FH Wedel

Seminararbeit

in der Fachrichtung Wirtschaftsinformatik

im Rahmen des Informatik-Seminars WS2006/2007

mit dem Thema

Service-orientierte Architektur (SOA)

bei

Prof. Dr. Sebastian Iwanowski

Thema:

WS-Policy, WS-Addressing, WS-ReliableMessaging: Zielsetzungen, Zusammenhang, Lösungen und Beispiele

Bearbeitet von: Moritz Steinbicker

Matr.-Nr.: WI 5445

Inhaltsverzeichnis

1. Einführung- WS-* Spezifikationen

Die 1. Generation der Web Services stellen die Basiskomponenten einer XML basierten SOA dar. Sie besteht aus den XML Spezifikationen WSDL, SOAP und UDDI, die allerdings nur ein rudimentäres Konzept darstellen. Microsoft, IBM und andere große IT-Unternehmen haben weitere Spezifikationen entwickelt, die die Basiskomponenten erweitern – die *2. Generation Specifications* oder auch *WS-*. Sie sollen den Anforderungen einer vollwertigen SOA entsprechen und dabei Web Services sicherer und zuverlässiger machen und sie befähigen Transaktionen besser zu unterstützen. Sie sind nach dem Design-Prinzipien der Komponierbarkeit erstellt wurden: Jede Spezifikation erfüllt einen unmittelbaren Funktionalität. Sie können unabhängig voneinander benutzt werden und nach jeweiligen Bedürfnissen und Anforderungen miteinander komponiert werden. WSDL in Kombination mit WS-Policy unterstützen diese Komponierbarkeit und ermöglichen eine Beschreibung der verwendeten bzw. geforderten WS-Spezifikationen. SOAP unterstützt sie durch ihren optionalen Header-Block, der es ermöglicht einer bestehenden Nachrichten konfliktfrei weitere Funktionalitäten hinzuzufügen. Im folgenden werde Ich die WS-* Spezifikationen WS-Addressing, WS-ReliableMessaging und WS-Policy vorstellen.

2. WS-Addressing

2.1 Problemstellung und Zielsetzung

Beim SOAP Protokoll sind die wesentlichen Informationen zur Adressierung, insbesondere die Nachrichten-Identität und das Routing, in der darunter liegenden Transportschichten, meistens HTTP, angesiedelt. Diese Abhängigkeit führt zu Problemen, die ohne WS-Addressing nur durch den jeweiligen Entwickler aufwendig und von Fall zu Fall in den jeweiligen Anwendung lösen lassen. Folgende Problemstellung ergeben sich, die durch das WS-Addressing Protokoll gelöst werden:

1. Der Zugang zu einen Service ist nicht beschränkt auf ein Transportprotokoll. Neben HTTP / HTTPS ist vor allem SMPT von großer Bedeutung. Es ist zu dem vorstellbar, dass bei einer Interaktion mit einen Service verschiedene Protokolle verwendet werden, z.B. eine Bestellung über HTTP und eine Bestätigung per Mail erfolgt.
2. Sevice Interaktionen sind nicht nur auf das das einfache, synchrone Request-Response-Muster beschränkt. Asynchrone Nachrichten-Muster sind immens wichtig, lassen sich aber nicht mit HTTP realisieren,da hier die Antwort einer Anfrage, immer zum Absender zurückgeht.

3. Geschäftsprozesse, die mit Web Sevices modellieren werden, laufen gewöhnlich lange und sind zustandsabhängig, was zu einen Nachrichten-austausch über einen langen, nicht genau absehbaren Zeitraum führt. Eine permanente Verbindung währenddessen ist nicht zumutbar.
4. Da die Quelle und das Ziel nicht Teil der Nachricht sind, können diese Informationen beim Transport verloren gehen, etwa durch einen Vermittler, z.B. Firewall, oder durch das Unterbrechen der Verbindung.

2.2 Aufbau und Architektur von WS-Addressing

Für die Adressierung einer Nachricht benötigt man, wie bei der Versendung eines Briefes, folgende Informationen: einen Absender, z.B. der Service Requestor, eine Zieladresse, z.B. der Endpoint des Service Providers, ggf. eine spezielle Instanz der Ziel-Adresse, die den Service Provider ermöglicht, Nachrichten einer verarbeitenden Instanz schneller zuzuordnen, und ggf. eine Adresse für Fehlermeldungen. WS-Addressing führt zwei Grundlegende Konzepte ein, die diese Informationen bereitstellen:

- Endpoint Reference, eine neu eingeführte Datenstruktur, die alle Notwendigen Daten kapselt, die benötigt werden, um einen Service Enpoint zur Laufzeit zu erreichen.
- Message Information Headers, die die Enpoint References benutzen, um einen Kontext für den Nachrichtenaustausch herzustellen. Beim Brief wäre dies z.B. der Absender und der Empfängeradresse.

2.2.1 Endpoint References (EPR)

Ein Endpoint ist jede Quelle oder Ziel einer Web Services Nachricht. Eine Endpoint Reference ist ein XML Dokument, das die notwendigen Informationen für die Verwendung eines Endpoint bereitstellt und somit eine standardisierte Darstellung von Service Endpoints einführt. Gegenüber der statischen WSDL Beschreibung ist es nun möglich, den Laufzeitcharakter eines Endpoints zu erfassen, was für folgende Fälle notwendig ist:

- Fehlende WSDL-Informationen: Endpoints können dynamisch erstellt woder angepasst werden,durch Updates oder zusätzliche Policy-Informationen etwa, so dass es keine WSDL-Beschreibung gibt.
- Mehrere Endpoints teilen sich den gleichen WSDL Port: Services die mehrfache zustandsorientierte Interaktionen unterstützen, also mehrere Instanzen für einen Service haben, werden nicht durch WSDL beschrieben, da sie zustandsabhängig sind. Sie müssen aber identifiziert werden oder konfiguriert werden können.

