



universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

Funktionsweise und Aufbau von aktuellen
Webservicearchitekturen unter näherer Betrachtung
der Webservice-Frameworks WSRF und WSN

Verfasser

Michael Jäger

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Wien, am 29.Sept.2008

Studienkennzahl: A175

Studienrichtung: Wirtschaftsinformatik

Betreuer: Ao. Univ.-Prof. Dipl.-Ing. Dr. Siegfried Benkner

Vorwort

Wenn man sich mit den Technologien der Webservices befasst, muss man feststellen, dass es sich um einen umfangreichen Themenbereich handelt, welcher in viele Bereiche der heutigen IT-Welt einwirkt.

Als Autor dieser Arbeit war es eine große Herausforderung für mich, sich mit der Bewältigung des Themengebietes auseinanderzusetzen, da es sehr viele interessante Anwendungsbereiche im Webserviceumfeld gibt. Das liegt darin begründet, dass es sich bei Webservices um eine Technologie handelt, welche in der heutigen IT-Welt zu einem immer wichtigeren Thema geworden ist.

Zu Beginn der Arbeit war es deswegen relevant, sich zu entscheiden aus welcher Perspektive man an das Thema herangeht. Beispielsweise kann man die Webservicethematik rein aus der „Buisness“-Sicht betrachten. Diesbezüglich sind Fragen zu behandeln wie:

- Wo liegen die Anwendungs- und Einsatzgebiete von Webservices?
- Wie kann die Integration von Webservices in bestehende IT-Systeme erfolgen?
- Welche Standardisierungsbestrebungen gibt es von seiten der Wirtschaft?
- Wie hoch ist der zeitliche Aufwand, um Webservices einzusetzen?
- Wie hoch sind die Investitionen, die getätigt werden müssen?
- Wie können Webservices die heutigen Informationssysteme verändern?
- usw.

Auf der anderen Seite lassen sich Webservices aus der rein technischen Sicht betrachten, wobei es hierbei auch wieder mehrere Betrachtungswinkel zu unterscheiden gibt.

Generell habe ich beim Lesen der Literatur festgestellt, dass es oft gar nicht so eine große allgemeine Übereinstimmung gibt, um was es sich konkret beim Begriff „Webservices“ handelt.

Umso schwieriger war es zunächst herauszufinden, aus welcher Perspektive man sich an Begriffe wie SOA, SOAP, WSDL, WSRF oder WSN heranwagen kann. Diesbezügliche Fragen die sich mir stellten, waren u.a :

- Sollen die Standards im Rahmen von konkreten Implementierungen aus der Sicht eines Programmierers aufgezeigt werden?
- Soll der Schwerpunkt auf die automatisierte Integration in Unternehmenslandschaften unter Verwendung von Webservice-Toolkits gelegt werden?
- Wie sind die Standards im Kontext mit SOAs bzw. Gridsystemen einzuordnen?
- Wie ist der konkrete Architekturaufbau der Standards definiert?

Ich habe im Rahmen dieser Arbeit einen Schwerpunkt auf die letzt genannte Frage gelegt: also auf den Strukturaufbau der Spezifikationen.

Das genaue Betrachten einer Webservice-Spezifikation gehört in der heutigen IT-Welt nicht unbedingt zu den Tätigkeiten mit denen man sich andauernd auseinandersetzt. Das liegt vor allem daran, dass es eben gerade das große Ziel der Webservicetechnologie ist, dass die Integration und das Ablaufen einer Webservice-Interaktion automatisiert erfolgen soll. Auch Entwickler können Webservicekomponenten heutzutage unter Verwendung von Entwicklungstools erstellen, ohne sich eingehend mit einer eher abstrakt aufgebauten Spezifikation auseinandersetzen zu müssen.

Natürlich wird zunächst in einem ersten Teil an die allgemeine Thematik der Webservices herangeführt. Inhalt dieser Erläuterung ist die generelle Funktionsweise, ein Überblick über die Standardisierung, sowie die Beschreibung einer SOA. Des weiteren werden die Basisstandards SOAP und WSDL anhand ihres Aufbaus veranschaulicht. Die eingehende Betrachtung der neueren Webservicespezifikationsfamilien wie WSRF und WSN steht im Vordergrund dieser Diplomarbeit.

Trotz der vielen interessanten Themengebiete war es natürlich notwendig das Thema einzugrenzen, da sonst der Rahmen der Arbeit gesprengt werden würde. Verzichtet wird ganz bewusst u.a. auf Implementierungsszenarien, sowie auf ausführliche Erläuterungen zum Thema Grid Computing.

Eidesstattliche Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Michael Jäger

Gänserndorf, am 29.09.2008

Danksagung

Während des Studiums und der Erarbeitung der vorliegenden Diplomarbeit haben mich viele Personen begleitet und unterstützt. Dafür möchte ich allen meinen herzlichen Dank aussprechen.

Mein Dank gilt Herrn Ao. Univ.-Prof. Dipl.-Ing. Dr. Siegfried Benkner, der mir die Auseinandersetzung mit diesem Thema erst ermöglichte.

Ganz besonderer Dank gilt vor allem meinen Eltern Johanna und Friedrich Jäger, die mir während meines Studiums eine große Unterstützung waren.

Gänserndorf, am 29.09.2008

INHALTSVERZEICHNIS

Vorwort	3
Eidesstattliche Erklärung	5
Danksagung	6
INHALTSVERZEICHNIS	7
I. ZUSAMMENFASSUNG	12
II. ABSTRACT	13
III. ABBILDUNGSVERZEICHNIS	14
IV. TABELLENVERZEICHNIS	16
V. ABKÜRZUNGSVERZEICHNIS	18
1 Einleitung	19
1.1 Ziel der Diplomarbeit	22
2 Grundlagen Webservices	23
2.1 Einführung.....	23
2.2 IT-Systeme im Wandel	24
2.3 Webservice Definitionen.....	27
2.4 Merkmale (Manhart, 2006)	30
2.4.1 Programmierbarkeit	30
2.4.2 Protokolltransparenz.....	30
2.4.3 Kapselung.....	30
2.4.4 Lose Kopplung.....	30
2.4.5 Komposition	30
2.4.6 Ortstransparenz	30
2.4.7 Selbstbeschreibung	30
2.4.8 Transaktions-Charakter	31
2.5 Vor- und Nachteile (Sotomayor, 2005, S. 6).....	31

2.5.1	Vorteile	31
2.5.2	Nachteile.....	31
2.6	Webservice Funktionsweise.....	32
2.6.1	Service Requestor (Clientapplikation).....	32
2.6.2	Ablauf des Webserviceaufrufes (Sotomayor, 2005, S. 11)	33
2.6.3	Service Provider (Ausführungsumgebung) (Sotomayor, 2005, S. 11f)	34
2.7	Web Service Architektur.....	35
2.8	Basisstandards.....	37
2.8.1	SOAP.....	37
2.8.2	WSDL	37
2.8.3	UDDI.....	38
2.9	Erweiterte Standards.....	38
2.9.1	Zuverlässigkeit.....	39
2.9.2	Transaktionen und Operationen	39
2.9.3	Buisness Prozesse	39
2.9.4	Sicherheit.....	39
2.9.5	Ressourcen	39
2.10	Serviceorientierte Architektur.....	40
2.10.1	SOA - Konzept	41
2.10.2	Definition – SOA.....	41
2.10.3	Defintion – Service	42
2.10.4	SOA – Szenario.....	43
3	SOAP	44
3.1	Grundkonzept.....	44
3.2	Art des Nachrichtenaustausches.....	46
3.3	Nachrichtenpfad und Nachrichtenknoten	47

3.4	Nachrichtenverarbeitungsmodell	48
3.5	Nachrichtenstruktur	49
3.5.1	Envelope.....	49
3.5.2	Header.....	50
3.5.3	Body	52
3.6	SOAP-Encoding	53
3.7	SOAP-Fehlerbehandlung	53
3.8	SOAP Binding	54
4	WSDL.....	56
4.1	Allgemeines.....	56
4.2	Aufbau.....	56
4.3	Elemente	58
4.3.1	Element <Description> (vormals <definitions>)	59
4.3.2	Element <Types>.....	60
4.3.3	Element <Interface> (vormals <portType>)	61
4.3.4	Element <binding>.....	62
4.3.5	Element <Service>	63
4.3.6	zusätzliches Element <documentation>	64
4.3.7	Verlinkung von WSDL-Dokumenten	64
5	WSRF.....	66
5.1	Webservices und Zustandsmodellierung.....	66
5.1.1	Webservices – ein ursprünglich, zustandsloses Medium.....	67
5.1.2	Mangel an Konventionen bei bisherigen Standards.....	68
5.2	Web Service Resource Framework – Das Konzept	69
5.2.1	Allgemeines	69
5.2.2	Definitionen.....	72

5.2.3	Stateful Resource	73
5.2.4	WS-Resource	74
5.2.5	WS-Ressourcen Kommunikation (The Implied Resource Pattern) .	75
5.2.6	WS-Resource Zustandsinformationen	79
5.2.7	Deklaration am Beispiel eines Shoppingservices:	80
5.2.8	Vergleich zu Servicedefinitionen ohne WSRF-Nutzung.....	82
5.3	Abfrage und Update von Zustandsinformationen	85
5.3.1	ResourceProperties-Operationen	85
5.3.2	Beispielszenario Satellitenverwaltung.....	86
5.4	Fehlerbehandlung	90
5.4.1	BaseFault-Funktionsweise (Liu & Meder, 2006, S. 6).....	91
5.4.2	Erzeugung von neuen Fehlern	92
5.5	Lebenszyklus von WS-Ressourcen	94
5.5.1	Allgemeines	94
5.5.2	Erstellung der WS-Resource	95
5.5.3	Zerstörung der WS-Resource.....	97
5.6	WS-Ressourcen Gruppierung	104
5.6.1	Inhalt der Spezifikation	105
5.6.2	ServiceGroup.....	106
5.6.3	ServiceGroupEntry	109
5.6.4	ServiceGroupRegistration.....	110
5.6.5	Kombinierbarkeit mit anderen Webservice-Standards.....	113
5.7	Geschichtlicher Hintergrund von WSRF.....	114
5.7.1	Gridkonzept	114
5.7.2	OGSI.....	116
5.7.3	WSRF als gridspezifischer Ansatz.....	118

6	WSN.....	119
6.1	Allgemeines.....	120
6.1.1	Konzept	120
6.1.2	Inhalt der Spezifikation	121
6.1.3	Begriffserklärung.....	122
6.1.4	Notification	124
6.2	WS-Base Notification	127
6.2.1	Allgemeines	127
6.2.2	Schnittstelle Notification Consumer	130
6.2.3	Filterverwendung (Niblett & Graham, 2005).....	132
6.2.4	Schnittstelle Notification Producer	133
6.2.5	Subscription	139
6.3	WS-Topics.....	143
6.3.1	Begriff Topic.....	143
6.3.2	Begriff TopicNamespace	145
6.3.3	Begriff TopicSet	147
6.3.4	Begriff TopicExpression	148
6.4	WS-Brokered Notification	150
6.4.1	BrokeredNotificationPattern.....	150
6.4.2	Deklarationen und Funktionalitäten von WS-Brokered	152
6.4.3	Vorteile der Brokered Notification (im Vergleich zum direct notification pattern).....	153
6.4.4	Publisher-Interaktion mit dem Notification Broker.....	154
7	Zusammenfassung und Ausblick	157
	LITERATURVERZEICHNIS	161

I. ZUSAMMENFASSUNG

Webservicetechnologien verwenden Standardarchitekturen, mit denen ein automatisierter Datenaustausch zwischen Softwarekomponenten über Rechengrenzen hinweg ermöglicht werden kann.

Die folgende Diplomarbeit beschäftigt sich mit einer eingehenden Betrachtung von aktuellen Webservice-architekturen. Einen Schwerpunkt bildet u.a die detaillierte Funktionsbeschreibung der Webserviceframeworks WSRF und WSN, welche zu den erweiterten Business-Standards zählen und auch die Interaktionen mit etablierten Kernstandards wie SOAP und WSDL gut unterstützen.

Das WSRF ist ein Gruppe von fünf Teilspezifikationen, welche klare Richtlinien für den Zugang zu zustandsbehafteten Ressourcen bereitstellt. Das Framework definiert hierfür eine WS-Resource, welches eine Kombination aus einem Webservice und einer "stateful resource" repräsentiert und gleichsam als ein XML-Dokument und einem WS-Adressing Endpunkt dargestellt wird.

In den WSRF-Teilspezifikationen sind eine Reihe von Patterns und Mechanismen für die Definition, die Operation, den Lebenszyklus, die Gruppierung , sowie die Fehlerbehandlung von WS-Ressourcen definiert.

Nicht nur die Zustandsmodellierung von Webservices, sondern auch das Event-Handling stellen in vielen Computersystemen eine nicht zu vernachlässigbare Notwendigkeit dar.

Das WSN , welches sich aus drei Teilspezifikationen zusammensetzt, stellt einen Mechanismus zur Verfügung, welcher für das Auslösen von Ereignissen in IT-Infrastrukturen verwendet werden kann. Potenzielle Nutzer können damit standardisierte Informationen über Komponenten, Verfügbarkeit und freie Kapazitäten von anderen Instanzen abrufen.

Die angeführten Standardframeworks bieten eine gute Voraussetzung, um den Anforderungen in Bezug auf Ressourcennutzung in gegenwärtigen aber auch zukünftigen IT-Architekturen gerecht zu werden.

II. ABSTRACT

Web Services technologies use standard architectures, in order to enable automatic integration and data-exchange between software components and applications.

The following diploma thesis is concerned with a close consideration of actual Web Services architectures. A focus is put on a detailed functional description of the webservice frameworks WSRF and WSN, which belong to the extended business standards and which also support the interactions with well-established core standards like SOAP and WSDL.

WSRF is a group of five specifications, which provide clear conventions for the access to state-afflicted resources. For this purpose the framework provides a definition for a WS-Resource, which combines a Web Service with a “stateful resource” and which is represented as an XML-Document and a WS-Addressing endpoint. WSRF standardizes a set of patterns and mechanism for the definitions, the operation, the lifecycle, the grouping, and the fault-handling of WS-Resources. Not only the state-modeling of Web Services, but also the event-handling outlines a not negligible need in many computer-systems.

WSN, which consists of three specifications, provides a mechanism, which can be used for the release of events in IT-infrastructures. Potential users are enabled to retrieve standardised information about components, availabilities and free capacities of other instances.

The standard frameworks offer a good basis to fulfil the requirements in reference to resource utilizations in actual and in prospective IT-architectures.

