org/wiki/MTOM).

• Aufbau der WSDL-Datei
In der WSDL-Datei ist eine Operation „execute“ definiert, welche die eXTra-Nachrichten entgegen nimmt. Als Parameter wird eine Message vom Typ TransportRequestType verwendet. Die Antwort der Operation ist vom Typ TransportResponseType. Im Gegensatz zu einer „normalen“ eXTra-Nachricht fehlt also der übergeordnete Type XMLTransport.

Wie in Anhang 1 dargestellt, wurden für dieses Beispiel SOAP-Faults auf der Basis der eXTra Nachricht ExtraError integriert.

Ebenso wurde MTOM wie oben erläutert eingefügt.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="extra"
  targetNamespace="http://www.extra-standard.de/namespace/webservice"
```

```
  xmlns:extraws="http://www.extra-standard.de/namespace/webservice"
  xmlns:extraerror="http://www.extra-standard.de/namespace/service/1"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/
optimizedmimeserialization"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsp:Policy wsu:Id="eXTraMTOMPolicy">
    <wsoma:OptimizedMimeSerialization />
  </wsp:Policy>

  <wsdl:types>
    <xsd:schema targetNamespace="http://www.extra-standard.de/namespace/webservice"
      xmlns:Q1="http://www.extra-standard.de/namespace/request/1"
      xmlns:Q2="http://www.extra-standard.de/namespace/response/1">
      <xsd:import namespace="http://www.extra-standard.de/namespace/request/1"
        schemaLocation="eXTra-request-1.xsd" />
      <xsd:import namespace="http://www.extra-standard.de/namespace/response/1"
        schemaLocation="eXTra-response-1.xsd" />
      <xsd:import namespace="http://www.extra-standard.de/namespace/codelists/1"
        schemaLocation="eXTra-codelists-1.xsd" />
      <xsd:import namespace="http://www.extra-standard.de/namespace/components/1"
        schemaLocation="eXTra-components-1.xsd" />
      <xsd:import namespace="http://www.extra-standard.de/namespace/logging/1"
        schemaLocation="eXTra-logging-1.xsd" />
      <xsd:import namespace="http://www.extra-standard.de/namespace/service/1"
        schemaLocation="eXTra-error-1.xsd" />

      <xsd:element name="execute">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="Q1:TransportRequestType" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="executeResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="out" type="Q2:TransportResponseType" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

    </xsd:schema>
  </wsdl:types>



  <wsdl:message name="executeRequest">
    <wsdl:part element="extraws:execute" name="parameters" />
  </wsdl:message>
  <wsdl:message name="executeResponse">
    <wsdl:part element="extraws:executeResponse" name="parameters" />
  </wsdl:message>
  <wsdl:message name="ExtraFault">
    <wsdl:part element="extraerror:ExtraError" name="fault" />
  </wsdl:message>

  <wsdl:portType name="extra">
    <wsdl:operation name="execute">
      <wsdl:input message="extraws:executeRequest" />
      <wsdl:output message="extraws:executeResponse" />
      <wsdl:fault name="fault" message="extraws:ExtraFault" />
```

```
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="extraSOAP" type="extraws:extra">

    <wsp:PolicyReference URI="#eXTraMTOMPolicy" wsdl:required="true" />

    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />

    <wsdl:operation name="execute">
      <soap:operation
        soapAction="http://www.extra-standard.de/namespace/webservice/execute" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="ExtraFault">
        <soap:fault name="ExtraFault" use="literal" />
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="extra">
    <wsdl:port binding="extraws:extraSOAP" name="extraSOAP">
      <soap:address location="http://localhost/eXTra-WS-MTOM/extra" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```