Hier eine Beispiel EPR[1] eines Endpoints eines Web Services, der die Abrechnung von Versicherungsansprüche von Patienten an eine Krankenkasse erfasst:

```
(01)<wsa:EndpointReference xmlns:c="http://example.org/claims"
(02) xmlns:p="http://schemas.xmlsoap.org/ws/2002/12/policy">
(03)   <wsa:Address>http://claimserver/ins/p.asmx</wsa:Address>
(04)   <wsa:ReferenceProperties>
(05)     <c:PatientProfile>123456</c:PatientProfile>
(06)     <c:CarrierID>987654</c:CarrierID>
(07)   </wsa:ReferenceProperties>
(08)   <wsa:PortType>c:ClaimsPortType</wsa:PortType>
(09)   <wsa:ServiceName PortName="c:ClaimsSoapPort">c:ClaimsService
(10)   </wsa:ServiceName>
(11)   <p:Policy>
(12)     ... <!-- policy statement omitted for brevity -->
(13)   </p:Policy>
(14)</wsa:EndpointReference>
```

(03)Adress: URI des Enpoints. Das einzige obligatorische Element einer EPR.

(04)-(07) RefernceProperties und RefernceParameters können vom Provider eines Endpoints benutzt werden, um eine bestimmte Instanz seines Web Services bzw. der verarbeitenden Anwendung zu identifizieren. Die darin enthaltenden Informationen sind ausschließlich für den jeweiligen Provider bestimmt, der auch die Informationen einer EPR hinzufügt. Sie dienen ihm später, Nachrichten schneller und einfacher der verarbeitenden Instanz seines Web Services zuzuordnen. In unseren Beispiel hat der Service eine spezielle Instanz zur Verarbeitung eines bestimmten Patientenprofils(05), da über seinen Web Service die Daten von mehreren Patienten gleichzeitig bearbeitet werden. Gleichzeitig wird mit einer CarrierID (06) ein bestimmter Abrechnungsposten beschrieben. Zusammen bilden sie einen bestimmten Abrechnungsposten für einen bestimmten Patienten, der von einer Instanz des Web Services bearbeitet wird. Endpoint Refernces mit der gleichen Address(03) aber unterschiedlichen ReferenceProperties, stellen somit auch unterschiedliche EPRs dar. Zusätzlich könnte der Provider noch Refernce Parameter hinzufügen, die keine neue Instanz identifizieren, aber die Bearbeitung zusätzlich erleichtern können. Der genaue Inhalt von ReferncePrameter und ReferenceProperties werden vom Aussteller bestimmt.

(08) Selected port type ist der qualifizierte Name des WSDL Port Typ des Web Services. der die abstrakte Beschreibung eines Web Services und seiner Operationen enthält.

1 Vgl. Aron Skonnard (2004), *Moving from WS-Routing to WS-Addressing Using WSE 2.0,* Internet http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsroutetowsadd.asp

(09-10)Service-port: enthält den qualifizierten Service name und port name, die eine statische Repräsentation des Endpoints darstellen. Er soll wie selcted port typ für Kompatibilität mit der WSDL Beschreibung sorgen.

(11-13)Policy des Endpoints, die beschreibt, welche Anforderungen,wie Sicherheitsprotokolle etc, erfüllt sein müssen, um an ihn eine Nachricht zu schicken.

2.2.2 Message Information Headers

Die Message Information Headers (MI), die in den Header Block von SOAP Messages eingefügt werden, stellen die notwendigen Adressierungs- Informationen zu Verfügung, um den Nachrichtaustausch zu realisieren. Dabei unterstützen sie auch Asynchrone Nachrichtenaustausch-Muster, wie in der folgenden Beispiel[2]-Nachricht: Ein Service schickt die Rechnungen(29) für einen Krankenhausaufenthalt eines Patienten an einen weiteren Service, der aus den vielen Rechnungen, eine genaue Abrechnung erstellt:

```
(02)  <s:Header>
(03)    <was:MessageID>uuid:someid</wsa:MessageID>
(04)    <wsa:Action>http://skonnard.com/SubmitClaim</wsa:Action>
(05)    <wsa:To>http://skonnard.com/Claims/Submit.asmx</wsa:To>
(06)    <wsa:From>
(07)     <wsa:Address>http://skonnard.com/main/sub.asmx</wsa:Address>
(08)       <wsa:ReferenceProperties>
(09)         <c:PatientProfile>123456</c:PatientProfile>
(10)         <c:CarrierID>987654</c:CarrierID>
(11)       </wsa:ReferenceProperties>
(12)    </wsa:From>
(13)    <wsa:ReplyTo>
(14)    <wsa:Address>http://skonnard.com/resp/resp.asmx</wsa:Address>
(15)       <wsa:ReferenceProperties>
(16)         <c:PatientProfile>123456</c:PatientProfile>
(17)         <c:CarrierID>987654</c:CarrierID>
(18)       </wsa:ReferenceProperties>
(19     </wsa:ReplyTo>
(20)    <wsa:FaultTo>
(21)    <wsa:Address>http://skonnard.com/fault/err.asmx</wsa:Address>
(22)    <wsa:ReferenceProperties>
(23)         <c:PatientProfile>123456</c:PatientProfile>
(24)         <c:CarrierID>987654</c:CarrierID>
(25)    </wsa:ReferenceProperties>
(26)    </wsa:FaultTo>
(27)  </s:Header>
(28)  <s:Body xmlns:c="http://example.org/claims">
(29)    <c:SubmitClaim> ... </c:SubmitClaim>
(30)  </s:Body>
(31)</s:Envelope>
```

2Vgl.Aron Skonnard (2004), *Moving from WS-Routing to WS-Addressing Using WSE 2.0,* Internet http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsroutetowsadd.asp

(5)To ist URI vom Endpoint, an dem die Nachricht geschickt wird, also vom Service, der die Abrechnung erstellt. Er ist genauso wie der Action Header obligatorisch.

(4)Action: Eine URI, die die Semantische Bedeutung der Nachricht angibt. Die URI identifiziert input, output oder fault message innerhalb eines WSDL Port Typs. Sie muss dazu mit der WSDL Beschreibung verbunden sein, was implizit oder explizit geschehen kann. Hier gibt sie an, dass es sich um eine eingehende Nachricht für die Abrechnungserstellung handelt.