III. ABBILDUNGSVERZEICHNIS

Abbildung 1 Baukastenprinzip (Öztürk & Stiller, 2004)	21
Abbildung 2 Webserviceumgebung in sehr vereinfachter Darstellung (Sotomayor, 2005, S. 6)	24
Abbildung 3 Heterogene Middleware-Architektur ohne Einsatz von Webservices (Alonso, Casati, Harumi, & Machiraju, 2004, S. 129).....	25
Abbildung 4 Hohe Interoperabilität durch Webservices (Alonso, Casati, Harumi, & Machiraju, 2004, S. 135).....	26
Abbildung 5 WS-Ablauf einer WS-Interaktion (Benkner, 2006, S. 10)	29
Abbildung 6 Kompilierung von Stubs und Skeletons aus der WSDL-Spezifikation (Alonso, Casati, Harumi, & Machiraju, 2004, S. 154).....	33
Abbildung 7 Serverseitiger Aufbau eines Webservices (Sotomayor, 2005, S. 12).....	34
Abbildung 8 Web Services Architektur (Booth, et al., Web Services Architecture, 2004)	35
Abbildung 9 Zusammenspiel von SOAP, WSDL und UDDI (Srinivasan & Treadwell, 2005, S. 2)	43
Abbildung 10 Aufbau einer SOAP-Nachricht (Hauser & Löwer, 2004, S. 47)	45
Abbildung 11 Einfache SOAP-Übertragung (Skonnard, 2003).....	46
Abbildung 12 SOAP-Übertragung mit Request/Response (Skonnard, 2003)	46
Abbildung 13 SOAP-Nachrichtenübertragung mit Zwischenknoten (Skonnard, 2003)	47
Abbildung 14 Abstrakte und konkrete Beschreibung von Webservices (Dhesiaseelan, 2004)..	57
Abbildung 15 WSDL 2.0 Infoset-Diagramm (Nilo & Lafon, 2007).....	58
Abbildung 16 Zustandsloser Webservice-Aufruf (Sotomayor, 2005, S. 14)	66
Abbildung 17 Zustandsbehafteter Webservice-Aufruf (Sotomayor, 2005, S. 15).....	67
Abbildung 18 Implied Resource Pattern (Papazoglou, 2007, S. 219)	70
Abbildung 19 Einordnung in Bezug auf andere Webservice-Standards (Sabbah, 2004, S. 8) ..	71
Abbildung 20 Webservice mit 3 Ressourcen (Graham, Karmarkar, Mischkinsky, Robinson, & Sedukhin, 2006, S. 6).....	74
Abbildung 21 Reference enthält einen stateful Resource Identifier (Foster, et al., 2004b, S. 12)	78
Abbildung 22 SOAP-Anfrage unter Verwendung einer qualified endpoint reference (Foster, et al., 2004b, S. 14)	79
Abbildung 23 Einfacher Einkaufswagen (Banks, 2006, S. 13)	80
Abbildung 24 Einfaches Shopping Service (Banks, 2006, S. 10)	81
Abbildung 25 SOAP-Nachricht mit und ohne WSRF (Banks, 2006, S. 8)	83
Abbildung 26 WS-Resource mit WS-ResourceProperties-Schnittstellen (Joseph, Ernest, & Fellenstein, 2004).....	85
Abbildung 27 ServiceGroup (Maguire, Snelling, & Banks, 2006, S. 11).....	106
Abbildung 28 OGSA Plattform Architektur (Joseph, Ernest, & Fellenstein, 2004, S. 628)	116
Abbildung 29 Komponenten der OGSF (Joseph, Ernest, & Fellenstein, 2004, S. 631)	117
Abbildung 30 WSRF als Schnittstelle zwischen Webservices und GridComputing (Foster, 2004a, S. 7).....	118

Abbildung 31 Grundlegende Rollen von WSN (Graham, et al., 2004, S. 414)	121
Abbildung 32 Abfolge der Operationen zwischen Subscriber, NotificatoinProducer und NotificationConsumer	126
Abbildung 33 subscribeRequest eines NotificationsConusmers (Niblett & Graham, 2005, S. 873)	128
Abbildung 34 Szenario Printer-Management (Niblett & Graham, 2005, S. 873).....	130
Abbildung 35 Beispiel Aktienkursinformation: Benachrichtigung an mehrere Notification Consumer(Niblett & Graham, 2005, S. 873)	130
Abbildung 36 TopicTree telephoneContact . (Graham, et al., 2004, S. 421)	144
Abbildung 37 Darstellung eines TopicNamespace (Vambenepe, Graham, & Niblett, 2006, S. 10)	146
Abbildung 38 Konzept der BrokeredNotification (Niblett & Graham, 2005, S. 879).....	151
Abbildung 39 Nachrichtenablauf bei Simple Publishing (Niblett & Graham, 2005, S. 880)	155
Abbildung 40 Nachrichtenablauf bei Demand basierter Publisher (Niblett & Graham, 2005, S. 881).....	156
Abbildung 41 Das beste aus den beiden Welten WebServices und Grid(Sabbah, 2004, S. 2)	158
Abbildung 42Einordnung bezüglich Automation und Semantik(Dostal & Jeckle, 2004)	159

IV. TABELLENVERZEICHNIS

Tabelle 1 Beispiel für SOAP-Envelope (Chase, 2007, S. 44).....	50
Tabelle 2 Beispiel-Listing: Routing Information im Header (Chase, 2007, S. 14)	52
Tabelle 3 Beispiel-Listing: Payload innerhalb des Bodys (Nilo & Lafon, 2007)	52
Tabelle 4 Beispiel-Listing für <description> (Nilo & Lafon, 2007)	60
Tabelle 5 Beispiel-Listing für <Types> (Nilo & Lafon, 2007).....	60
Tabelle 6 Beispiel-Listing für <interface> (Nilo & Lafon, 2007).....	61
Tabelle 7 Beispiel-Listing für <binding> (Nilo & Lafon, 2007).....	63
Tabelle 8 Beispiel-Listing für <service> (Nilo & Lafon, 2007)	64
Tabelle 9 Beispiel-Listing für <documentation> (Nilo & Lafon, 2007).....	64
Tabelle 10 XML Infoset (Gudgin, Hadley, & Rogers, 2006).....	77
Tabelle 11 Resource Properties Document (Banks, 2006, S. 18)	81
Tabelle 12 WSDL-Import Element (Banks, 2006, S. 16)	81
Tabelle 13 ESDL-Schema für types (Banks, 2006, S. 16).....	82
Tabelle 14 WSDL-Schema für portType (Banks, 2006, S. 16).....	82
Tabelle 15 Verwendung von standardisierten Operationen (Banks, 2006, S. 15).....	84
Tabelle 16 Response-Listing (Banks, 2006, S. 17)	84
Tabelle 17 Deklaration des Namensraums des Satellitensystems (Sundaram, 2005a, S. 8)...	87
Tabelle 18 Deklaration RPD des Satellitensystems (Sundaram, 2005a)	88
Tabelle 19 Bekanntgabe des Identifier im Zuge des Create Requests (Sundaram, 2005a, S. 13)	88
Tabelle 20 Header des SOAP-Requests (Sundaram, 2005a, S. 22)	89
Tabelle 21 Body des SOAP-Requests (Sundaram, 2005a, S. 22)	89
Tabelle 22 Header des SOAP Response (Sundaram, 2005a, S. 23).....	90
Tabelle 23 Body des SOAP-Response (Sundaram, 2005a, S. 23)	90
Tabelle 24 Beispiel für einen BaseFault (Sundaram, 2005a, S. 38).....	91
Tabelle 25 XML-Schema für die Erweiterung von Fehlertypen (Sundaram, 2005a, S. 47)	92
Tabelle 26 XML-Schema für retournierten Fehler (Sundaram, 2005a, S. 47).....	93
Tabelle 27 SOAP-Request create (Sundaram, 2005a, S. 20)	96
Tabelle 28 SOAP-Response der endpoint refernce (Banks, 2006, S. 20)	96
Tabelle 29 SOAP-Request für die Zerstörung einer WS-Resource(Sundaram, 2005b, S. 6)....	98
Tabelle 30 SOAP-Request für die Zerstörung einer WS-Resource(Sundaram, 2005b, S. 7)....	99
Tabelle 31 Elemente des ResourcePropertiesDocument des Satellitensystems (Sundaram, 2005b, S. 7).....	101
Tabelle 32 Header-Teil (Sundaram, 2005b, S. 7).....	101
Tabelle 33 Body-Teil (Sundaram, 2005b, S. 7).....	102
Tabelle 34 Header-Teil (Sundaram, 2005b, S. 8).....	102
Tabelle 35 Body-Teil (Sundaram, 2005b, S. 8).....	102
Tabelle 36 SOAP-Request für eine neue TerminationTime (Sundaram, 2005b, S. 9).....	103
Tabelle 37 SOAP-Response für eine neue TerminationTime (Sundaram, 2005b, S. 10).....	103

Tabelle 38 SOAP-Request für eine neue ServiceGroup (Sundaram, 2005b, S. 15)	106
Tabelle 39 SOAP-Response für eine neue ServiceGroup (Sundaram, 2005b, S. 10)	107
Tabelle 40 Format des MembershipContentRule resource property Elements (Maguire, Snelling, & Banks, 2006, S. 13).....	108
Tabelle 41 Beispiel-Listing RPD der ServiceGroup (Sundaram, 2005b, S. 21).....	108
Tabelle 42 Struktur einer Entry Resource Property (Maguire, Snelling, & Banks, 2006, S. 13).....	108
Tabelle 43 Beispiel-Listing RPD (Sundaram, 2005b, S. 17)	109
Tabelle 44 Beispiel Listing eines RPD der ServiceGroupEntry (Sundaram, 2005b, S. 16).....	110
Tabelle 45 Header-Teil (Sundaram, 2005b, S. 20)	111
Tabelle 46 Body-Teil (Sundaram, 2005b, S. 20).....	111
Tabelle 47 Header-Teil (Sundaram, 2005b, S. 20)	112
Tabelle 48 Body-Teil (Sundaram, 2005b, S. 20).....	112
Tabelle 49 Listing: „Notify“ – Standardformat für Notifications am Beispielszenario Satellitensystem (Sundaram, 2005c, S. 5)	125
Tabelle 50 TopicSpace (Sundaram, 2005c).....	125
Tabelle 51 WSDL-Deklaration am Beispiel Satellitensystem (Sundaram, 2005c).....	131
Tabelle 52 Beispiel für ein RPD eines NotificationProducers (Sundaram, 2005c)	134
Tabelle 53 Angabe der Namensräume	134
Tabelle 54 Angabe des Topic-spezifischen Schemas(Graham, Hull, & Murray, 2006, S. 14) ...	135
Tabelle 55 Header-Teil - Request bei der Erstellung einer Subscription (Sundaram, 2005c, S. 16).....	136
Tabelle 56 Body-Teil - Request bei der Erstellung einer Subscription (Sundaram, 2005c, S. 16)	137
Tabelle 57 Response bei der Erstellung einer Subscription(Sundaram, 2005c, S. 17)	137
Tabelle 58 SOAP-Request (Header- und Body-Teil) (Sundaram, 2005c)	138
Tabelle 59 SOAP-Response (Header- und Body-Teil) (Sundaram, 2005c)	139
Tabelle 60 SOAP-Request (Header- und Body-Teil)(Sundaram, 2005c, S. 24).....	141
Tabelle 61 SOAP-Request (Header- und Body-Teil)(Sundaram, 2005c, S. 24).....	142
Tabelle 62 XML Darstellung der Topics telephoneContact (Graham, et al., 2004, S. 421).....	144
Tabelle 63 XML-Darstellung des TopicNamespace (Vambenepe, Graham, & Niblett, 2006, S. 10).....	146
Tabelle 64 XML-Darstellung TopicSet(Vambenepe, Graham, & Niblett, 2006, S. 11)	147
Tabelle 65 XML-Beispiel für Simple TopicExpression(Vambenepe, Graham, & Niblett, 2006, S. 18).....	149
Tabelle 66 XML-Struktur für Concrete TopicExpressoin (Vambenepe, Graham, & Niblett, 2006, S. 20)	149

V. ABKÜRZUNGSVERZEICHNIS

- API Application Programming Interface
- B2B Business-to-Business
- CORBA Common Object Request Broker Architecture
- CRM Customer Relationship Management
- DCE Distributed Computing Environment
- DCOM Distributed Component Object Model
- EJB Enterprise JavaBeans
- EPR Endpoint Reference
- FTP File Transfer Protocol
- GED Global Element Declaration
- HTML Hypertext Markup Language
- HTTP Hypertext Transfer Protocol
- IDL Interface Definition Language
- J2EE Java Platform, Enterprise Edition
- MEP Message Exchange Patterns
- OASIS Organization for the Advancement of Structured Information Standards

- QoS Quality of Services
- RMI Remote Method Invocation
- RPC Remote Procedure Call
- SOA Service Oriented Architecture
- SOAP Simple Object Access Protocol
- SMTP Simple Mail Transfer Protocol
- TCP/IP Transmission Control Protocol/Internet Protocol
- UDDI Universal Description, Discovery and Integration

- UTC Universal Time Format
- WS Web Service
- WSDL Web Service Definition Language;
Web Service Description Language

- WSN Web Service Notification
- WSRF Web Service Resource Framework
- W3C World Wide Web Konsortium
- XML Extensible Markup Language

1 Einleitung

Die automatisierte Vernetzung von verteilten Applikationen ist in den letzten Jahren zu einem immer wichtigeren Thema in der IT-Welt geworden.

Der Begriff "Distributed Computing" bezeichnet hierfür ein Paradigma, welches es ermöglicht, dass unterschiedliche Applikationen über Rechnergrenzen hinweg miteinander kommunizieren können. Hierfür existieren zahlreiche Technologien, welche für Interaktion zwischen verteilten Komponenten eingesetzt werden können. Vertreter dieser Technologien, wie beispielsweise CORBA oder Java RMI, stellen Verzeichnisse zur Verfügung, mit welchem es anderen Anwendungen ermöglicht wird, gewünschte Komponenten aufzufinden. Der Aufruf dieser Komponenten kann dann so stattfinden, als ob sich diese auf dem lokalen System befinden würden. Applikationen können somit auf die Ressourcen von anderen Systemen zugreifen, obwohl sie sich auf verschiedenen geografischen Standorten befinden. Der große Nachteil all dieser Systeme ist jedoch die Tatsache, dass es sich quasi noch immer um ein geschlossenes System handelt. Letztendlich muss hierbei die Client-Applikation noch immer dieselbe Technologie wie die Server-Applikation verwenden.(Chase, 2007)

Wünschenswert wäre eine Lösung, wo es eher eine losere Kopplung zwischen den verteilten Kommunikationspartnern gibt, damit eine automatisierte Interaktion ohne aufwändige Anpassungen erfolgen kann. Webservices gelten als eine Kommunikationstechnologie, welche diese Anforderungen erfüllen kann.

„Der Begriff Webservices beschreibt ein wichtiges verteiltes Distributed Computing-Paradigma, welches sich gegenüber anderen Ansätzen wie DCE, CORBA und Java RMI vor allem dadurch unterscheidet, dass internet-basierende Standards (wie z.B. XML) verwendet werden, um heterogenes Distributed Computing zu realisieren. Web Services definieren Vorgehensweisen zur Beschreibung von Softwarekomponenten, Methoden für den Komponentenzugriff, und Auffindungsmethoden zur Identifizierung von relevanten Service Providern. Webservices sind programmiersprachen- und betriebssystemunabhängig.“(Foster, Kesselman, Nick, & Tuecke, 2002)

Das World Wide Web Consortium stellt folgende Definition für Webservices bereit:

„Ein Webservice ist ein Softwaresystem, welches entworfen wurde ,um interoperable "machine-to-machine"-Interaktionen über ein Netzwerk zu unterstützen. Es stellt eine Schnittstelle bereit, welche in einem maschinenausführbaren Format (WSDL) beschrieben ist. Andere Systeme interagieren mit dem Webservice mittels SOAP-Nachrichten in jener Art, wie es seine Beschreibung vorsieht. Typischerweise werden diese Nachrichten über HTTP mit einer XML-Darstellung in Verbindung mit anderen web-bezogenen Standards übertragen.“(Booth, et al., Web Services Architecture, 2004, S. 7)

Im Allgemeinen handelt es sich bei Webservices um in sich abgeschlossene Softwarekomponenten, auf welche mittels offener standardisierten Schnittstellen von anderen Systemen zugegriffen werden kann.(UDDI.org, 2001, S. 1)

Die Funktionalitäten bzw. die Ressourcen dieser "Webdienste" können auf einfache Art und Weise von anderen Applikationen genutzt werden. Der Kommunikationsaustausch basiert auf Basis von standardisierten xml-basierten Nachrichtenmustern. Zwischen den Komponenten besteht eine lose Kopplung. Webservices können beispielsweise für die automatisierte Kommunikation bei Buchungsanfragen zwischen Reisebüros und Fluggesellschaften eingesetzt werden.

Um sich ein Bild über die Funktionsweise von einem Webservice-Szenario zu machen, kann folgendes Beispiel anhand einer Reisebürobuchung betrachtet werden:

Um eine Flugbuchung automatisch abzuwickeln, nimmt ein Intranet-Server des Reisebüros unter Verwendung eines standardisierten Formulars einen Buchungswunsch entgegen. Dann werden an die Webservices von geeigneten Fluggesellschaften oder anderen Reisebüros automatisch Anfragen gestellt. Ein geeigneter Webservice wird automatisch ausgewählt. Es wird ein Nachrichtenaustausch zu diesem hergestellt und das Ergebnis wird weiters unmittelbar in den bestehenden unternehmensinternen Geschäftsprozess des Reisebüros weitergereicht. (Ilg & Fischer, 2003)

Wie man anhand des genannten Beispiels erkennen kann, ist es ein wichtiges Kriterium, offene Informationsarchitekturen zu entwickeln, um Applikationen auf einfache Art und Weise miteinander zu verbinden.

Die Lösung liegt in der Standardisierung. Heutzutage ist es notwendig, dass standardisierte Technologien geschaffen werden, denn nur durch Standards kann man Systeme verbinden und integrieren, die von beliebigen Herstellern produziert werden. Dadurch wird optimale Interoperabilität erreicht.

Vor allem das gemeinsame Zusammenspiel von offenen Standards macht das Wesen von Webservices aus. Diesbezüglich kann der Vergleich mit einem Baukastensystem (siehe Abbildung 1) herangezogen werden.

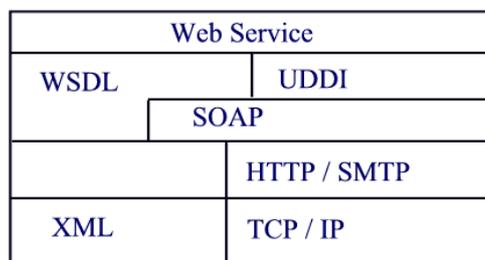


Abbildung 1 Baukastenprinzip (Öztürk & Stiller, 2004)

Heutzutage gibt es bereits eine Fülle von Spezifikationen und Standards für viele unterschiedliche Bereiche, welche für das Webservice-Umfeld von einer Vielzahl von Organisationen und Anbietern entwickelt werden.