(13-19)ReplyTo: Enthält die Endpoint Refernce des Endpoints, an dem die Antworten geschickt werden sollen. Der Service wünscht, dass die Antwort, also die Abrechnung für den Krankenhausaufenthalt, an einen anderen Service geschickt wird, der für die weitere Bearbeitung des Rechnungsposten des Patienten zuständig ist. Er hat eine andere Adresse, aber die gleichen ReferncеProperties, die den Patienten und den Rechnungsposten identifizieren und von einer entsprechnenden Instanz verarbeitet werden.

(06)-(12)From: Gibt die Quelle, also den Ursprung der Nachricht als EPR an.

(20)-(26)FaultTo: Gibt den Endpoint eines Services an, der für die Verarbeitung von fault messages zuständig ist, die im Falle eines Fehlers auftreten können. Um zu wissen bei welcher Instanz der Fehler aufgetreten ist, benötigt er auch die ReferncеProperties.

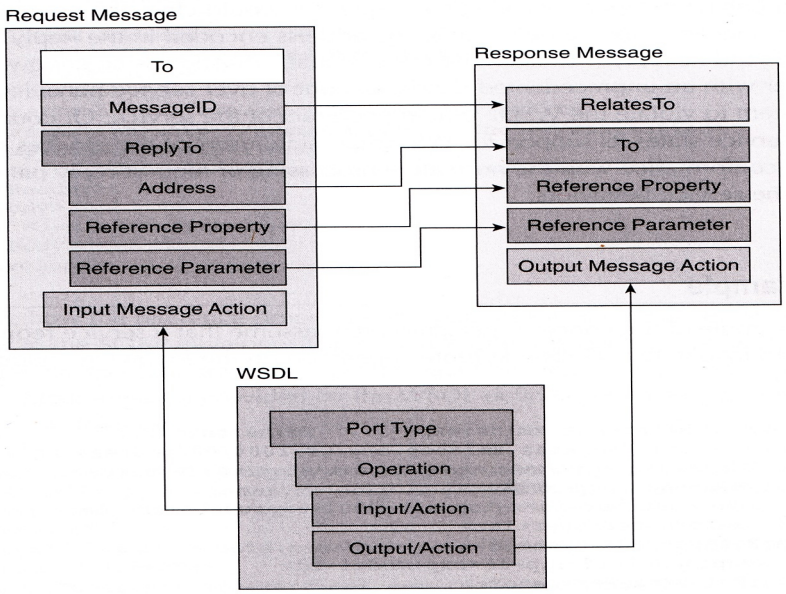
(03)MessageID: URI, die eine Nachricht in Raum und Zeit eindeutig identifiziert. Dieses Element wird immer benötigt, wenn eine Antwort erwartet wird oder ein FaultTo Element existiert, es also notwendig ist, eine Nachricht einer anderen zuordnen zu können.

Der Web Service zur Erstellung der genauen Krankenhausabrechnung empfängt die Beispiel Nachricht. Er erstellt für die mit der Nachricht mitgeschickten Krankenhausrechnungen(29) eine genaue Abrechnung und schickt diese an die vom Requestor gewünschte Antwortadresse (13-19). Er benutzt dazu die EPR aus dem ReplyTo Header, die er für die Erstellung der Response Message benötigt. Die folgende Abbildung[3] zeigt die Zusammenhänge von Request und Response

---

3 Vgl. S. Weerawarana / F. Curbera / F. Leymann / T. Storey / D. F. Ferguson, Web

Message und der WSDL Beschreibung des Service Providers, bei uns der Service zur Abrechnungserstellung, auf:



Das ganze auf das Beispiel übertragen sieht folgendermaßen aus[4]: Der erste Teil zeigt noch ein Mal die relevanten Elemente der Response Message:

```
<was:MessageID>uuid:someid</wsa:MessageID>
<wsa:ReplyTo>
<wsa:Address>http://skonnard.com/resp/resp.asmx</wsa:Address>
<wsa:ReferenceProperties>
<c:PatientProfile>123456</c:PatientProfile>
<c:CarrierID>987654</c:CarrierID>
</wsa:ReferenceProperties>
</wsa:ReplyTo>
```

Die farblich markierten Elemente werden folgendermaßen in der Response-Nachricht kopiert:

```
<s:Header>
   <wsa:RelatesTo>uuid:someid</wsa:RelatesTo>
   <wsa:To>http://skonnard.com/resp/resp.asmx</wsa:To>
   <c:PatientProfile>123456</c:PatientProfile>
   <c:CarrierID>987654</c:CarrierID>
  <wsa:Action>http://skonnard.com/SubmitClaimResponse</wsa:Action>
</s:Header>
<s:Body xmlns:c="http://example.org/claims">
  <c:SubmitClaim> ... </c:SubmitClaim>
</s:Body>
```

4Vgl.Aron Skonnard (2004), *Moving from WS-Routing to WS-Addressing Using WSE 2.0,* Internet http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsroutetowsadd.asp

```
</s:Envelope>
```

Die URI des Adress-Elements der EPR wird also in den To Header kopiert, der die Zieladresse abbildet. Die Elemente der EndpointRefernce bekommen einen eigenen Header. Der Service, der die Nachricht empfängt, kann dadurch die Abrechnung genau der Instanz zuordnen, die den entsprechenden Posten des Patienten bearbeitet. Zusätzlich muss noch ein **RelatesTo** Header eingefügt werden, der angibt, dass es sich um eine Antwort auf die Nachricht mit der MessageID „**uuid:someid**“ handelt. Es wird somit eine eindeutige Beziehung zur Request Message hergestellt.

2.3 Konsequenzen von WS-Addressing

Eine Konsequenz von WS-Addressing ist, dass der Requestor nun die volle Kontrolle darüber erlangt, wo und wie er eine Antwort erhalten will. Mit Endpoint Refernces und den MI-Headers wird die SOAP Message zu einer autonomen Einheit der Kommunikation. Die Einführung von EPRs bietet eine einheitliche Beschreibung von Endpoints zur Laufzeit, was deren Wiederverwendbarkeit und das aufspüren von Metadaten in Rahmen von WS-MetadataExchange möglich macht. Die Einführung der Mi Header führt dazu, dass die Interaktionslogik in die Prozesse hinein verlagert wird, was eine dynamische Festlegung des Nachrichtenaustauschs ermöglicht.