Eine der großen Herausforderungen der Standardisierungsprozesse war es unter anderem auch, jene Probleme zu bewältigen, die im Zusammenhang mit Authentifizierungen, Transaktionen oder zustandsbezogenen Operationen auftraten. Der Grund dafür ist, dass Webservices grundsätzlich zustandslos sind. D.h. Clientapplikationen speichern prinzipiell keine Informationen bzw. Zustände zwischen zwei hintereinander folgenden Webservice-Interaktionen ab.

Die Herausforderung bestand darin, Lösungen zu entwickeln, den prinzipiellen Vorteil der Zustandslosigkeit für Webservices zu erhalten, aber trotzdem die Möglichkeit zu schaffen, zustandsgebundene Operationen durchzuführen.

Mit der Entwicklung der vom OASIS-Konsortium vorangetriebenen Standards WSRF und WSN wurde schließlich für die sonst generell zustandslosen Webservices eine Lösung bezüglich Zustandsmodellierung entwickelt.

1.1 Ziel der Diplomarbeit

Im Rahmen dieser Diplomarbeit sollen aktuelle Webservice-Standardarchitekturen vorgestellt werden. Zu Beginn der Arbeit werden theoretische Grundlagen erklärt. Dabei wird das Prinzip der Web Services vorgestellt und die Idee einer SOA erläutert.

Der Schwerpunkt der Arbeit beschäftigt sich hauptsächlich mit der Vorstellung der jeweiligen Spezifikationen, die von Webservices verwendet werden.

Dazu gehören zunächst die von der Organisation W3C freigegebenen Spezifikationen SOAP und WSDL, welche als die Basisstandards für Webservices gelten. In einem weiteren Teil werden dann die vom OASIS-Konsortium vorangetriebenen Standards WSRF und WSN vorgestellt, welche sich mit der Thematik der Zustandsmodellierung für Webservices beschäftigen.

Ein Ausblick soll schließlich Auskunft geben, in welche Richtung sich die Webservicetechnologien weiterentwickeln können, um die zukünftigen IT-Systeme nachhaltig zu verändern.

2 Grundlagen Webservices

2.1 Einführung

Wenn man sich mit der Thematik von Web Services auseinandersetzt, lässt sich feststellen, dass der Begriff zu einem beliebten Modewort in der IT-Welt geworden ist. Oft werden Webservices als die zukunftsweisende Technologie angepriesen, deren Einsatz zu völlig neuartigen Informationssystemen führen soll.

Vereinfacht ausgedrückt, handelt es sich bei Webservices um eine weitere Technologie des Distributed Computing wie CORBA, RMI, EJB usw. Webservices erlauben es somit Client-Server Applikationen zu realisieren. (Sotomayor, 2005, S. 6)

Das Webservice-Konzept ist generell vergleichbar mit der klassischen Request-Response Interaktion zwischen einer Client- und einer Server-Komponente.

An dieser Stelle ist zunächst anzumerken, dass Webservices tatsächlich wenig mit dem HTML-zentrierten World Wide Web zu tun haben, auch wenn durchaus eine gewisse Analogie erkennbar ist. Beim klassischen Webansatz wird eine Anfrage an einen entfernten Webserver gestellt, welcher wiederum die Information im HTML-Format als Antwort zurückgibt. Für eine mögliche Weiterverarbeitung muss ein menschlicher Nutzer jedoch die Information ablesen und dann gegeben falls die Daten in einer anderen Anwendung neuerlich zur Weiterverarbeitung neu eintragen.

Im Unterschied zum klassischen Webeinsatz steht beim Webservice-Konzept der automatische Ablauf im Vordergrund – dh. ohne direkte Interaktion menschlicher Nutzer. Eine Client-Applikation kann beispielsweise die im Rahmen einer Abfrage von einem Webservice erhaltenen und maschinell lesbaren Daten selbstständig zur Weiterverarbeitung weiterreichen. Man spricht in diesem Zusammenhang auch von Maschine-zu-Maschine Kommunikation. (Hauser & Löwer, Web Services - Die Standards, 2004)

Die folgende Abbildung soll zunächst veranschaulichen, wie eine beispielhafte Webserviceumgebung in einer sehr vereinfachten Darstellung sein kann.



Abbildung 2 Webserviceumgebung in sehr vereinfachter Darstellung (Sotomayor, 2005, S. 6)

Bsp: Ein Client möchte einen bestimmten Webservice bzw. Dienst nutzen. Der Client schickt einen Servicerequest an einen Webservice um die Wetterdaten für einen bestimmten Ort zu erhalten. Das Webservice antwortet daraufhin mit einer Serviceantwort. Die Kommunikation erfolgt durch Übertragung von Nachrichten. Dabei werden standardisierte Nachrichtenmuster bzw. Protokolle verwendet.

2.2 IT-Systeme im Wandel

Die heutige IT-Welt muss sich heutzutage ständig an neue wechselnde Bedingungen anpassen. Die Industrie ist sich der Tatsache bewusst, dass IT-Systeme in der heutigen dynamischen Welt nur bestehen können, wenn sie eine hohe Flexibilität, Wandlungsfähigkeit und Wirtschaftlichkeit aufweisen können.

Von der Webservice-Technologie erwartet man sich in erster Linie, dass sie die Interoperabilitäts-Probleme in den heutigen IT-Landschaften lösen kann. Im allgemeinen lassen sich Webservices sowohl für EAI- als auch für B2B Einsatzgebiete verwenden.

Die Herausforderung für die Webservice-Technologie liegt besonders in der unternehmensübergreifenden Vernetzung (B2B).

Im zu Beginn angeführten Flugbuchungsszenario wurde bereits erwähnt, dass eine automatisierte Integration über Unternehmensgrenzen hinweg sehr gut mittels Webservices umgesetzt werden kann. Im Gegensatz dazu sind konventionelle Middleware-Systeme hierbei schon oft an ihre Grenzen gestoßen. So können Schwierigkeiten beispielsweise insbesondere dann auftreten, wenn die Geschäftspartner eines Unternehmens jeweils unterschiedliche Middleware-Plattformen verwenden.

Für all die IT-basierten geschäftlichen Interaktionen, welche dann im Rahmen eines Geschäftsprozess-Szenarios anfallen, wäre es somit notwendig, mehrere unterschiedliche Middleware-Systeme zu integrieren, was natürlich zur Folge hat, dass diesbezüglich große Investitionen getätigt werden müssen.

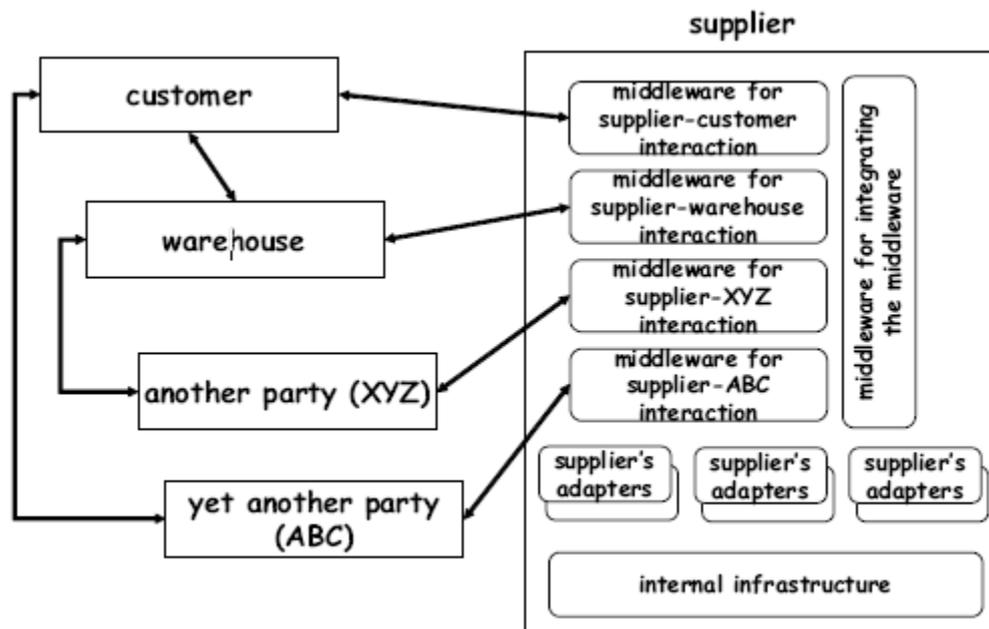


Abbildung 3 Heterogene Middleware-Architektur ohne Einsatz von Webservices (Alonso, Casati, Harumi, & Machiraju, 2004, S. 129)

Webservices haben jedenfalls die Eigenschaft, dass diese über bereits bestehende konventionelle Middleware-Plattformen als zusätzliche Schicht implementiert werden können. Bereits existente Middleware-Services können somit für die Clients von anderen Unternehmen aufrufbar gemacht werden.

Eine der Stärken von Webservices ist es also, dass die Funktionalitäten von bereits bestehenden Applikationen als Schnittstelle für Clientapplikationen freigelegt werden können und somit eine optimale Interoperabilität gewährleistet werden kann. Die nachfolgende Abbildung veranschaulicht diese Tatsache.

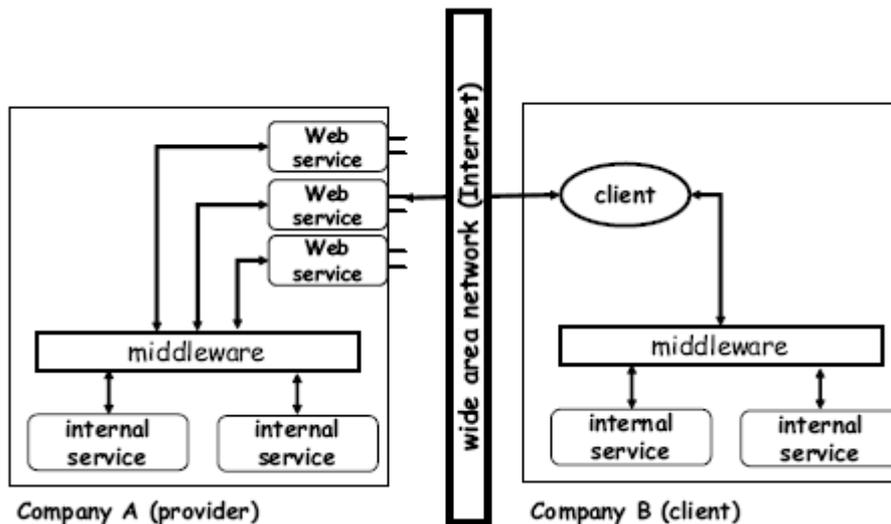


Abbildung 4 Hohe Interoperabilität durch Webservices (Alonso, Casati, Harumi, & Machiraju, 2004, S. 135)

Das Prinzip der losen Kopplung ist jenes wichtige Charakteristikum, welches Webservices im Besonderen zu anderen Ansätzen unterscheidet.

Technologien wie CORBA und EJB sind vor allem für verteilte Systeme ausgerichtet, welche zueinander eine sehr enge Kopplung aufweisen. Ein Client und ein Server weisen hier eine hohe Abhängigkeit zueinander auf. Diese Systeme sind gut geeignet für Intranet Applikationen und haben bei Internet-Größenordnungen eher ihre Schwächen. (Sotomayor, 2005, S. 7)

Webservices sind hingegen eher für sogenannte „loosely coupled systems“ geeignet. Bis zum Zeitpunkt des Aufrufes muss ein Client hierbei keinerlei früheres Wissen über den Webservice aufweisen. Somit können Webservices sehr gut für internet-weite Applikationen eingesetzt werden. Diese sind deshalb ebenso sehr gut für grid-orientierte Belange geeignet. (Sotomayor, 2005, S. 7)

Aufgrund der Tatsache, dass Webservices die Eigenschaft der losen Kopplung aufweisen, sind sie als technische Implementationsart für das Modell von serviceorientierten Architekturen (SOA) gut geeignet.

An dieser Stelle ist jedenfalls anzumerken, dass die lose Kopplung und die daraus sich ergebenden Vorteile vor allem dadurch erreicht werden können, weil die Webservice-Technologie eine Reihe von Standards (XML) verwendet, was auch den bedeutenden Unterschied zu anderen Ansätzen ausmacht.

Die Internettechnologie am Beispiel von Standardprotokollen wie HTTP ist ein Beispiel dafür, dass man letztlich nur durch einen hohen Grad an Standardisierung heterogene Systeme über Unternehmensgrenzen hinweg verbinden und integrieren kann.

2.3 Webservice Definitionen

Im Laufe der letzten Jahre hat es verschiedene Auffassungen gegeben wie Webservices definiert werden. Zahlreiche Definitionen sind einerseits sehr allgemein gehalten, andererseits aber auch sehr spezifisch und restriktiv (beschränkend, einschränkend).

Oft wird ein Webservice als eine Applikation angesehen, welche für andere Applikationen über das Web zugänglich ist. Dabei handelt es sich jedoch um eine sehr offene Definition, welche den Eindruck vermittelt, dass alles was eine URL hat, gleichzeitig auch ein Webservice ist. Es wird hierbei nicht auf eine standardisierte Schnittstelle hingewiesen, was jedoch im Besonderen die korrekte Definition von Webservices ausmacht.

Eine präzisere Definition wird vom UDDI Konsortium bereitgestellt, welche Webservices wie folgt definiert:

"Webservices are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces"(UDDI.org, 2001)

Diese detaillierte Definition betont vor allem, dass Internet Standards eingesetzt werden. Des weiteren wird ausgedrückt, dass das Service (bzw. der Dienst) eine offene Schnittstelle haben muss, welche über das Internet aufgerufen werden kann. In der Definition kommt aber noch nicht klar zum Ausdruck, was genau mit "modular business applications" gemeint ist. (Alonso, Casati, Harumi, & Machiraju, 2004, S. 124)

Eine präzisere Definition stellt schließlich das World Wide Web Konsortium (W3C) bereit:

"A Web Service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanges via Internet-based protocols." (Booth, et al., Web Services Architecture, 2004)

In dieser Definition wird vor allem betont, dass Webservices fähig sein sollen, "definiert, beschrieben und gefunden" zu werden. Somit sollen Möglichkeiten geschaffen werden, um Clients zu schreiben, die mit den Webservices interagieren können. Webservices sind somit Komponenten, welche in komplexere verteilte Applikationen integriert werden können. (Alonso, Casati, Harumi, & Machiraju, 2004, S. 124)

Zum besseren Verständnis, um was es sich bei Webservices konkret handelt, lässt sich noch folgende weitere Definition anführen. *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"* (Booth, et al., Web Services Architecture, 2004)

In dieser Definition werden nun Standards wie SOAP oder WSDL angeführt. Zunächst ist anzumerken, dass XML als die zentrale Technologie für Webservices gilt und dabei als Datenformat eingesetzt wird. Mit XML werden die Daten bzw. Nachrichten, welche bei Webservice-Aufrufen verwendet werden, strukturiert. Alle andere Webservicesstandards wie SOAP, WSDL, etc bauen auf XML auf.

Mit WSDL wird die Beschreibung und Definition der Schnittstellen (API) eines Webservices erreicht. Ein Webservice-Consumer erhält damit eine Anleitung wie ein potentiell Webservice aufgerufen und genutzt werden kann.

SOAP ist ein Protokollstandard und verpackt die Nachrichten in einen Umschlag. Der Transport erfolgt über internetbasierende Übertragungsprotokolle wie beispielsweise HTTP.

Die folgende Abbildung veranschaulicht zunächst eine grobe Darstellung einer Webservice-Interaktion und zeigt wie die genannten Standards im Kontext dazu einzuordnen sind.