3. WS-Reliable Messaging (RM)

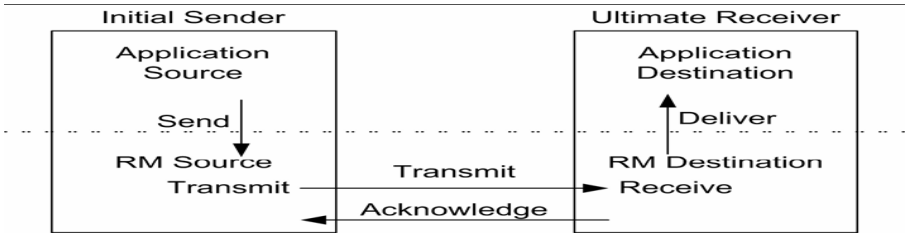
3.1 Motive, Problemfelder und Ziele von Reliable Messaging

Die nachrichtenbasierte lose Koppelung von Services bringt den Verlust der Kontrolle über den aktuellen Kommunikationsprozess mit sich: Wenn ein Web Service eine Nachricht über ein Netzwerk verschickt, weiß er weder, ob die Nachricht das Ziel erfolgreich erreicht, ob sie es nicht tut und damit eine erneute Übertragung notwendig wird, ob eine Folge von Nachrichten die beabsichtigte Reihenfolge eingehalten hat oder ob Nachrichten dupliziert worden sind. Kommt es zum Fehlerfall, so ist es extrem schwierig den Fehler zu erkennen und zu beheben. Ohne einen Standardprotokoll wie WS-Reiable Messaging müssten Entwickler die notwendigen Maßnahmen zur zuverlässigen Übertragung in der Buissenes Logik der Applikationen implementieren, was nicht nur Aufwendig ist, sondern auch zu

Problemen der Interoperabilität führen kann, da sie auf unterschiedlichen Middelware Systemen bzw. Infrastrukturen aufsetzen können. WS-Reliable Messaging Protokoll überbrückt unterschiedliche Infrastrukturen, indem es eine logische Ende-zu-Ende-Verbindung zwischen Web Services sicherstellt. Es ist interoperabel und transportunabhängig und garantiert, dass zwischen verteilten Anwendungen Nachrichten zuverlässig zugestellt werden können oder es zu einer aussagekräftigen Fehlermeldung kommt.

3.3 Reliable Messaging (RM) Modell und Terminology

Das Protokoll ist ähnlich aufgebaut wie das TCP Protokoll: Es übernimmt den zuverlässigen Transport für Daten für den Datenaustausch von darüber liegende Anwendungen. WS-RM Spezifikation führt dazu ein abstraktes Modell[5] für das Protokoll ein:



Die Rolle der Application Source wird typischerweise von den Anwendungen bzw. Services gespielt, von der eine Nachricht zuverlässig gesendet werden soll. Sie übergibt (“send“) die Nachricht an RM Source, die von diesen Punkt an die Verantwortung für die zuverlässige Übertragung oder beim Fehlerfall für die erneute Übertragung übernimmt. Sie sendet („transmit“) die Nachricht an RM Destination und nimmt von dieser die Bestätigung („Acknowledge“) entgegen und leitet die daraus möglicherweise resultierenden Maßnahmen ein. Die RM Destination empfängt die Nachrichten und bestätigt(„Acknowledge“) der RM Source den erfolgreichen Empfang und liefert („deliver“) die Nachricht an die darüber liegende Application Destination, die von der Anwendung gespielt wird, die am empfangenden Endpoint läuft. Die RM Destination übernimmt die Verantwortung für die zuverlässige Übertragung und bestimmt die genaue Konfiguration des RM Protokolls.

5Vgl. Donald Ferguson u.A. (2005), *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)* S.7

3.2 Nachrichtenübertragung mit RM

Ich werde nun anhand eines Beispiels[6] beschreiben, wie das RM Modell mit dem RM Protokoll realisiert wird. Ich beschreibe dabei nur den Fall, das ein Endpoint einen anderen eine Reihe von Nachrichten schicken will. Es lassen sich mit dem RM-Protokoll natürlich auch komplexere Nachrichtenaustausch-Muster mit mehr als zwei beteiligten Endpoints und bidirektionalen Nachrichtentausch bewerkstelligen. Ich beschreibe nun den Ablauf des Protokolls in mehreren Schritten:

1. Als erstes müssen die Vorbedingungen für das Protokoll erfüllt sein: Die RM Source muß die Endpoint Reference der RM Destination besitzen, die Policies ihres Endpoints kennen und in der Lage sein, die dort formulierten Anforderungen zu erfüllen.
2. Als nächstes muss die RM Source einen Anfrage(09) zur Erstellung einer Sequenz an die RM Destination richten, innerhalb derer die Nachrichten übermittelt werden. Dazu muss in eine SOAP Message Body ein CreateSequence Element (16)-(20) eingefügt werden. Dieses Element enthält ein AcksTo Element(17)-(19) mit der EPR für den Endpoint, an den die Bestätigungen für empfangene Nachrichten,die SequenceAcknowledgement Messages, oder Fehlermeldungen, gesendet werden.

```
(03) <S:Header>
(04)   <wsa:MessageID>
(05)http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
(06)   </wsa:MessageID>
(07)   <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
(08)    <wsa:Action>
(09)       http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence
(10)    </wsa:Action>
(11)   <wsa:ReplyTo>
(12)  <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
(13)   </wsa:ReplyTo>
(14) </S:Header>
(15) <S:Body>
(16)    <wsrm:CreateSequence>
(17)      <wsrm:AcksTo>
(18)  <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
(19)      </wsrm:AcksTo>
(20)    </wsrm:CreateSequence>
(21)  </S:Body>
(22)</S:Envelope>
```