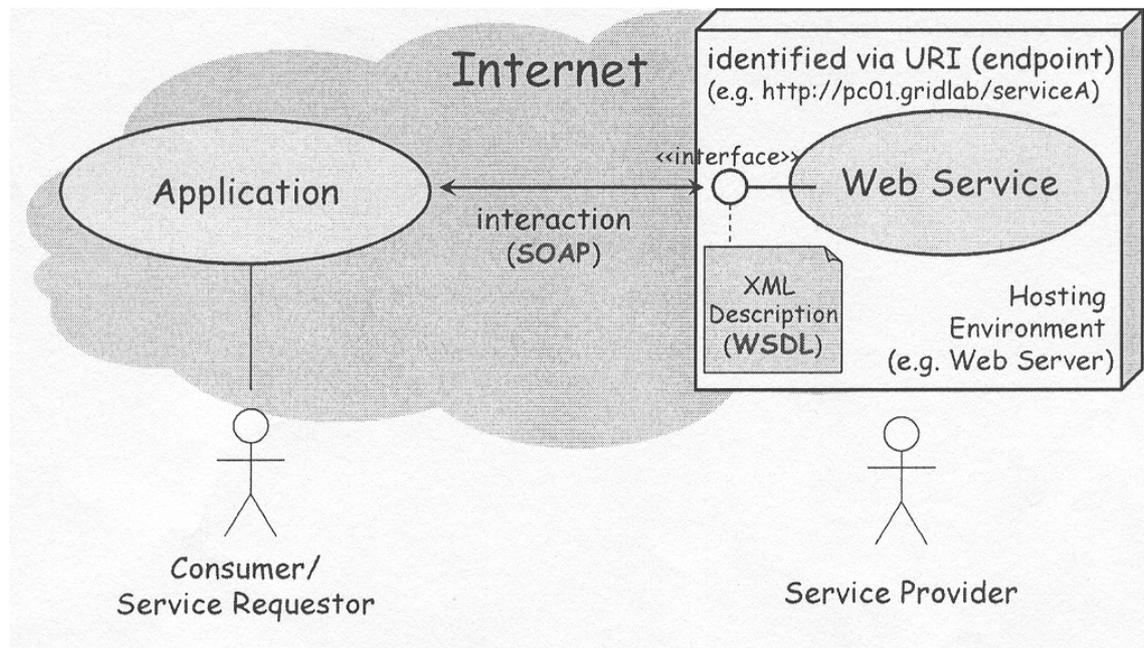


Abbildung 5 WS-Ablauf einer WS-Interaktion (Benkner, 2006, S. 10)

Zusammenfassend lassen sich aus den zuvor genannten Definitionen folgende Schlüsseigenschaften eines Webservices anführen (Kossmann & Leymann, 2004):

Die Identifikation eines Web Services erfolgt durch einen URI¹.

- Das Interface bzw. die Schnittstelle eines Web Services ist maschinenlesbar.
- Die Beschreibung erfolgt durch WSDL.
- Ein Web Service kommuniziert mit anderen Softwarekomponenten durch XML Nachrichten. Der Nachrichtenaustausch kann insbesondere mit Hilfe von Internetprotokollen (z.B. HTTP oder SMTP) erfolgen.
- Web Services sind autonom. Das heißt, dass man nicht beeinflussen kann, ob und wie eine Nachricht von einem Web Service verarbeitet wird.
- Die Regelung von Qualitätseigenschaften wie z.B. Antwortzeitgarantien muss durch zusätzliche Vereinbarungen erfolgen.

¹URI = Uniform Resource Identifier: URIs sind der Standard, um Objekte im Internet eindeutig zu identifizieren. URLs wie sie in HTML zur Referenzierung von Webseiten verwendet werden sind besondere URIs.

2.4 Merkmale (Manhart, 2006)

Trotz der Schwierigkeit, die Merkmale von Webservices einheitlich zu beschreiben bzw. zu definieren, was ein Webservice ist, können sich Standardisierungsgremien wie das W3C auf eine Übereinstimmung einigen:

2.4.1 Programmierbarkeit

Webservices können über programmierbare Schnittstellen und unter Verwendung von xml-basierten Nachrichten (Einsatz von SOAP und WSDL) aufgerufen werden.

2.4.2 Protokolltransparenz

Die Operationen und Nachrichten von Webservices können in internetbasierte Protokolle eingebettet werden (HTTP, SMTP).

2.4.3 Kapselung

Unter Webservices versteht man unabhängige, in sich abgeschlossene Applikationen. Sie sind für die Erfüllung einer genau definierten Aufgabe zuständig.

2.4.4 Lose Kopplung

Benutzer von Diensten haben keinerlei Kenntnis über jegliche Implementierungsdetails bzw. dahinterliegende Softwareinfrastruktur.

2.4.5 Komposition

Es ist möglich, dass ein Webservice in weitere Webservices zerlegt wird. Ebenso kann die Zusammenstellung zu einem neuen Webservice aus mehreren, wiederverwendbaren Webservices erfolgen.

2.4.6 Ortstransparenz

Wenn die Benutzer entsprechende Zugriffsrechte besitzen, dann kann die Nutzung von Webservices ortsunabhängig erfolgen.

2.4.7 Selbstbeschreibung

Ein Webservice stellt Metadaten zur Verfügung, welche zur Auswertung verwendet werden können.

2.4.8 Transaktions-Charakter

Webservices können auch in komplexere Geschäftsprozesse integriert werden.

2.5 Vor- und Nachteile (Sotomayor, 2005, S. 6)

Als Vor- und Nachteile können folgende Punkte gegenüber anderen Technologien angeführt werden.

2.5.1 Vorteile

2.5.1.1 Plattform- und programmiersprachenunabhängigkeit

Ein Clientprogramm kann beispielsweise in C++ programmiert sein und unter Windows laufen, während das Webservice selbst in Java programmiert ist und Linux als Betriebssystem hat.

2.5.1.2 Firewall-Resistenz

Da hauptsächlich HTTP zur Übertragung der Webservice-Nachrichten eingesetzt wird, ist es möglich einen ungehindertern Nachrichtenfluss beim Aufbau von internetweiten Applikationen zu gewährleisten. In den meisten Fällen kann nämlich jeglicher HTTP-Traffic die Firewalls störungsfrei passieren.

2.5.2 Nachteile

2.5.2.1 Overhead

Die Übertragung der Daten mittels XML ist nicht so effizient wie wenn dafür ein proprietärer Binär-Code verwendet werden würde. Der Portabilitätsgewinn führt somit zu einem Effizienzverlust.

Für die meisten Applikationen ist solch ein Overhead akzeptabel. Problematisch ist es eher, wenn eine kritische real-time Applikation mittels Webservices implementiert werden soll.

2.5.2.2 Mangelnde Vielseitigkeit

Im Vergleich zu CORBA beispielsweise haben sich Webservices in der Vergangenheit nicht immer durch eine besonders hohe Vielseitigkeit ausgezeichnet. Corba stellt für Programmierer eine Reihe an Services zur Verfügung (Persistence , Notifications ,Lifecycle Management, Transactions).

Mittlerweile muss man jedoch anmerken, dass sich durch das Aufkommen zahlreicher erweiterter Spezifikationen wie beispielsweise WSRF dieser Nachteil verringert hat.

2.6 Webservice Funktionsweise

Um die Funktionsweise der Webservices zu verstehen, soll im folgenden die Clientkomponente, die Serverkomponente und der eigentliche Aufrufprozess betrachtet werden.

2.6.1 Service Requestor (Clientapplikation)

Voraussetzung für einen Webservice-Aufruf ist zunächst, dass der Clientapplikation, welche den Webservice aufruft, die Aufrufmodalitäten des Services bekannt sind. Dazu muss die Applikation in Besitz der zugehörigen WSDL-Beschreibung sein, aus welcher dann nötige Client-Stubs erzeugt werden können.

Für den Fall, dass ein Webservice Consumer erst nach geeigneten Service-Anbietern bzw potentiellen Webservices suchen muss, kann auch ein Verzeichnisdienst (discovery service, registry) genutzt werden. Dieser liefert den Webservice Consumern auf Anfrage die Aufrufmodalitäten zurück, damit dieser dann den Aufruf und die Kommunikation initiieren kann.

Zu erwähnen sind in diesem Zusammenhang die Standards UDDI und WS-Inspection, mit denen diese Auffindungsszenarien umgesetzt werden können.

Aus der WSDL-Spezifikation kann ein Client-Stub generiert werden, mit welchem dann die Webserviceanfragen (SOAP-Requests) durchgeführt werden. Dieser ist auch dafür zuständig, dass die erhaltene Antwort (SOAP-Response) wieder in die internen Formate der aufrufenden Applikation zurückverwandelt wird.

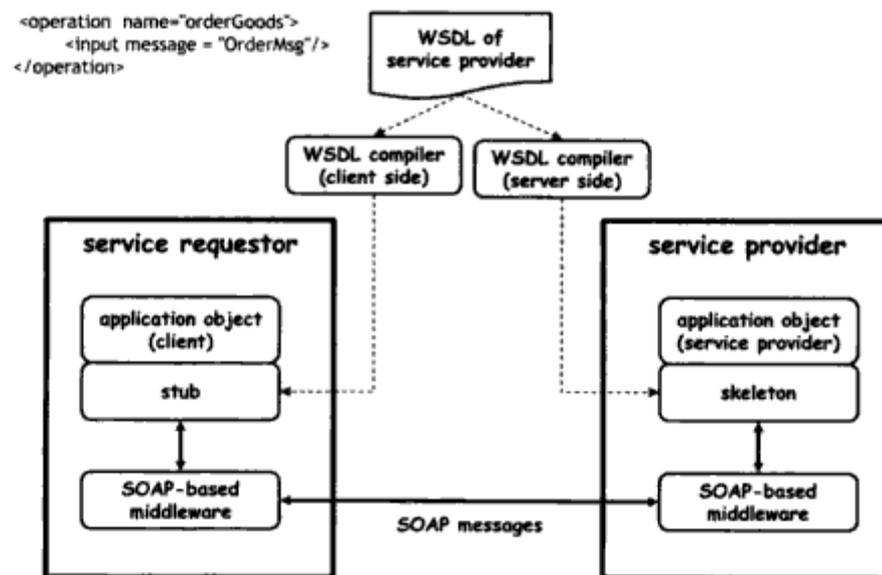


Abbildung 6 Kompilierung von Stubs und Skeletons aus der WSDL-Spezifikation (Alonso, Casati, Harumi, & Machiraju, 2004, S. 154)

2.6.2 Ablauf des Webserviceaufrufes (Sotomayor, 2005, S. 11)

Zum Aufruf des Webservices ist es notwendig, dass die Client-Applikation den Client Stub aufruft. Der Stub wandelt den lokalen Aufruf in einen geeigneten SOAP Request um. Dieser Vorgang wird auch „Marshalling“ oder „Serializing“ genannt. Der SOAP-Request wird über das Netzwerk gesendet. Typischerweise wird dazu das HTTP-Protokoll verwendet. Der Server erhält den SOAP-Request und reicht diesen an den Server Stub weiter. Die Stub-Komponente konvertiert den SOAP Request in eine Art und Weise, welche die Service Implementation versteht. Dieser Vorgang wird auch „Unmarshalling“ oder „Deserializing“ genannt.

Nun ruft der Server Stub die Service-Implementation auf, welche dann die jeweilige Operation ausarbeitet, nach welcher angefragt wurde. Das Resultat der angefragten Operation wird an den Server Stub zurückgereicht, welches diese wieder in eine SOAP-Antwort konvertiert.

Die SOAP-Antwort wird über das Netzwerk (HTTP) an den Client Stub zurückgesendet. Diese wird dann an die eigentliche Client Applikation in einer verständlichen Art und Weise weitergereicht.

Die Applikation kann letztendlich das Ergebnis des Webserviceaufrufes entgegennehmen und diese für weitere Prozesse verwenden.

2.6.3 Service Provider (Ausführungsumgebung) (Sotomayor, 2005, S. 11f)

Um Nachrichten von möglichen Requestoren entgegenzunehmen und abzuarbeiten, muss ein Webservice in eine Laufzeitumgebung integriert werden. Folgende Abbildung veranschaulicht und erklärt wie man sich den serverseitigen Aufbau im Kontext mit Webservices vorzustellen hat.

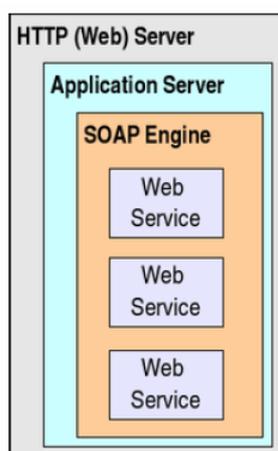


Abbildung 7 Serverseitiger Aufbau eines Webservices (Sotomayor, 2005, S. 12)

Web Server (HTTP Server): Ein Webserver (zB. Apache HTTP Server) hat die Aufgabe, HTTP Nachrichten zu empfangen und zu senden, welche dann an den Applikationsserver weitergereicht werden.

Application Server: Auf einem Applikationsserver werden die Applikationen zur Verfügung gestellt, welche von verschiedenen Clients aufgerufen werden können. Ein Beispiel für diese Komponente ist der Jakarta Tomcat Server. Die SOAP-Engine Komponente läuft hierbei als Applikation innerhalb des Applikationsservers.

SOAP Engine: Die Soap-Engine hat die Aufgabe, SOAP Requests zu interpretieren und entsprechende SOAP-Antworten zu erzeugen. In der Praxis ist es üblich eine SOAP-Engine zu verwenden, anstatt für jedes einzelne Webservice einen eigenen Server-Stub zu erzeugen. Eine SOAP-Engine fungiert somit quasi als sogenannter universeller Stub. Ein Beispiel für diese Komponente ist Apache Axis.

Web Service: Die eigentliche Webservicekomponente ist das Stück Software, welches eine Reihe von Operationen zur Verfügung stellt,

welche dann von Clientprogrammen aufgerufen werden können. Ein Webservice kann beispielsweise in Java implementiert sein.

2.7 Web Service Architektur

Die Web Service Architektur schließt viele Technologien ein, die sich auf unterschiedlichen Schichten befinden und untereinander in Beziehung stehen.

Die beteiligten Komponenten sind nach dem Baukastenprinzip aufgebaut. Dabei werden die darunterliegenden Technologien von den darüber liegenden Komponenten verborgen. Die folgende Abbildung veranschaulicht eine grobe Schichten-Darstellung der Webservicekomponenten.

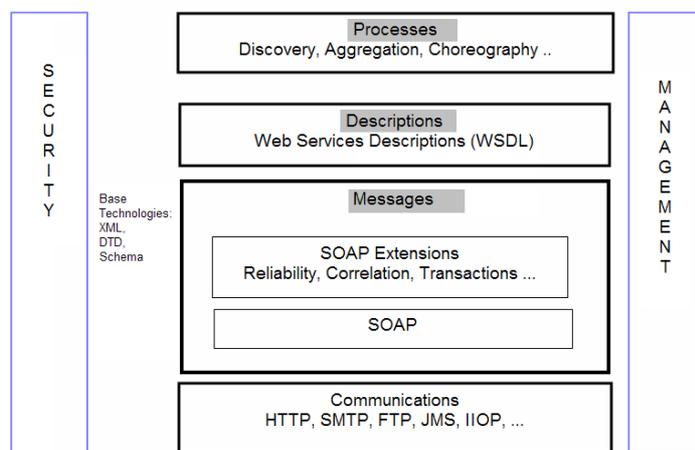


Abbildung 8 Web Services Architektur (Booth, et al., Web Services Architecture, 2004)

Die Anordnung stellt nur einen Stapel unter vielen möglichen dar. Dies liegt daran, dass es aufgrund verschiedener Interessen und Betrachtungsweisen eine Vielzahl von möglichen Varianten gibt.

Um ein besseres Verständnis über die Fülle der Standards zu bekommen, sind nachfolgend die grundlegenden Webservicebereiche und ihre zugehörigen wesentlichen Spezifikationen angeführt (Tilkov & Roth, 2006):

- **XML**
 - Namespaces
 - XML Schema
- **Message Specifications**
 - SOAP – Protokoll zur Nachrichtenübertragung
 - WS – Adressing

- WS – Notification
- WS Based Notifications
- WS Topics
- WS – Eventing
- **Meta Data Standards**
 - WSDL - Definition und Beschreibung der Schnittstellen
 - UDDI - Verzeichnisdienst zum Auffinden von Webservices
 - WS-Policy
 - WS-Discovery
- **Attachments**
 - WS-Attachments
- **ReliableMessaging**
 - WS-ReliableMessaging
 - WS-Reliability
- **Transactions and Coordination**
 - WS-Coordination
 - WS-Atomic Transaction
 - BusinessActivity
- **Business Processes**
 - BPEL for Web Services
- **Security**
 - WS-Security: SOAP Message Security
 - WS-Security: Kerberos Binding
 - WS-Security: SAML Token Profile
 - WS-Security: X.509 Certificate Token Profile
 - WS-Security: Username Token Profile
- **Ressources**
 - Web Service Resource Framework
 - WS-BaseDefaults
 - WS-ServiceGroup
 - WS-ResourceProperties
 - WS-ResourceLifetime
- **Management**
 - WS-Events
 - WS-Management

- Web Services Management Using Web Services
- Web Services Management Of Web Services

2.8 Basisstandards

2.8.1 SOAP

SOAP (ursprünglich für Simple Object Access Protocol) ist ein Standard, welcher allgemeine Regeln definiert, um den Nachrichtenaustausch zwischen den beteiligten Komponenten durchführen zu können (dh. die Verbindung zwischen dem Servicerequestor und dem Webservice). Das Protokoll gewährleistet, dass die Aufrufe an die Webmethoden des Services, sowie die dazugehörigen Antworten übertragen werden. Die SOAP-Nachrichten werden hierzu in ein Transportprotokoll eingebettet. In den meisten Fällen handelt es sich dabei um HTTP.