3. Die RM Destination erstellt dann eine Sequenz und schickt der RM Source eine CreateSequenceResponse Message (07) mit einen **CreateSequenceResponse**

6 Vgl. Donald Ferguson u.A. (2005), *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)* S.31-37

Element (11)-(13), das einen eindeutigen Bezeichner (URI) der Sequenz(12) enthält, zurück.

```
(01)<S:Header>
(02)  <wsa:To>http://Business456.com/serviceA/789</wsa:To>
(03)   <wsa:RelatesTo>
(04)http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8a7c2eb546817
(05)   </wsa:RelatesTo>
(06)   <wsa:Action>
(07) http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequenceResponse
(08)   </wsa:Action>
(09)</S:Header>
(10)<S:Body>
(11)    <wsrm:CreateSequenceResponse>
(12)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(13)    </wsrm:CreateSequenceResponse>
(14)</S:Body>
(15)</S:Envelope>
```

4. Die RM Source beginnt nun mit dem Senden der Nachrichten. Sie fügt dabei dem SOAP-Header das Sequence-Header(10)-(13) Element hinzu, Es hat drei Kinderelemente: Den eindeutigen Bezeichner der Sequenz(11), eine Message Number(12), die die Nachricht innerhalb einer Sequenz eindeutig identifiziert, mit 1 anfängt und dann hochgezählt wird, und LastMessage, das eingefügt wird, wenn es sich um die letzte Nachricht einer Sequenz handelt:

```
(01)<S:Header>
(02)   <wsa:MessageID>http://Business456.com/guid/71e0654e-5ce8- (03)
477b-bb9d-34f05cfcbc9e </wsa:MessageID>
(04)   <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
(05)   <wsa:From>
(06)   wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
(07)   </wsa:From>
(08)   <wsa:Action>http://fabrikam123.com/serviceB/123/request
(09)   </wsa:Action>
(10)    <wsrm:Sequence>
(11)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(12)      <wsrm:MessageNumber>1</wsrm:MessageNumber>
(13)    </wsrm:Sequence>
(14) </S:Header>
(15) <S:Body>
(16)   <!-- Some Application Data -->
(17) </S:Body>
```

Die zweite Nachricht. MessageNumber(05) wird auf 2 erhöht

```
...
(01)<wsa:MessageID>
```

```
  http://Business456.com/guid/daa7d0b2-c8e0-476e-a9a4-
d164154e38de
(02)</wsa:MessageID>
(03)<wsrm:Sequence>
(04)   <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
```

```
(05)   <wsrm:MessageNumber>2</wsrm:MessageNumber>
(06)</wsrm:Sequence>
```

Die dritte und letzte, die nun das LastMessage(06) Element besitzt:

```
...
(01)<wsa:MessageID>
     http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546819
(02)</wsa:MessageID>
(03)<wsrm:Sequence>
(04)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(05)   <wsrm:MessageNumber>3</wsrm:MessageNumber>
(06)   <wsrm:LastMessage/>
(07)</wsrm:Sequence>
...
```

5. Die 2. Nachricht ist bei der Übermittlung verloren gegangen. RM Destination Antwortet mit einer SequenceAcknowledgement Message(10), da sie nun die letzte Nachricht erhalten hat. Der SOAP Header wird dabei mit dem Sequence-Acknowledgement Element(12)-(16) erweitert, das neben dem Bezeichner für die Sequenz(13), ein oder mehrere AcknowledgementRange Elemente(14)-(15) enthalten kann. Diese beschreiben jeweils zusammenhängende Bereiche von den empfangenen Nachrichtennummern, durch Angabe der niedrigsten(„lower“) und höchsten („upper“) Nummer. Dadurch lässt sich erschließen, welche Nachrichten Verloren gegangen sind. Hier ist es die 2. Nachricht.

```
(01)<S:Header>
(02)   <wsa:MessageID>
(03)http://fabrikam123.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546810
(04)   </wsa:MessageID>
(05)   <wsa:To>http://Business456.com/serviceA/789</wsa:To>
(06)   <wsa:From>
(07)   <wsa:Address>http://fabrikam123.com/serviceB/123</wsa:Address>
(08)   </wsa:From>
(09)   <wsa:Action>
(10)http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement
(11)   </wsa:Action>
(12)   <wsrm:SequenceAcknowledgement>
(13)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(14)     <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
(15)     <wsrm:AcknowledgementRange Upper="3" Lower="3"/>
(16)   </wsrm:SequenceAcknowledgement>
(17)</S:Header>
(18)<S:Body/>
```

6. Die RM Source sendet die 2. Nachricht erneut. Es ist eine neue Nachricht, die aber die gleiche MessageNumber(13) und Sequenz-Bezeichner(12) enthält. Auch

die MessageID(03) ist die gleiche! RM Source fügt der Nachricht ein AckRequested Element hinzu(15)-(17). Damit fordert sie vom empfangenden Endpoint eine Acknowledgement Message für die entsprechende Sequenz(16) an.

```
(01)<S:Header>
(02)  <wsa:MessageID>
(03)http://Business456.com/guid/daa7d0b2-c8e0-476e-a9a4-d164154e38de
(04)  </wsa:MessageID>
(05)  <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
(06)  <wsa:From>
(07)  <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
(08)  </wsa:From>
(09)  <wsa:Action>http://fabrikam123.com/serviceB/123/request
(10)  </wsa:Action>
(11)  <wsrm:Sequence>
(12)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(13)    <wsrm:MessageNumber>2</wsrm:MessageNumber>
(14)  </wsrm:Sequence>
(15)  <wsrm:AckRequested>
(16)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(17)  </wsrm:AckRequested>
(18)</S:Header>
(19)<S:Body>
(20)<!-- Some Application Data -->
(21)</S:Body>
```

7. Die RM Destination erhält die Nachricht und bestätigt den Empfang durch eine SequenceAcknowledgement Message:

```
      ...
(01)  <wsrm:SequenceAcknowledgement>
(02)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(03)    <wsrm:AcknowledgementRange Upper="3" Lower="1"/>
(04)  </wsrm:SequenceAcknowledgement>
...
```