Das vom W3C freigegebene Verpackungsprotokoll SOAP ist in der sogenannten Messaging-Ebene angesiedelt. Diese Ebene ist für den Austausch der Nachrichten und Metadaten zuständig. Mittels SOAP werden die Nachrichten und die Nutzlast in einem Umschlag (Envelope) gesteckt. Die Metadaten informieren dabei über Inhalt, Absender und Adressat. Die Metadaten können erweitert werden. Dadurch können sie für erweiterte Webservice-Standards wie Sicherheit und Adressierung genutzt werden. (Heinz, 2005)

2.8.2 WSDL

WSDL ist jener Standard, mit welchem die Webservice-Schnittstellen detailliert beschrieben werden.

WSDL gehört zur Beschreibungsschicht. In dieser Schicht werden die Services und deren Eigenschaften beschrieben, welche eine Webservice-Consumer-Einheit benötigt, um einen Aufruf durchzuführen.

Mittels WSDL werden die Fähigkeiten des Webservices in Form einer Ansammlung von Operationen definiert, die von anderen verteilten Komponenten (Service Requestors) aufgerufen werden können. Die Beschreibung einer jeden Operation erfolgt durch die Angabe von Nachrichtenmuster (Message Exchanges). Die Definition betrifft sowohl das

Nachrichtenformat für den Operationenaufruf, als auch das Nachrichtenformat möglicher Antwortnachrichten. Zusätzlich werden auch die Fehlermeldungen angegeben (Foster, et al., 2004b).

Mit WSDL können dann clientseitige Webservice-Tools automatisch Programmkomponenten erzeugen, mit welchen dann die Dienste aufgerufen und genutzt werden können.

2.8.3 UDDI

Webservices können von anderen Komponenten unter Verwendung des Verzeichnisdienstes UDDI ((Universal Description, Discovery and Integration) gefunden und aufgerufen werden.

Es handelt sich dabei um einen Industrievorschlag für verteilte "Dienstekataloge", welche als Web Services zur Verfügung gestellt werden. Bereitgestellt wird ein Verzeichnis von Adress- und Produktdaten sowie Anwendungs-Schnittstellen der verschiedenen Web Services-Provider (Parys & Mauerer, 2004).

Generell besteht keine unbedingte Notwendigkeit, dass UDDI für den Einsatz von Webservices verwendet werden muss. Verzeichnisdienste wie UDDI gehören jedoch zu den unbedingten Infrastrukturkomponenten einer SOA.

2.9 Erweiterte Standards

Es gibt natürlich Anforderungen, welche die grundlegenden Spezifikationen SOAP, WSDL und UDDI nicht ausreichend erfüllen können. Zu den erweiterten Anforderungen gehören u.a die Bereiche Sicherheit, Zuverlässigkeit und Transaktionen. Diese sind auch unter dem Begriff „Quality of Services“ (QoS) bekannt. Zahlreiche Spezifikationen bezüglich QoS werden von den Standardisierungsgremien W3C und OASIS ausgearbeitet. (Kodali, 2005)

Aufgrund der großen Vielzahl an unterschiedlichen Standards ist es im Rahmen dieser Arbeit verständlicherweise nicht möglich, explizit die Gesamtheit aller einzelnen Entwicklungen näher zu erläutern.

Als Beispiel für mögliche Anwendungsgebiete können folgende Anforderungen genannt werden:

2.9.1 Zuverlässigkeit

Der Standard WS-ReliableMessaging dient dazu, dass eine Zuverlässigkeit des Nachrichtenaustausches auch auf nicht zuverlässigen Transportinfrastrukturen wie zb. HTTP erreicht wird. Das Protokoll stellt sicher, dass eine korrekte Reihenfolge und keine unnötige Wiederholung bei der Nachrichtenübermittlung gewährleistet sind (Heinz, 2005).

2.9.2 Transaktionen und Operationen

Es sind Standards wichtig, um Operationen abzusichern und die Koordination von verschiedenen Partner innerhalb einer Transaktion zu bewerkstelligen. Dies wird gewährleistet durch eine Reihe von Spezifikationen wie WS-Coordination, WS-Atomic Transaction und BusinessActivity (Heinz, 2005).

2.9.3 Buisness Prozesse

Der Standard BPEL4WS (Business Process Execution Language for WS) wird angewendet um Geschäftsprozesse zu betrachten und diese zwischen Geschäftspartnern zu integrieren. Geschäftsprozessabläufe können über Anwendungs-, Partner und Firmengrenzen hinweg formal definiert werden. Dies ist die Grundlage für die Prozessimplementierung, um die Prozesse in Werkzeuge und in EAI-Plattformen importieren zu können (Heinz, 2005).

2.9.4 Sicherheit

Die Authentifizierung, Signierung und Verschlüsselung von Nachrichten wird beispielsweise mit dem Standard WS Security erreicht.

2.9.5 Ressourcen

In diesem Anwendungsbereich werden Ansätze definiert , um Webservices mit zustandsgebundenen Operationen zu erweitern und somit eine Verwaltung von zustandsbehafteten Ressourcen zu ermöglichen. Die besondere Eigenschaft von Standards wie WSRF und WSN liegt darin, dass diese ebenso in den Bereichen des Grid Computing eingesetzt werden können und somit eine Verbindung zwischen der Webservice- und der Gridcommunity schaffen.

2.10 Serviceorientierte Architektur

Ein empfehlenswerter Lösungsansatz ist es, IT-Systeme mit dem Design einer sogenannten Serviceorientierten Architektur (SOA) auszustatten.

Einer SOA liegt die Idee des sogenannten „Composite Computing Model“ zugrunde. Bei diesem Modell handelt es sich um eine Architektur, welche eine verteilte, Ausführungsumgebung nutzt, um eine Reihe von serviceorientierten Softwarekomponenten² nach außen verfügbar zu machen und zu verwalten.

Die Anforderungen des Composite Computing Modells sind (Chappel & Jewell, 2003, S. 13):

- Dynamische Suche und Entdeckung der Leistungen der Anwendungslogik
- Trennung der Anwendungslogik-Beschreibung von deren Implementierung
- Aufbaumöglichkeit von IT-System-Verbänden mittels minimalen Planungsaufbau, Installationsaufwand und manuellen Eingriffen

Obwohl die Begriffe SOA und Webservices oft in Kombination miteinander genannt werden, handelt es sich um zwei verschiedene Dinge. In technischer Hinsicht, bezieht sich der Begriff SOA auf das Design eines Systems und nicht auf seine Implementation. Im Unterschied dazu handelt es sich bei Webservices hingegen ganz konkret um die technische Implementierungsart einer SOA. (Srinivasan & Treadwell, 2005, S. 2)

Die Kombination von SOA und Webservices wird klar der Fokus der Entwicklung für die vorhersehbare Zukunft sein, aber es muss erkannt werden, dass es noch immer eine Anzahl an einigen beherrschenden Themen gibt, die bewältigt werden müssen. Dazu gehören: noch immer unreife Technologien, Ablehnung und Verwirrung im Bereich der Standards, und einen signifikanten Performance Overhead beim Serializing, Deserializing und Parsing von SOAP

² Unter einer Softwarekomponente versteht man einen Teil der Anwendungslogik, die eine nützliche Funktion anbietet und nach außen hin verfügbar gemacht wird.

Messages und XML Dokumenten im Zuge einer jeden Message Exchange (Srinivasan & Treadwell, 2005, S. 2).

Der weiter fortschreitende Standardisierungsprozess soll letztlich bewirken, dass unter Einsatz von SOAs bzw. Webservices eine größtmögliche Interoperabilität erreicht wird und die IT-Systeme Vorteile in Bezug auf Kosten und Flexibilität aufweisen.

2.10.1 SOA - Konzept

In einer allgemeinen Darstellung kann man eine SOA als einen Architekturstil beschreiben, in welchem die Systeme aus ServiceUsern und ServiceProvidern bestehen .

Wie schon der Name vermuten lässt, ist beim SOA-Paradigma das Service das zentrale Thema. Eine Geschäftslogik oder die individuellen Funktionen einer Applikation werden modularisiert und als Services für Konsumenten bzw. Client Applikationen dargestellt. Schlüsselkriterium all der Services ist die Natur der losen Kopplung, dh. das Service Interface ist unabhängig von der Implementation.

Anwendungsentwickler können Applikationen zusammenbauen indem sie mehrere Services zusammensetzen, ohne die darunterliegende Implementation der Services zu kennen. Beispielsweise kann ein Service entweder in .NET oder J2EE implementiert werden. Die Applikationen die dann das Service konsumieren, können auf unterschiedlichen Plattformen oder Sprachen laufen (Kodali, 2005).

„Service User“ und „Service Provider“ sind die grundlegenden Komponenten in einer SOA. Zusätzliche Komponenten, so wie beispielsweise der Enterprise Service Bus (EBS) und das Dienstverzeichnis, können ebenso genutzt werden. SOA-Verbindungsarten umfassen synchrone und asynchrone Aufrufe. In den meisten Fällen erfolgt dies durch Verwendung von SOAP, HTTP und einer Nachrichten-Infrastruktur (Bianco, Kotermanski, & Merson, 2007).

2.10.2 Definition – SOA

Für den SOA-Begriff existieren zahlreiche Definitionen. So wie beim Webservices-Begriff, stellt auch hier wieder das OASIS-Konsortium eine

allgemein gültige Definition bereit (MacKenzie, Laskey, McCabe, Brown, & Metz, 2006):

- Eine SOA beschreibt eine Methode für den Entwurf und das Implementieren von im Unternehmen eingesetzte Applikationen, welche sich mit der Vernetzung von lose gekoppelten, grobkörnigen, wiederverwendbaren Artefakten (Services) befasst.
- SOA bezeichnet ein Paradigma für die Organisation und den Gebrauch von verteilten Ressourcen, welche sich unter der Kontrolle von verschiedenen Eigentums-Zuständigkeitsbereichen befinden.
- Eine SOA stellt einheitliche Mittel für das Anbieten, Auffinden und Interagieren mit Ressourcen bereit und verwendet diese Ressourcen, um gewünschte Effekte zu erzeugen, die mit messbaren Vorbedingungen und Erwartungen konsistent sind.

2.10.3 Definition – Service

Auch für den Begriff des Services stellt OASIS folgende Definition bereit (MacKenzie, Laskey, McCabe, Brown, & Metz, 2006):

Ein Service bezeichnet einen Mechanismus, um Zugang zu einer oder mehreren Ressourcen zu ermöglichen, wobei der Zugang unter Verwendung einer vorgeschriebenen Schnittstelle bereitgestellt wird und dabei konsistent mit den Beschränkungen und Richtlinien – so wie es durch die Servicebeschreibung spezifiziert wird - ausgeführt wird.

- Ein Service wird durch eine Entität zur Verfügung gestellt – dem Service Provider. Die Konsumenten dieses Services müssen dem Service Provider nicht bekannt sein. Dabei kann das Service genutzt werden, indem es über den ursprünglich vom Service-Provider erdachten Anwendungsbereich hinausgeht.
- Auf das Service wird durch die Möglichkeiten eines Service Interfaces zugegriffen, wobei das Interface die Spezifika beinhaltet, auf welche Art und Weise auf die darunterliegenden Ressourcen zugegriffen werden kann.
- Ein Service ist undurchsichtig – Seine Implementation wird vor dem Service Konsumenten versteckt.

2.10.4 SOA – Szenario

Die folgende Abbildung veranschaulicht einen einfachen Service-Interaktions-Zyklus, so wie es auch häufig im Kontext bezüglich des Zusammenspiels der Webservicestandards SOAP, WSDL und UDDI veranschaulicht wird.

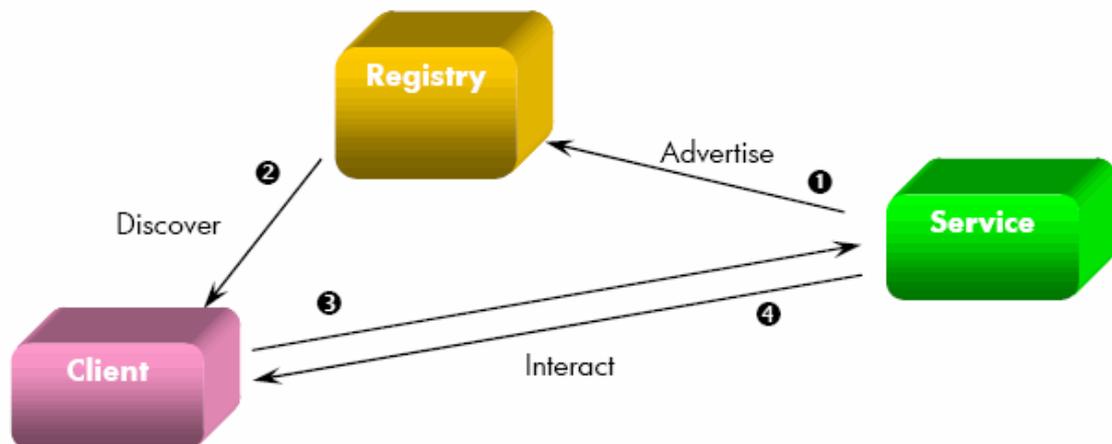


Abbildung 9 Zusammenspiel von SOAP, WSDL und UDDI (Srinivasan & Treadwell, 2005, S. 2)

Der Kreislauf beginnt mit einer Service Ausschreibung durch ein bekanntes Registry Service . Ein potentieller Client stellt eine Anfrage an die Registry , um nach einem Service zu suchen, welche seinen Anforderungen gerecht wird. Die Registry gibt eine Liste an die passenden Services zurück, und der Client wählt eines aus und reicht eine Request Message zu diesem weiter. Dabei verwendet er ein gemeinsames Protokoll . Das Service antwortet in diesem Beispiel entweder mit einem Resultat von der angefragten Operation oder mit einer Fault Message.

Während Abbildung 9 einen einfachen, synchronen, bidirektionalen Message Exchange Pattern zeigt, sind generell eine Vielfalt an Patterns möglich. Eine Interaktion kann beispielsweise in nur eine Richtung erfolgen.

Es kann auch sein, dass die Response nicht von dem Service zurückgeliefert wird, zu welchem der Client den Request gesendet hat, sondern von einem anderen Service, welche die Transaktion zu Ende geführt hat (Srinivasan & Treadwell, 2005, S. 2).

3 SOAP

3.1 Grundkonzept

Die vom W3C freigegebene Protokoll-standard SOAP ist ein wichtiger Baustein beim Ablauf einer Webservice Interaktion. Mit SOAP ist es möglich, Nachrichten zu spezifizieren, damit eine Kommunikation zwischen einem Service-Anbieter und einem Service-Konsumenten hergestellt werden kann. Des weiteren stellt es den Applikationen Wege zur Verfügung, wie gewisse Aspekte einer Nachricht (beispielsweise die Header-Elemente) behandelt werden sollen.

Dadurch ist es möglich Applikationen zu erstellen, in denen eine Nachricht mehrere Zwischenknoten passiert, bevor sie den endgültigen Empfänger erreicht (Chase, 2007, S. 7).

Die veröffentlichten Spezifikationen sind als Dokument unter <http://www.w3.org/TR/soap12> zu finden.

Die Bezeichnung SOAP stand zunächst für Simple Object Access Protocol, steht aber nun seit der Version 1.2 für sich selbst (Hauser & Löwer, 2004, S. 40).

SOAP stellt ein erweiterbares Nachrichten-Rahmenwerk (Messaging Framework) zur Verfügung, damit ein Nachrichten-Konstrukt über eine Reihe von zugrunde liegenden Protokollen ausgetauscht werden kann (Gudgin, et al., SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007).

Durch den Einsatz dieses Konstrukts ist es möglich, xml-basierte strukturierte Information zwischen Teilnehmern in einer dezentralisierten, verteilten Umgebung auszutauschen. Man spricht in diesem Zusammenhang vom Austausch sogenannten SOAP-Nachrichten (SOAP-Messages).

SOAP Nachrichten bestehen aus dem SOAP Envelope, dem SOAP Header und dem SOAP Body. Der Envelope ist sozusagen der Umschlag. Der Header und der Body sind dabei Teil des Envelopes. Die SOAP Message wird über ein Transportprotokoll versendet und deshalb sind die drei Teile wiederum in einem Transportprotokoll eingekapselt (siehe Kapitel 3.5 Nachrichtenstruktur).

Die nachfolgende Abbildung 10 veranschaulicht diese Tatsache.

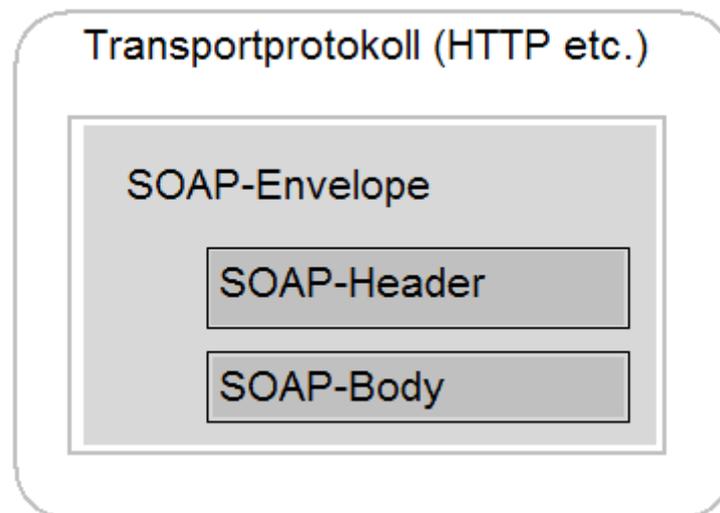


Abbildung 10 Aufbau einer SOAP-Nachricht (Hauser & Löwer, 2004, S. 47)

Die grundsätzliche Idee von SOAP ist es, Objekte, welche auf einem Server liegen, auf eine XML-Struktur abzubilden. Nachdem dieses XML-Konstrukt mittels SOAP übertragen worden ist, kann dieses auf einem Client dann wieder in Objekte und in Funktionen umgewandelt werden.

In der SOAP-Spezifikation ist u.a festgelegt (Gudgin, et al., SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007):

- wie die Struktur einer Nachricht definiert ist (Message Construct)
- nach welchen Regeln die Nachricht abgearbeitet wird (Processing Model)
- welche Erweiterungen (Reliability, Security, etc.) eingerichtet werden können (Exentsibility Model)
- welche darunterliegenden Protokolle zum Austausch von SOAP-Nachrichten zwischen den SOAP-Knoten verwendet werden können (Protocol Binding).

Das Framework wurde entworfen, um eine Unabhängigkeit von jeglichen Programmiermodellen und anderen Implementations spezifischen Semantiken zu erreichen.

Einfachheit und Erweiterbarkeit sind die zwei grossen Designziele von SOAP. Diese Ziele werden erreicht, indem auf zahlreiche Einrichtungen, wie Reliability, Security, Correlation und Routing verzichtet wird, jedoch können diese als

Erweiterung durch andere Spezifikationen definiert werden (Gudgin, et al., SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007).

Von den SOAP-Machern ist bewusst gewollt, dass das SOAP-Protokoll nicht selbst alle Funktionalitäten bereitstellen kann. Aus diesem Grund wurde beispielsweise auf Sicherheitsmechanismen und Transaktionssteuerung verzichtet. Die Header sind aber offen angelegt, sodass SOAP erweitert werden kann. Diese Tatsache verhindert, dass das SOAP-Protokoll zu komplex werden würde. (Hauser & Löwer, 2004, S. 53)

3.2 Art des Nachrichtenaustausches

Grundsätzlich handelt es sich bei SOAP um ein Paradigma, bei welchem die Übertragung „one-way“ erfolgt.



Abbildung 11 Einfache SOAP-Übertragung (Skonnard, 2003)

Für den Nachrichtenaustausch zwischen den Knoten erlaubt SOAP aber auch noch weitere Vorlagen wie beispielsweise Request/Response.

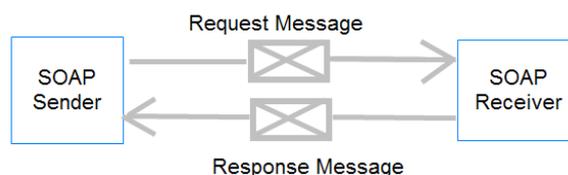


Abbildung 12 SOAP-Übertragung mit Request/Response (Skonnard, 2003)

Dieses Austauschmuster wird Message Exchange Pattern (MEP) genannt. Im SOAP-Erweiterungsmodell sind diese MEPs definiert (Gudgin, et al., SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007).

Nachrichten können als einfache Nachrichten in eine Richtung versandt werden. Nachrichten können aber auch aus zwei Nachrichten bestehen, aus Anfrage und Antwort, ähnlich einem RPC (Remote Procedure Call). Für RPC gibt es in SOAP 1.2 einen eigenen Abschnitt, welcher regelt wie RPC über SOAP auszusehen hat (Gudgin, et al., 2007a)

Die SOAP-Spezifikation regelt außerdem, wie eine Implementierung ein eigenes MEP einbinden kann.

Denkbar ist aber auch der Nachrichtenaustausch auf Basis von mehreren Netzwerkverbindungen analog zu einer Peer-to-peer-Architektur. Eine Nachricht kann hier mehrere Empfänger haben, und die Rückgaben können beispielsweise gebündelt werden. Allerdings sind RPCs und reine Nachrichten die in der Praxis hauptsächlich eingesetzten Varianten (Hauser, 2004, S. 43).

3.3 Nachrichtenpfad und Nachrichtenknoten

Der Pfad, den eine Nachricht zwischen einem initialen SOAP-Sender und einem ultimate SOAP-Empfänger einnimmt, wird „Message Path“ genannt und kann auch über sogenannte SOAP-Zwischenknoten führen. Zwischenknoten agieren wie eigene SOAP-Anwendungen und können auf die Nachricht mitwirken. Es ist aber auch eine direkte Sendung möglich ohne jegliche Zwischenknoten. (Hauser, 2004, S. 44)

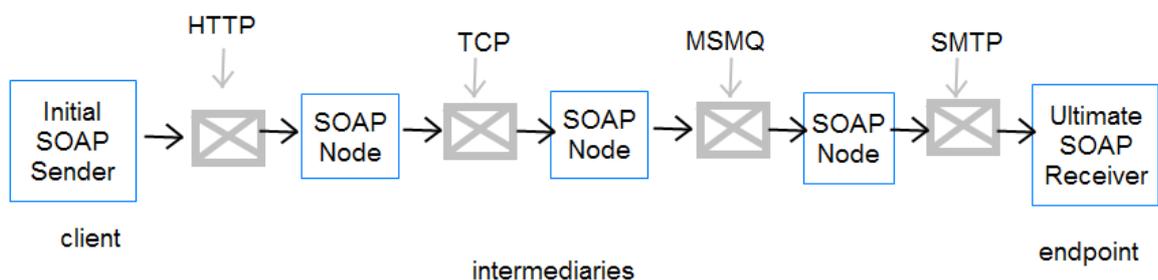


Abbildung 13 SOAP-Nachrichtenübertragung mit Zwischenknoten (Skonnard, 2003)

Man unterscheidet folgende SOAP-Knoten Typen:

- **SOAP-Sender:** jener Knoten, welcher die Nachricht an einen anderen Knoten versendet
- **SOAP-Receiver:** jener Knoten, welcher die Nachricht von den Sender erhält und diese auch verarbeiten muss. Man bezeichnet solch einen Empfänger auch als „ultimate“ Receiver, wenn die Nachricht genau an diesen adressiert wurde. Dieser Knoten erhält die Nachricht als Letzter und schickt sie nicht mehr weiter, sondern verarbeitet die Nachricht.
- **SOAP-Intermediary:** sind jene Knoten, welche sich am sogenannten „messagepath“ zwischen dem initialen und dem ultimativen Knoten befinden. Ein solch ein Zwischenknoten ist dabei auch gleichzeitig ein

SOAP-Sender und SOAP-Empfänger, da er sie entsprechend dem SOAP Verarbeitungsmodell verarbeiten muss.

Im Gegensatz zum Endknoten erfolgt bei diesen Zwischenknoten keine direkte Adressierung. Es kann Einfluss auf die Bearbeitung der Nachricht genommen werden (nicht jedoch auf die Beantwortung). Dh. Teile daraus können genutzt werden. (Graham, et al., 2004, S. 127)

SOAP Zwischenknoten können also zusätzliche Services bereitstellen und werden eingesetzt um beispielsweise folgende Aufgaben zu übernehmen (Graham, et al., 2004, S. 128):

- **Absicherung des Nachrichtenaustausches:** Zum Zwecke der Verschlüsselung und Digitalen Signierung werden zuerst Zwischenknoten passiert.
- **Beglaubigung:** Eine dritte Instanz speichert die Interaktion – die gespeicherte Referenz einer Transaktion kann von den Teilnehmern für zukünftige Zwecke verwendet werden.
- **Nachverfolgen:** Speicherung des Nachrichtenpfades inklusive detaillierte Zeitpunkte der Ankünfte. Dadurch können Engpässe identifiziert werden, welche für das Messen von Quality of Service relevant sein kann.

Den Zwischenknoten ist es erlaubt, Header-Teile entlang des Nachrichtenpfades zu überwachen, einzufügen, zu löschen oder weiterzuleiten. (Nilo & Lafon, 2007)

Bei der Abarbeitung einer SOAP-Nachricht kann ein SOAP-Knoten eine oder mehrere sogenannte SOAP-Rollen einnehmen, welche durch eine URI identifiziert wird. (Gudgin, et al., 2007)

3.4 Nachrichtenverarbeitungsmodell

Durch das SOAP-Verarbeitungsmodell wird beschrieben, wie eine SOAP-Nachricht von einem SOAP-Prozessor bei dem Erhalt der Nachricht verarbeitet wird bzw. welche Maßnahme getroffen werden müssen.

Zu Beginn wird zunächst untersucht ob die Nachricht eine syntaktisch korrekte Form aufweist. Die Elemente (envelope, header, body) müssen gemäß dem SOAP-Namensraum konform sein. Als ersten Schritt bei der Bearbeitung einer

Nachricht ist zunächst jeder Knoten angewiesen alle Header-Elemente zu sammeln, die an diesen Knoten gerichtet sind. Danach werden die Headerelemente durchgesehen, die das Attribut `<mustUnderstand>` gesetzt haben. Dadurch ist es dem Knoten möglich zu bestätigen, dass er diese Elemente gemäß der Regeln welche in diesem SOAP Header angeben sind, bearbeiten kann. (Graham, et al., 2004, S. 129)

Wenn ein SOAP Knoten erfolgreich einen Header abgearbeitet hat, dann ist er angewiesen den Header von der Nachricht zu entfernen. (Skonnard, 2003)

Das eigentliche Body-Element der SOAP-Nachricht ist schließlich für den ultimativen SOAP-Receiver bestimmt. Die darin enthaltenen Informationen können letztlich von der jeweiligen Anwendung interpretiert und verarbeitet werden.

Es ist abhängig von der Anwendung, ob weitere Nachrichten an andere SOAP-Knoten verschickt werden. Der SOAP-Prozessor erzeugt eine Fehler-Nachricht, wenn die Verarbeitung der Nachricht zu Fehlern geführt hat.

3.5 Nachrichtenstruktur

SOAP-Nachrichten bestehen immer aus drei Teilen:

- dem SOAP Envelope ,
- dem SOAP Header,
- dem SOAP Body

3.5.1 Envelope

Der Envelope (zu deutsch: Umschlag) ist der Umschlag einer SOAP-Nachricht und zugleich Wurzelement und beinhaltet die Kinderelemente Header und Body. Der SOAP-Envelope beinhaltet Angaben zur Deklaration des Namensraumes und kann darüberhinaus SOAP-Encoding-spezifische Angaben enthalten (siehe nachfolgendes Kapitel).

3.5.1.1 Deklaration des Nachrichtenraumes

Die Deklaration des SOAP-Nachrichtenaufbaus muss durch Angabe des zugehörigen XML-Namensraumes (bspw. <http://www.w3.org/2003/05/soap-envelope>) definiert werden.

Wie im folgenden Listing ersichtlich wird durch den Namensraum-Bezeichner `<env>` für alle Elemente in dem XML-Dokument festgelegt, dass es sich hierbei um eine SOAP-Spezifikation handelt.

Tabelle 1 Beispiel für SOAP-Envelope (Chase, 2007, S. 44)

```
<?xml version='1.0' ?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Header>
  </env:Header>
<env:Body>
</env:Body>
</env:Envelope>
```

Es handelt sich bei der angegebenen URI um das XML-Schema, welches den Aufbau und den Regelsatz vorgibt, nach welchen sich die SOAP-Nachricht richten muss. Es ist hierin beispielsweise definiert, dass die äusseren Tags in einer SOAP-Nachricht Envelope heissen müssen.

3.5.2 Header

Der SOAP-Header ist ein Erweiterungsmechanismus, um Informationen innerhalb einer SOAP-Nachricht zu transportieren, die nichts mit der eigentlichen Nutzlast einer Anwendung zu tun haben. (Nilo & Lafon, 2007)

Die hierin angegebenen auf SOAP basierende Erweiterungen sind u.a. die Spezifikationen WS-Reliability (Sicherheitsmechanismus zur Abwicklung von Transaktionen) oder WS-Security (zur Sicherstellung von vertraulichen Nachrichten).

Der Header ist vorgesehen um für den Aufruf zusätzliche Metadaten anzugeben. Der Header fungiert quasi als zusätzlicher Behälter um Informationen wie beispielsweise Authentifizierungsdaten (Login, Benutzername, Passwort) oder Routing-Informationen mitzuliefern. Es kann beispielsweise eine Nachrichtennummer oder Hinweis angegeben werden dass die Nachricht eine Antwort auf eine andere Nachricht ist. (Kossmann & Leymann, 2004)

Innerhalb des Headers einer SOAP-Nachricht können weitere sogenannte Header-Blöcke angegeben werden. Vor allem Intermediaries (SOAP-Zwischenknoten), die sich entlang des Nachrichtenpfades befinden, nutzen das Header-Element und können somit auf die Bearbeitung der Nachricht Einfluss

nehmen. Der Header kann eine beliebige Anzahl an Elementen beinhalten, welche einem anderen Namensraum angehören können (anders als der SOAP Namespace). (Skonnard, 2003)

3.5.2.1 Attribut `<mustUnderstand>`

Das Attribut `<mustUnderstand>` wird eingesetzt, um dem Empfänger der SOAP-Nachricht mitzuteilen, dass das zugehörige Header-Element auf jeden Fall vor der Abarbeitung der Nachricht verstanden werden muss.

Ein `<mustUnderstand>` mit dem Wert `<true>` bedeutet, dass der SOAP-Knoten den Header-Block gemäß der über die Header-Spezifikation beschriebenen Semantik verarbeiten muss, da ansonsten eine SOAP-Fehlernachricht generiert wird und die Verarbeitung gestoppt wird. (Nilo & Lafon, 2007)

3.5.2.2 Attribut `<role>`

Es wurde bereits erwähnt, dass den Header-Blöcken sogenannte Rollen zugeordnet werden. Von diesen Rollen ist es dann abhängig, wie die Verarbeitung erfolgt und ob der Header bedient werden muss oder nicht.

Die Rolle wird in den Headerblöcken durch Einsatz des Attributs `<role>` angegeben. Man unterscheidet folgende Rollen-Bezeichnungen:

- `<next>` (<http://www.w3.org/2003/05/soap-envelope/role/next>): Jeder SOAP-Zwischenknoten und auch der ultimative SOAP-Empfänger müssen in dieser Rolle agieren und somit im Stande sein diesen Headerblock zu verarbeiten.
- `<none>` (<http://www.w3.org/2003/05/soap-envelope/role/none>): Diese Rolle darf von dem SOAP-Knoten nicht angenommen werden.
- `<ultimateReceiver>` (<http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver>): Um sich als ultimativer Empfänger auszugeben, muss diese Rolle angenommen werden.

Im nachfolgenden Listing (vgl. Tabelle 2) beschreibt beispielsweise das WS-Adressing Schema die enthaltenen Elemente. Das Listing veranschaulicht zwei im Header enthaltene WS-Adressing Elemente. Als Information ist hierbei

angegeben, wohin die Nachricht gesendet wird und wohin die Antwort erfolgen soll.