8. RM Source erhält die Bestätigung und sendet eine TerminateSequence Message(02),(06)-(08) an die RM Destination, die ihr zu verstehen gibt, dass die Sequenz beendet ist, somit keine weiteren Nachrichten folgen, und dass sie so alle mit der Sequenz verbundenen Ressourcen freigeben kann:

```
...

(01)<wsa:Action>
(02)    http://schemas.xmlsoap.org/ws/2005/02/rm/TerminateSequence
(03)</wsa:Action>
(04)</S:Header>
(05)<S:Body>
(06)  <wsrm:TerminateSequence>
```

```
(07)<wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
(08)  </wsrm:TerminateSequence>
(09)</S:Body>
```

3.3 WS-RM und WS-Addressing

WS-Addressing und WS-RM sind eng miteinander verbunden: Das zeigt sich insbesondere beim messageID Element von WS-Addressing, das eigentlich immer einzigartig sein soll, aber für das bei RM eine Ausnahme gemacht wurde. Eine Realisierung des RM Protokolls ohne WS-Addressing ist zwar theoretisch machbar, aber nur sehr mühsam und kaum interoperabel zu realisieren. WS-Addressing und WS-RM bieteten eine zuverlässige und flexible Übertragung von Nachrichten, auf die viele weitere Spezifikationen und Technologien aufbauen. WS-RM steht auch im engen Zusammenhang mit WS-Policy, mit dem sich das nächste Kapitel beschäftigt.

4. WS-Policy

4.1 Beweggründe und Problemfelder

Beim RM- Beispiel. sieht man die Notwendigkeit von Policys, also Metadaten, die ein domänspezifisches Verhalten beschreiben: Zwei interagierenden Endpoints müssen sich darüber einig sein, dass sie ihre Nachrichten über das RM Protokoll verschicken und dass sie WS-Addressing dazu in Kombination benutzen. Diese Übereinkunft stellen sie über Policys her: Eine Policy beschreibt, dass das RM Protokoll erforderlich ist und kann darüber hinaus noch angeben, wie dieses Protokoll genau konfiguriert werden soll.

Policys werden also dazu benutzt, die Bedingungen für eine Interaktion zwischen zwei Endpoints zu beschreiben. Das typische Szenario ist, das ein Service Requestor den Web Service eines Providers nutzen will. Der Provider gibt mit Hilfe von Policys an, welche Bedingungen ein Requestor erfüllen muss, um den Service nutzen zu können. Die Bedingungen werden mit Hilfe von „Assertions“, Erklärungen für ein domänspezifisches Verhalten, beschrieben. Da diese Bedingungen in einen maschinenlesbaren Format geschrieben sind, kann der Requestor sie mit Hilfe von entsprechenden Tools benutzten, um Code-Module zu erzeugen, die das geforderte Verhalten abbilden und sie für einen Service-Request nutzen. Wird das WS-RM und WS-Addressing Protokoll etwa vom Provider verlangt, so kann ein Requestor über diese Tools automatisch einen Service Request erstellen, der die Protokolle benutzt - vorausgesetzt natürlich, dass seine Tools diese auch unterstützen. Folgende weitere Szenarien sind typische Anwendungsfälle von WS-Policy:

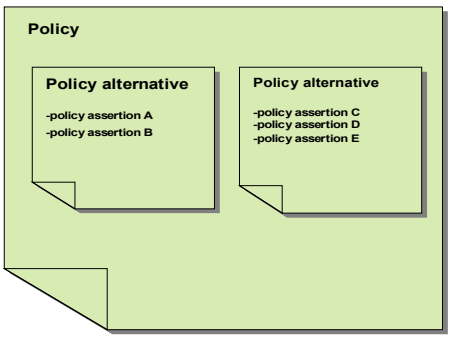
- WS-Policy in Kombination mit WSDL stellt einen Beschreibung, der Anforderungen eines Services an einen Requester zu Verfügung, die ihm anleiten, Anwendungen zu entwickeln und zu benutzen
- Finden und Auswahl geeigneter Web Services: WS-Policy ermöglicht es Service Requestors einen passenden Service zu finden, der neben der gewünschten Funktionalität auch andere Qualitäten besitzt, wie Sicherheitsstandards und Garantien für Zuverlässigkeit.
- Dynamisches Aktualisierung: Interagierende Endpoints können mit Hilfe von WS-Policy ihre Konfiguration zur Laufzeit aktualisieren und spezifische Interaktionen anpassen. So können Requestors etwa über MetadataExchange Policy informationen vom Provider zur Laufzeit erhalten, um sich neuen Anforderungen anpassen zu können.

4.2 Das WS-Policy Framework

Das WS-Policy Framework realisert ein Modell und eine Syntax, Policys für Web Services beschreiben und veröffentlichen zu können. Es besteht aus zwei Spezifikationen:

- WS-Policy beschreibt ein abstraktes Modell und Grammatik, mit der man die Anforderungen einer Web Service Einheit beschreiben kann
- WS-PolicyAttatchment beschreibt Mechanismen, mit denen man Policys an ein bestimmte Web Service Einheit, wie WSDL, UDDI, EPRs, binden kann.

Das WS-Policy Modell führt ein abstraktes Modell für Policys ein:



Die bereits erwähnten **Policy Assertions** sind dabei ein Stück Metadaten, die ein spezifisches Verhalten für eine Domain, wie Messaging, Sicherheit, Zuverlässigkeit, Transaktionen, identifizieren. Assertions werden von den Autoren einer Domain definiert und müssen eindeutig identifizierbar und konsistent sein. So sind für fast alle WS-*Prtokoll Spezifikationen Policy Assertions definiert. Der Typ einer Assertion wird dabei durch einen Namespace Namen und einen lokalen Namen eindeutig festgelegt. Assertions können Parameter, Attribute und weitere Assertions als Kindelemente besitzen, die das Verhalten genauer spezifizieren. Die WS-RM Policy Assertion z.B. definiert definiert mit der „**RMAssertion“**, dass das RM Protokoll für

die Nachrichtenübertragung benutzt werden muss. Mit Parametern kann darüberhinaus das genaue Verhalten des Protokolls beschrieben werden.

Eine Policy kann mehrerer dieser Assertions enthalten. Assertions können dabei zu **Policy Alternatives** zusammengefasst werden, die jeweils eine erlaubte Kombination von Assertions darstellen. Dies gibt Service Providern die Möglichkeit Requestors eine Auswahl, über eine geeignete Kombination von Assertions, zu Verfügung zu stellen. Ein Requestor erfüllt die Policy eines Providers, wenn er alle beschrieben Assertions einer einzigen Policy Alternative umsetzt.

4.2.1 Policy Expressions

Policy Expressions setzen das Policy Modell um**,** indem sie Policys als XML Infosets darstellen: Die erste Element einer Policy Expression ist das Policy-Element. Es ist ein Container für eine Sammlung von Assertions oder kombinierten Assertions. Neben Policy gibt es noch einen ALL und ExactlyOne Operator, um Assertions miteinander zu kombinieren, wie im folgenden Beispiel[7]:

```
(01)<wsp:Policy
(02) xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
(03) xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
(04) xmlns:wsap_"http://www.w3.org/2006/05/wsdl
(05)xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
(06) <All>
(07)   <wsrm:RMAssertion>
(08)     <wsrm:InactivityTimeout Milliseconds="600000" />
(09)     <wsrm:BaseRetransmissionInterval Milliseconds="3000" />
(10)     <wsrm:ExponentialBackoff />
(11)     <wsrm:AcknowledgementInterval Milliseconds="200" />
(12)   </wsrm:RMAssertion>
(13)   <wsap:UsingAdressing/>
(14)   <wsp:ExactlyOne>
(15)     <sp:Basic256Rsa15/>
(16)     <sp:TripleDesRsa15/>
(17)   </wsp:ExactlyOne>
(18)  <ALL>
(19)<wsp:Policy>
```

ALL(06)-(08) oder **Policy** bedeutet, dass alle mit ihm kombinierten Assertions erforderlich sind. In unserm Beispiel sind dies das RM-Protokoll(07)-(12) und WS-Addressing(13). Das RM Protokoll wird noch weiter durch Parameter spezifiziert: **Inactivity(08)** gibt einen Zeitraum an, nachdem eine Sequenz auf Grund von Inaktivität beendet wird. **BaseRetransmissionInterval(09)** gibt einen Zeitraum an, nachdem unbestätigte Nachrichten erneut gesendet werden. Das zeitintervall wird

7 Eigenes Beispiel

durch einen Backoff-Algorithmus berechnet**(10)**.Das **AckowledgmentInterval (11)g**ibt ein Zeitintervall an, nach dem die RM Destination eine Acknowledgment Message an die RM Source sendet. **Der ExactlyOne(14)-(17)** Operator besagt, dass genau eine Assertion von mehreren erforderlich ist. Hier gibt es also die Möglichkeit einer Auswahl zwischen zwei kryptographischen Verschlüsselungstechniken (15) und(16), von denen genau eine ausgewählt werden muss. Die Policy erfordert also nicht nur WS-RM und WS-Addressing, sondern auch noch das Verschlüsslungsverfahren Basic256Rsa15 oder TripleDesRsa15. WS-Policy beschreibt einen Algorithmus solche verschachtelte Policy Ausdrücke in einer **Normalform** auszudrücken, die dem abstrakten Policy Modell entspricht, wo eine Policy eine Menge von mehreren Alternativen ist, vor der man genau eine erfüllen muss. Die Normalform besteht also aus einen ExactlyOneOperator(02),(17) der alle Alternativen,(03)-(09) und (10)-(16), umfasst, wie es in der Normalform für das Beispiel dargestellt ist:

```
(01)<Policy>
(02)  <ExactlyOne>
(03)    <All>
(04)      <wsrm:RMAssertion>
(05)      ....
(06)      </wsrm:RMAssertion>
(07)      <wsap:UsingAdressing/>
(08)      <sp:Basic256Rsa15/>
(09)    </All>
(10)    <All>
(11)      <wsrm:RMAssertion>
(12)      ....
(13)      </wsrm:RMAssertion>
(14)      <wsap:UsingAdressing/>
(15)      <sp:TripleDesRsa15/>
(16)    </All>
(17)  </ExactlyOne>
(18)<Policy/>
```

4.2.2 Policy Intersection

Der **Policy Intersection** Mechanismus nutzt die Normalform, um die übereinstimmenden Alternativen zweier Policys zu ermitteln. Da Policys eine Menge von Alternativen sind, kann man eine Schnittmenge zweier Policys ermitteln, die Alternativen enthält, die übereinstimmen. Dieser Mechanismus kann z.B. von einen Requestor benutzt werden, der durch eine Policy seine Anforderungen und Fähigkeiten beschreibt, also etwa welche Protokolle er unterstützen kann. Er kann

nun mit dem Policy Intersection Mechanismus seine Policy mit der eines Providers abgleichen. Die ermittelte Schnittmenge gibt ihn an, welche Policy Alternativen er benutzen kann. Ist die Schnittmenge leer, weiß er, dass er nicht den Service des Providers nutzen kann. Der Mechanismus überprüft allerdings nur die Assertions einer Alternative auf Typ-Gleichheit. Er kann nicht die Attribute und Parameter vergleichen. Um also wirklich eine Kompatibilität sicherstellen zu können, müssen weitere Überprüfungen stattfinden.