Tabelle 2 Beispiel-Listing: Routing Information im Header (Chase, 2007, S. 14)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
<env:Header>
<wsa:ReplyTo xmlns:wsa=
"http://schemas.xmlsoap.org/ws/2004/08/addressing">
<wsa:Address>
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
</wsa:Address>
</wsa:ReplyTo>
<wsa:From>
<wsa:Address>
http://localhost:8080/axis2/services/MyService</wsa:Address>
</wsa:From>
<wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
</env:Header>
<env:Body>
</env:Body>
</env:Envelope>
```

3.5.3 Body

Das Body-Element ist der wichtigste Teil in einer SOAP-Nachricht. Hier erfolgt der Transport der Informationen für die end-to-end-Kommunikation. Der Body beinhaltet die eigentlichen Nutzdaten der SOAP-Nachricht. Hierin sind Nutzinformationen wie beispielsweise eine Verkaufsauftrag oder Reservierungsdaten enthalten.

Das SOAP-Body-Element wird mit seinem lokalen Namen aus dem Envelope Namespace (zB. env:Body) definiert und es können ein oder mehrere Informationsblöcke enthalten sein.

Tabelle 3 Beispiel-Listing: Payload innerhalb des Bodys (Nilo & Lafon, 2007)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
...
</env:Header>
<env:Body>
<p:itinerary
xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
<p:departing>New York</p:departing>
<p:arriving>Los Angeles</p:arriving>
<p:departureDate>2001-12-14</p:departureDate>
<p:departureTime>late afternoon</p:departureTime>
```

```
<p:seatPreference>aisle</p:seatPreference>
</p:departure>
<p:return>
<p:departing>Los Angeles</p:departing>
<p:arriving>New York</p:arriving>
<p:departureDate>2001-12-20</p:departureDate>
<p:departureTime>mid-morning</p:departureTime>
<p:seatPreference/>
</p:return>
</p:itinerary>
<q:lodging
xmlns:q="http://travelcompany.example.org/reservation/hotels">
<q:preference>none</q:preference>
</q:lodging>
</env:Body>
</env:Envelope>
```

3.6 SOAP-Encoding

Die Klassen und Funktionen die bereits in existierenden Anwendungen existieren und über SOAP ansprechbar sein sollen müssen für SOAP aufbereitet werden.

Genauergesagt müssen beispielsweise Objekte dargestellt werden können und Funktionsaufrufe müssen mit Parametern und Rückgabewerten in XML kodiert werden. All diese Konstrukte müssen in SOAP abgebildet werden. Das bedeutet, dass aus Programmierkonstrukten XML wird. Dieser Vorgang heißt auch XML-Serialisierung und man verwendet dazu SOAP-Encoding-Regeln. Die Abbildung betrifft sowohl einfache Datentypen (zB. Strings) als auch komplexe Datenstrukturen (zB.Arrays). Es ist empfehlenswert die Regeln zu verwenden, da dadurch die Kompatibilität gewährleistet wird. (Hauser, 2004, S. 54)

Durch Verwendung des Attributs `<EncodingStyle>` können Serialisierungsregeln für bestimmte Teile einer SOAP-Nachricht angegeben werden. (Gudgin, et al., 2007)

Dieses Attribut kann in den Headern, Bodies, sowie deren Kinderelementen angegeben werden.

3.7 SOAP-Fehlerbehandlung

In einer dezentralisierten verteilten Umgebung, also dort wo auch SOAP im Einsatz ist, können natürlich Fehler (als `<fault>` definiert) bei der

Verarbeitung auftreten. Die SOAP-Spezifikation gibt vor, dass so ein Fehlerereignis auch dem Sender gemeldet werden soll. Aus diesem Grunde wurden in SOAP Fehler-Nachrichten definiert.

Eine SOAP Fehler Nachricht ist eine normale SOAP Nachricht mit dem Element innerhalb des Bodys: `<soapenv:Fault>`. Das Erscheinen dieses Elements signalisiert den Prozessoren dass ein Fehler aufgetreten ist. (Graham, et al., 2004, S. 135)

Mögliche Fehler wären beispielsweise, wenn die ursprüngliche Nachricht nicht der SOAP-Spezifikation genügt oder auch wenn es bei der Nachrichtenabarbeitung ein Fehler auftritt.

3.8 SOAP Binding

SOAP definiert XML-Nachrichtenkonstrukte, welche dann von einem Sender zu einem Empfänger übermittelt werden. Man benötigt dann dazu ein Transportprotokoll, welches die SOAP-Nachricht einkapselt, damit es dann transportiert werden kann. Das Protokoll fungiert quasi als Packesel.

Das Einsetzen und das enge Integrieren eines Transportprotokolls mit SOAP wird auch Binding (dt. Bindung) genannt. (Hauser, 2004, S. 65)

SOAP schreibt kein bestimmtes Protokoll für den Transport vor. Prinzipiell lassen sich SOAP-Nachrichten-Konstrukte mit jedem beliebigen Transportprotokoll verpacken, wie beispielsweise:

- HTTP – Hypertext Transport Protokoll
- FTP – File Transfer Protokoll
- SMTP – Simple Mail Transfer Protokoll

Bei Implementierungen und bei Praxisanwendungen wird meistens das Internet-Transportprotokoll HTTP verwendet.

3.8.1.1 Vorteile von SOAP über HTTP (Dustdar, Gall, & Hauswirth, 2003, S. 119)

1. HTTP ist überall verfügbar und ermöglicht auch Verbindungen über Firewalls.
2. Die Einbindung von SOAP-Nachrichten in HTTP-Aufrufe gestaltet sich einfach.

3. Skalierbarkeit: HTTP-Server sind weit entwickelt und erlauben großen Datenverkehr

4. Die Sicherheitskonzepte für HTTP können auf SOAP-Nachrichten angewendet werden.

4 WSDL

4.1 Allgemeines

Für eine Applikation, die einen Webservice verwendet, ist es notwendig, dass die Programmierschnittstelle genau beschrieben wird.

Die Web Services Description Language (WSDL) definiert eine XML-Sprache, die dazu verwendet wird, um Netzwerkdienste bzw. Webservices zu beschreiben. Der Zweck von WSDL ist es, dass Webservices durch diese Art der Selbstbeschreibung es einem Client auf einfache Art und Weise ermöglichen, die Funktionalitäten des Services zu nutzen, ohne die Implementierungsdetails zu kennen. Als Standard wurde WSDL in der Version 1.0 erstmals im Jahre 2001 vom W3C-Gremium freigegeben.

Im Wesentlichen beschreibt WSDL drei grundlegende Eigenschaften eines Webservices (Graham, et al., 2004, S. 173):

- **WAS** macht ein Service? Dies betrifft die Operationen bzw. Methoden, welche der Dienst zur Verfügung stellt, und die Daten (Argumente und Rückgabeparameter) die benötigt werden, um diesen Dienst aufzurufen.
- **WIE** wird auf das Service zugegriffen? Dies betrifft die Details der Datenformate und Protokolle, die notwendig sind, um auf die Service-Operationen zuzugreifen.
- **WO** befindet sich das Service? Dies betrifft die Details der protokoll-spezifischen Netzwerk-adresse, so wie beispielsweise eine URL.

4.2 Aufbau

Strukturell betrachtet, besteht ein WSDL-Dokument aus einem in XML beschriebenen Schema. In diesem Schema ist eine Grammatik enthalten, mit der Verträge (contracts) für die Kommunikation formuliert werden können. Genaugenommen werden die Schnittstellen von Web-Services beschrieben und nicht die Services selbst. (Dustdar, Gall, & Hauswirth, 2003, S. 121)

Über die Schnittstelle erfolgt die Definition der Operationen des jeweiligen Dienstes. Dazu gehören die Methoden, die Prozeduren oder die Funktionen der zugrundeliegenden Implementierung, sowie deren Aufrufparameter und die

Formate der Input- und Output-Nachrichten. Weiters wird festgelegt, über welche konkreten Transportprotokolle die Bindung erfolgen soll. Die Angabe der Aufrufmodalitäten – wie zB die URI des Dienstes – ist ebenso in der Beschreibung enthalten. Zugang zu den WSDL-Dokumenten erhält man über das Internet. Die Beschreibungen der Dienstanbieter lassen sich entweder direkt einsehen oder man erhält diese indirekt über einen Webservice-Verzeichnisdienst (UDDI).

Die Beschreibung eines Webservices kann auf folgende zwei Teile abgebildet werden: abstrakter und konkreter Teil

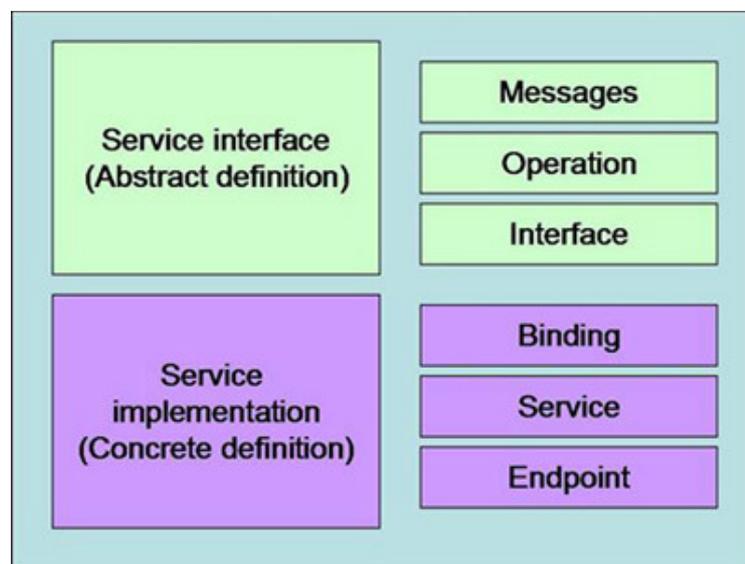


Abbildung 14 Abstrakte und konkrete Beschreibung von Webservices (Dhesiaseelan, 2004)

Der abstrakte Teil kann in Analogie zu herkömmlichen IDLs als eine Schnittstellebeschreibung betrachtet werden. Hierin beschreibt WSDL einen Webservice in Form von Messages, welche typischerweise über ein W3C XML Schema gesendet und empfangen werden. Nachrichtenaustauschmuster (MEPs) definieren die Abfolge und Kardinalität der Nachrichten. Eine Operation assoziiert MEPs mit einer oder mehreren Nachrichten. Ein Interface gruppiert diese Operationen in eine Transport und leitungsunabhängige Art und Weise. (Dhesiaseelan, 2004)

Im konkreten Teil erfolgt die Definition der Aufrufmodalitäten, sowie die Bindung des Dienstes an Transportprotokolle. Ein Service-Endpoint assoziiert die Netzwerkadresse mit einem Binding. Ein Service gruppiert die Endpunkte, welche die Schnittstelle implementieren.

4.3 Elemente

Abbildung 15 gibt einen Überblick über die WSDL-Struktur der im folgenden beschriebenen Elemente `<description>`, `<types>`, `<interface>`, `<binding>` und `<service>` ersichtlich.

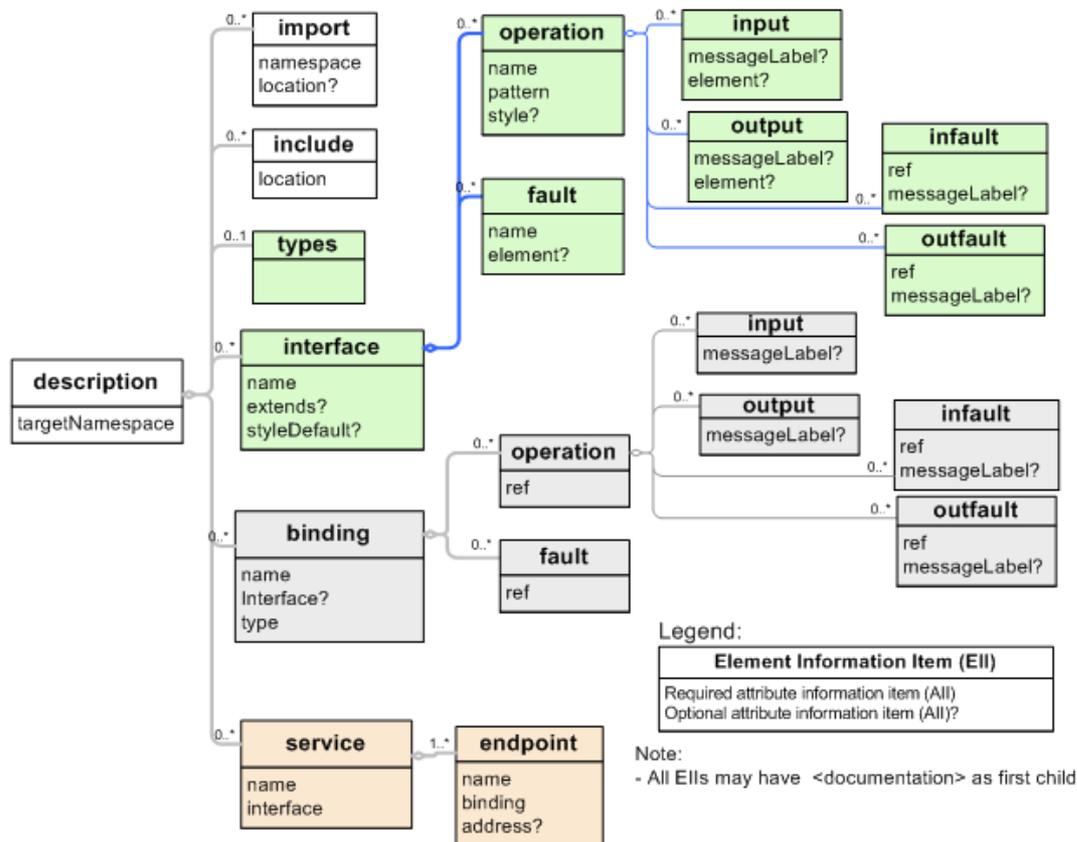


Abbildung 15 WSDL 2.0 Infoset-Diagramm (Nilo & Lafon, 2007)

Um ein besseres Verständnis für die Wirkungsweise der jeweiligen WSDL-Elemente zu erlangen, soll im folgenden ein Beispiel für ein Webservice (Nilo & Lafon, 2007) herangezogen werden.

Die Kernapplikation „Zimmerverfügbarkeit“ eines Hotels ermöglicht einem Reisebüro-Reservierungssystem, direkt über das Internet Zimmer zu buchen. Die angebotene Funktionalität wird wie folgt bezeichnet:

`<CheckAvailability>`

Um die Verfügbarkeit zu überprüfen, muss der Client folgende Angaben tätigen:

- `<Check-in-date>`,
- `<check-out-date>`,

- `<room type>`

Das Webservice liefert eine sogenannte `<room rate>` zurück, sofern ein Raum verfügbar ist (falls nicht, dann wird der `<room rate>` der Wert `<null>` zugewiesen). Wenn irgendwelche Eingabedaten ungültig sind, dann sollte das Webservice einen Fehler zurückgeben.

Das Webservice akzeptiert also eine `<CheckAvailability>`-Nachricht. Als Antwort gibt es eine `<CheckAvailability>`-Antwort oder eine `<invalidFault>`-Nachricht zurück.

Eine weitere Funktionalität des Webservices wäre es eine Reservierung durchzuführen, welche aus Übersichtsgründen in den folgenden Listings nicht zusätzlich angeführt wird.

Jede WSDL-Datei ist ein XML-Konstrukt und beginnt mit folgender Befehlszeile, die in den nachfolgenden Beispiel-Listings nicht extra angeführt wird:

```
<?xml version="1.0" encoding="utf-8" ?>
```

Nachfolgend sind nun die Elementblöcke angegeben, aus welchen sich diese zusammensetzt.

4.3.1 Element `<Description>` (vormals `<definitions>`)

Das Wurzelement der WSDL-Spezifikation, welches alle Bestandteile einschliesst, ist das `<description>`-Element.

Es ist als Gesamt-Container anzusehen, welcher alle notwendigen Informationen über das Webservice und seiner zugehörigen Attribute innehält. (Dhesiaseelan, 2004)

Dementsprechend ist das Element natürlich obligatorisch und kann nur einmal vorkommen. Die darin enthaltenen Elemente können auch mehrfach angegeben sein.

Hierin werden Namensräume deklariert, die dann im gesamten WSDL-Dokument verwendet werden.

Der Standard-WSDL-Namensraum kommt immer dann zum Einsatz wenn ein Element kein Namespace-Präfix besitzt. Dies ist bei den Basiselementen des WSDL-Dokuments der Fall. (Hauser, 2004, S. 81)

Das Attribut `<targetNamespace>` ist Teil des Wurzel-Elements und erfordert die Angabe einer URI. Mit diesem verpflichtenden Attribut wird der Namespace des gewählten Dienstes angegeben.