4.3 WS-PolicyAttatchment

Bis jetzt wurde nur beschrieben, wie Anforderungen einer Web Services mittels Policys beschrieben werden, aber nicht wie sie mit einer Web Services Einheit verbunden werden können. **WS-Policy Attatchment** beschreibt zwei allgemeine Mechanismen, um Policys mit Web Services Einheiten, wie UDDI, WSDL, EPRs, zu verbinden: **XML Element Attatchment** realisiert das Verbinden innerhalb von beliebigen XML Elementen, die eine Ressource beschreiben. Dabei gibt es zwei Möglichkeiten, die Policys einzufügen:Über das **PolicyRefernce** Element oder über das **wsp:PolicyURIs** Attribut:[8]

```
(01)<MyElement wsp:PolicyURIs="
(02)   http://www.fabrikam123.com/policies#RmPolicy
(03)   http://www.fabrikam123.com/policies#X509EndpointPolicy" />
```

Das Attribut weist einen Element über die PolicyURI(01) referenzierte Policys(02) und (03) zu. Die resultierende Policy ist eine Policy, die die referenzierten Policys enthält:

```
(00)<wsp:Policy
(01)xmlns:rmp="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
(02)xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
(03)xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(04)  <rmp:RMAssertion>
(05)    <rmp:InactivityTimeout Milliseconds="600000" />
(06)    <rmp:BaseRetransmissionInterval Milliseconds="3000" />
(07)    <!-- Details omitted for readability -->
(08)  </rmp:RMAssertion>
(09)  <sp:AsymmetricBinding>
(10)    <wsp:Policy>
(11)      <!-- Details omitted for readability -->
(12)      <sp:IncludeTimestamp />
(13)      <sp:OnlySignEntireHeadersAndBody />
(14)    </wsp:Policy>
(15)  </sp:AsymmetricBinding>
(16)</wsp:Policy>
```

Dieser Mechanismus wird vor allem bei WSDL Beschreibungen benutzt. Service Provider können dort so angeben, welche Anforderungen sie an einen Requestor

8 Vgl. Siddharth Bajaj / u.A. (2006)*Web Services Policy Attachment (WSPolicyAttachment) Version 1.2,* S.9

haben. Da WSDL Beschreibungen sich aber überlappen- so gibt es eine abstrakte und konkrete Beschreibung eines Services- überlappen sich auch die entsprechenden Policys. WS-Policy definiert, wie sich dabei die effektiven Policys ergeben.

Der andere allgemeine Mechanismus um Policys an Web Services Einheiten bzw. Ressourcen zu binden, soll nur kurz erwähnt werden. Er erfolgt über das **externe Policy Attachment,** das es ermöglicht Policys mit Web Service Einheiten außerhalb ihrer Definition zu verbinden. Die externe Anbindung von Poliys ist vor allem nützlich, um Policys zur Laufzeit an einen Endpoint zu binden, um Anforderungen abhängig vom Laufzeitverhalten formulieren zu können.

4.4 WS-Policy Fazit

Die WS-Spezifikationen, wie WS-RM und WS-Addressing, benötigen WS-Policy, um sie interoperabel einsetzen zu können: Service Requestor erfahren nämlich durch Policys, welche Spezifikationen bzw. Protokolle gefordert sind und wie diese ggf. konfiguriert werden müssen. Während WSDL-Beschreibungen nur beschreiben, was ein Web Service tut, beschreiben Policys wie er es tut und was er dafür benötigt, und ergänzen damit WSDL sinnvoll um eine qualitative Dimension. WS-Addressing, WS-Policy und WS-RM bauen auf einander auf und stellen elementare Konzepte zu Verfügung, auf die viele weitere WS-* Spezifikationen aufsetzen.

Anhang

Literaturverzeichnis:

Wolfgang Dostal / Mario Jeckle / Ingo Melzer / Barbara Zengler (2005): *Service-orientierte Architekturen mit Web-Services. Konzepte - Standards – Praxis*, Heidelberg: Spektrum Akademischer Verlag

Thomas Erl(2004): *Service-Oriented Architecture, A Field Guide to Integrating XML and Web Services,* Prentice Hall

Thomas Erl (2005): *Service-Oriented Architecture. Concepts, Technology and Design*, Prentice HallBeth Linker(2005): Introduction to WS-Addressing, Internet http://dev2dev.bea.com/lpt/a/28 Stand: 01-31-2005, Abruf 2006-11-27

Sanjiva Weerawarana / Francisco Curbera / Frank Leymann / Tony Storey / Donald F. Ferguson (2005), *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL WS-Reliable Messaging, and More*, Prentice Hall PTR

Siddharth Bajaj / Don Box / Dave Chappell / Francisco Curbera / u.A. (2006), *Web Services Policy Framework (WSPolicy) Version 1.2*, Internet http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf, Stand:2006-03-?? Abruf 2006-11-27

Siddharth Bajaj / Don Box / Dave Chappell / Francisco Curbera / u.A. (2006),*Web Services Policy Attachment (WSPolicyAttachment) Version 1.2*, Internet http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polatt/ws-polat-2006-03-01.pdf, Stand:2006-03-?? Abruf 2006-11-27

Don Box /Doug Davis / Donald Ferguson/ u.A. (2005), *Web Services Reliable Messaging Policy Assertion (WS-RM Policy)*, Internet ftp://www6.software.ibm.com/software/developer/library/ws-rmpolicy200502.pdf Stand: 2005-02-??, Abruf 2006-11-27

Don Box / Erik Christensen / Donald Ferguson/ u.A. (2004), *Web Services Addressing (WS-Addressing)*,Internet www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/ Stand: 2004-08-10, Abruf 2006-11-27
Aaron Skonnard (2004): *Moving from WS-Routing to WS-Addressing Using WSE 2.0* , Internet http://msdn2.microsoft.com/en-us/library/ms996537.aspx, Stand: 2004-04-??, Abruf 2006-11-27

Donald Ferguson / Tony Storey / Brad Lovering / John Shewchuk (2003), *Reliable, Transacted Web Services*,Internet http://www-128.ibm.com/developerworks/webservices/library/ws-securtrans/, Stand: 2003-10-28 Abruf:2006-27-11

Wolfgang Dostal / Mario Jeckle (2004), *Semantik, Odem einer Service-orientierten Architektur*, ,Internet http://jeckle.de/semanticWebServices/intro.html, Stand:2004 Abruf 2006-11-27

Asir S. Vedamuthu / Daniel Roth (2006), *Understanding Web Services Policy*, Internet http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/understwspol.asp Stand: 2006-07-06 Abfrage: 2006-11-27