Tabelle 4 Beispiel-Listing für `<description>` (Nilo & Lafon, 2007)

```
<description
  xmlns="http://www.w3.org/ns/wsd1"
  targetNamespace= "http://greath.example.com/2004/wsd1/resSvc"
  xmlns:tns= "http://greath.example.com/2004/wsd1/resSvc"
  xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  xmlns:wsoap= "http://www.w3.org/ns/wsd1/soap"
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsd1x= "http://www.w3.org/ns/wsd1-extensions">
...
<types> .. </types>
<interface> .. </interface>
<binding> .. </binding>
<service> .. </service>
</description>
```

4.3.2 Element `<Types>`

Das `<types>`-Element ist ein optionales Element in der WSDL-Spezifikation. Innerhalb des WSDL-Dokuments ermöglicht es die Definition von Datentypen, welche beim Nachrichtenaustausch verwendet werden und dient dabei als eine Art Container. Standardmäßig werden bei WSDL die einfachen Datentypen aus der XML-Schema Definition angegeben. Das `<types>`-Element wird verwendet, wenn man ein anderes Datentypensystem als das XML-Schema verwenden will. (Kuschke & Wölfel, 2002)

Tabelle 5 Beispiel-Listing für `<Types>` (Nilo & Lafon, 2007)

```
<types>
  <xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://greath.example.com/2004/schemas/resSvc"
    xmlns="http://greath.example.com/2004/schemas/resSvc">
    <xs:element name="checkAvailability" type="tCheckAvailability"/>
    <xs:complexType name="tCheckAvailability">
      <xs:sequence>
        <xs:element name="checkInDate" type="xs:date"/>
        <xs:element name="checkOutDate" type="xs:date"/>
        <xs:element name="roomType" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="checkAvailabilityResponse" type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
  </xs:schema>
</types>
```

Im oben angeführten Listing sind folgende Nachrichtentypen angegeben:

- `<checkAvailability>`,
- `<checkAvailabilityResponse>` und
- `<invalidDataError>`

4.3.3 Element `<Interface>` (vormals `<portType>`)

Ein WSDL 2.0-Element `<Interface>` definiert die abstrakte Schnittstelle eines Webservices als eine Zusammenstellung von abstrakten Operationen, wobei jede Operation eine einfache Interaktion zwischen einem Client und dem Service darstellt. Die `<Interface>`-Komponente beschreibt die Abfolge von Nachrichten, die im Zuge einer Kommunikation vom Service gesendet und empfangen werden. Die Abfolge einzelner Input- und Output-Nachrichten wird zu einer Operation zusammengefasst. (beispielsweise einer Anfrage und der zugehörigen Antwort). (Nilo & Lafon, 2007)

Tabelle 6 Beispiel-Listing für `<interface>` (Nilo & Lafon, 2007)

```

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault"
    element = "ghns:invalidDataError"/>
  <operation name="opCheckAvailability"
    pattern="http://www.w3.org/ns/wsdl/in-out"
    style="http://www.w3.org/ns/wsdl/style/iri"
    wsdlx:safe = "true">
    <input messageLabel="In"
      element="ghns:checkAvailability" />
    <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
  </operation>
</interface>

```

4.3.3.1 `<operation>`-Element

Das im `<Interface>` enthaltene `<operation>`-Element hat einen Namen (`<name>`) und ein Muster (`<pattern>`) als verpflichtendes Attribut. Das `<style>`-Attribut ist optional.

Im `<operation>`-Element können Unterelemente vom Type `<input>`, `<output>`, `<infault>` oder `<outfault>` enthalten sein. Das Empfangen einer Nachricht wird dabei als `<input>`-Element dargestellt, das Senden als `<output>`-Element. (Kuschke & Wölfel, 2002)

4.3.3.2 <fault>-Element

Dieses Element wird bei einem Fehlerereignis zusätzlich zum <operation>-Element verwendet (<fault name = "invalidDataFault"). Das <name>-Attribut definiert einen Namen für den Fehler. Der Name ist verpflichtend wenn eine Operation definiert wird, damit der gewünschte Fehler anhand seines Namens referenziert werden kann. (Nilo & Lafon, 2007)

Für den Fall eines Fehlers beim Nachrichtenaustausch kann ein <infault>-oder <outfault>-Element innerhalb des <operation>-Elements angegeben werden.

4.3.3.3 Message Exchange Patterns

In der Vorgängerversion WSDL 1.2 wurde noch ein eigenes Element <message> definiert, mit welchem die Nachrichten spezifiziert wurden.

Die Spezifikation WSDL 2.0 wurde nun dahingehend geändert, dass nun insgesamt acht vorgegebene Nachrichtenmuster definiert worden sind. Die Verwendung dieser Muster erfolgt in Abhängigkeit des jeweiligen Dienstes, je nachdem, ob die Sendung beispielsweise uni- oder bidirektional erfolgt.

Das Muster in-out (`pattern=http://www.w3.org/ns/wsd1/in-out`), welches im in der Tabelle 6 angeführten Listing angegeben ist, wird verwendet, wenn ein Dienst eine Nachricht empfängt und eine Antwort sendet. Für den Fall, dass keines der Muster zutrifft, weil beispielsweise eine komplexere Abfolge verwendet wird, ist ebenso die Definition von eigenen Nachrichtenmustern möglich. (Reimers, 2005)

4.3.4 Element <binding>

Die <binding>-Komponente beschreibt ein konkretes Nachrichtenformat und Übertragungsprotokoll, welches verwendet werden kann, um einen <endpoint> zu definieren. (Chinnici, Moreau, Ryman, & Weerawarana, 2007)

Bis jetzt wurde zwar bereits spezifiziert, welche abstrakte Nachrichten ausgetauscht werden können, aber nicht wie dieser Austausch erfolgen kann. Das <binding>-Element definiert, wie die Operationen, die im <interface>-Container angegeben werden, mit dem jeweiligen Protokoll übertragen werden. (Kuschke & Wölfel, 2002)

Jedes `<binding>` in der WSDL-Beschreibung hat eine Referenz zu dem `<interface>`-Container. Die Referenzierung erfolgt durch das Attribut `<interface>`. Alle Operationen, die innerhalb des `<interface>` definiert sind, müssen im `<binding>` eingebunden sein. Ein Endpoint im `<service>`-Container (nähere Erläuterung siehe nachfolgender Abschnitt) wird zu einem Binding referenziert. Beide, nämlich Endpoint und Binding, wurden entworfen, um Flexibilität und Orts-Transparenz zu fördern. Mehrere Endpoints mit verschiedenen Netzwerk-Adressen können somit noch immer das gleiche Protokoll `<binding>` gemeinsam nutzen. Die WSDL 2.0-Spezifikation definiert `<binding>`-Erweiterungen für Protokolle und Nachrichtenformate wie SOAP, HTTP und MIME. (Dhesiaseelan, 2004)

Tabelle 7 Beispiel-Listing für `<binding>` (Nilo & Lafon, 2007)

```

<binding name="reservationSOAPBinding"
        interface="tns:reservationInterface"
        type="http://www.w3.org/ns/wsdl/soap"
wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
  <fault ref="tns:invalidDataFault"
        wsoap:code="soap:Sender"/>
  <operation ref="tns:opCheckAvailability"
        wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>

```

4.3.5 Element `<Service>`

Innerhalb der `<service>`-Deklaration, erfolgt die Konkretisierung der abstrakten Dienstschnittstelle. Diese wird hierin an physische Aufrufmodalitäten gebunden. (Reimers, 2005)

Ein `<service>`-Element beschreibt eine Reihe von Endpoints, welche sich bezüglich eines Bindings auf eine einzelne Netzwerkadresse beziehen (also dort wo die Implementation bereitgestellt wird). All die andere protokoll-spezifische Information ist im `<binding>` enthalten. Der Dienst kann durch den QName referenziert werden. `<name>` und `<interface>` sind die Attribute die im `<service>`-Element angegeben werden. (Dhesiaseelan, 2004)

4.3.5.1 Endpoint-Komponente <endpoint>

Eine <endpoint>-Komponente definiert die Einzelheiten eines spezifischen Endpoints, an welchem ein bestimmtes Service zugänglich ist.

In der WSDL-Version 1.2 wurde statt <endpoint> der Name <port> verwendet.

Tabelle 8 Beispiel-Listing für <service> (Nilo & Lafon, 2007)

```
<service name="reservationService"
  interface="tns:reservationInterface">

  <endpoint name="reservationEndpoint"
    binding="tns:reservationSOAPBinding"
    address ="http://greath.example.com/2004/reservation"/>
</service>
```

4.3.6 zusätzliches Element <documentation>

Dieses Element ist als zusätzliches Dienstbeschreibungselement zu verstehen, in dem gesonderte autoren spezifische Servicebeschreibungen angegeben werden können.

Es enthält Informationen über den Service, die von den Implementierungen nicht angerührt werden. Sie dienen den Programmieren als Zusatzinformation. Das Element <documentation> kann in jedem WSDL-Element angegeben werden. (Hauser, 2004, S. 84)

In den meisten Fällen wird das <documentation>-Element direkt unterhalb des <description>-Elements angegeben.

Tabelle 9 Beispiel-Listing für <documentation> (Nilo & Lafon, 2007)

```
<documentation>
  This document describes the Greath Web service.  Additional
  application-level requirements for use of this service --
  beyond what WSDL 2.0 is able to describe -- are available
  at http://greath.example.com/2004/reservation-documentation.html
</documentation>
```

4.3.7 Verlinkung von WSDL-Dokumenten

Das Element <import> wird verwendet um WSDL Definitionen, die einem anderen Namespace angehören zu importieren.

Das Element `<include>` wird verwendet, um Elemente zu inkludieren, die in verschiedenen Dokumenten definiert sind, aber demselben Namespace angehören.

4.3.7.1 `<include>` Element

Mit Hilfe des `<include>`-Elements ist es möglich Webservice-Beschreibungen zu modularisieren. Schnittstellen-, Bindings-, und Service-Komponenten können von einem anderen WSDL-Dokument, welches den gleichen `<targetNamespace>` aufweist, zusammengesetzt werden. (Chinnici, Moreau, Ryman, & Weerawarana, 2007)

Das `<location>`-Attribut ist verpflichtend und spezifiziert den Ort dieser WSDL Dokumente. Der aktuelle Wert des `<targetNamespace>` des inkludierten WSDL muss mit dem `<targetNamespace>` des `<definitions>`-Element übereinstimmen. (Dhesiaseelan, 2004)

4.3.7.2 `<import>` Element

Das Konzept des `<import>`-Elements ist dem des `<include>`-Elements sehr ähnlich, mit Ausnahme, dass das importierte WSDL einem anderen Namespace angehört. Das `<namespace>`-Attribut ist verpflichtend, während das `<location>`-Attribut optional ist. (Dhesiaseelan, 2004)

Mit diesem Element kann man Teile eines WSDL-Dokuments oder ganze WSDL-Dokumente importieren, um damit die Möglichkeit zu schaffen, ein WSDL-Dokument flexibel aus mehreren Teilen zusammenzusetzen. (Hauser, 2004, S. 84)

5 WSRF

5.1 Webservices und Zustandsmodellierung

Neben den bereits bekannten Anwendungsbereichen wie beispielsweise Sicherheit, Transaktionen oder Zuverlässigkeit ist die Verwaltung von zustandsbehafteten Ressourcen in der Webservice-Community ein wichtiges Thema. Die Existenz von Zustandsinformationen steht der Webservice-Technologie generell nicht grundsätzlich entgegen.

Das Speichern, Verwalten und Managen von Zuständen ist bei vielen Computersystemen eine nicht zu vernachlässigbare Notwendigkeit. Zustandsinformationen werden benötigt, um Zwischenergebnisse anderen Clients zugänglich zu machen.

Beispielsweise verwaltet ein FlugReservierungssystem den gegenwärtigen Zustand an Flügen, Reservierungen, Flugzeug-Kapazitäten, Sitz-Zusammenstellungen und so weiter. Dieser Zustand dauert viel länger als die Dauer eines einzelnen Nachrichtenaustausches, um eine Reservierung zu erstellen oder eine Reservierung zu stornieren.

Die folgende Abbildung verdeutlicht am Beispiel einer Addierer-Funktionalität, dass bei herkömmlichen Webservices zwischen aufeinanderfolgenden Requests keine Speicherung des Zustands erfolgt:

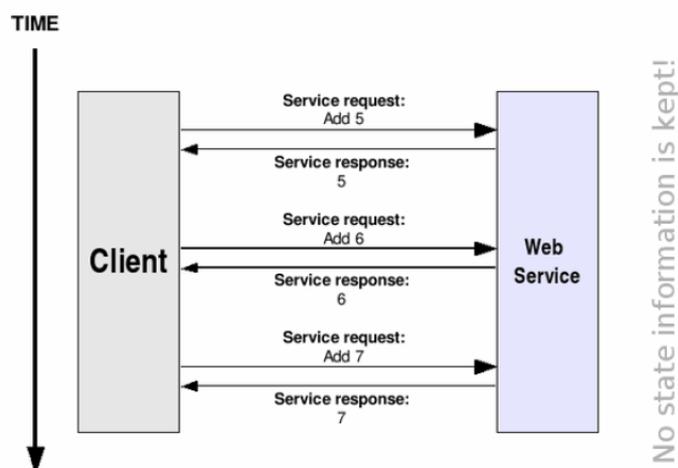


Abbildung 16 Zustandsloser Webservice-Aufruf (Sotomayor, 2005, S. 14)

Im Zusammenhang mit dem Begriff Zustand spricht man von einer Reihe persistenter Daten oder Informationselemente, die eine Lebensdauer besitzen, welche über einen einzelnen Request/Response-Nachrichtenaustausch zwischen Kommunikationspartnern hinausgeht. Die folgende Abbildung zeigt einen zustandsbehafteten Webserviceaufruf.

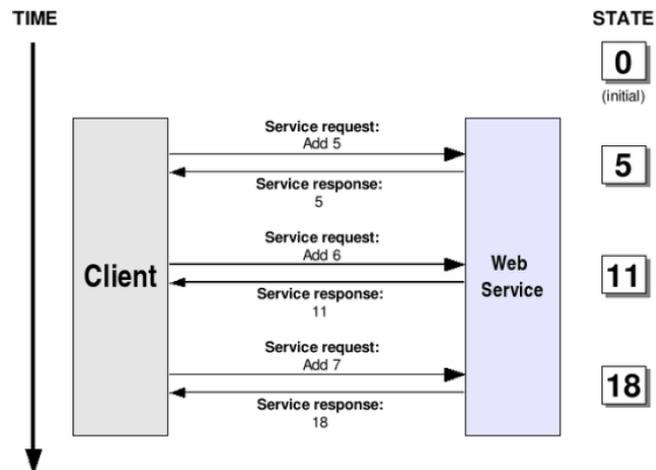


Abbildung 17 Zustandsbehafteter Webservice-Aufruf (Sotomayor, 2005, S. 15)

5.1.1 Webservices – ein ursprünglich, zustandsloses Medium

Die einzige Information, die zur Ausführungszeit eines Webservice-Requests bereitsteht, ist die Information, die innerhalb des Request eingeschlossen ist. Diese Tatsache macht es prinzipiell schwierig, Applikationen zu erstellen, bei denen die Information zwischen den Interaktionen weiter fortbesteht

Es ist ein Merkmal von Webservices, dass die zustandslose Implementierung dazu führt, dass die Zuverlässigkeit, die Skalierbarkeit und Wiederverwendbarkeit erhöht wird. Webservices können beispielsweise bei Auftreten von Fehlersituationen immer wieder neu gestartet werden, ohne dass auf eventuell verlorengegangene Zustandsinformationen aus vorangegangenen Interaktionen Rücksicht genommen werden muss. Ebenso können von einem Webservice mehrere Instanzen auf verschiedenen Servern bereitgestellt werden, damit eine bessere Lastverteilung erreicht wird.

Somit kann man feststellen, dass Zustandslosigkeit generell als eine gute Konstruktion für Webservice Implementationen betrachtet werden kann.

Generell sollte es also weiterhin oberstes Prinzip sein, dass möglichst zustandslose Services im Rahmen von Anwendungen realisierbar sind und auch die zahlreichen Vorteile von den üblicherweise zustandslosen Webservices nicht verlorengehen.

5.1.2 Mangel an Konventionen bei bisherigen Standards

Im Rahmen der Webservice-Technologie gab es bislang noch keinen formalen Mechanismus bzw. einheitliche Regelung, um die Beziehung zwischen Webservices und einem Zustand (state) darzustellen. Dies wurde bei den bisherigen Konzeptionen der Webservicestandards vernachlässigt.

Der Mangel an einer einheitlichen Regelung hat dazu geführt, dass der Umgang mit zustandsbehafteten Ressourcen von den Applikationen auf eigene und unterschiedliche Art und Weise gelöst wird.

Es gibt viele Wege, um einen Zustand bei Webservices darzustellen bzw. eine Art zustandsbehafteter Konversation zu implementieren. Die Folge daraus war, dass für diese Art zustandsbehafteter Konversation eine Vielzahl von unterschiedlichen Implementierungen existieren kann. Dies wiederum resultiert in erhöhten Integrations-Kosten zwischen den Systemen. Dadurch ist es schwierig verschiedene Systeme zu integrieren und dies ist mit einem hohen Aufwand verbunden. Es stellt sich die Frage, wie die Thematik von zustandsbehafteten Ressourcen im Bereich der Webservices-Interaktionen gelöst wird.

Deshalb ist es notwendig, dass klassische Webservices eine Erweiterung zur Verfügung gestellt bekommen, um den Zugriff auf Ressourcen individuell zu realisieren, und um somit neuen Herausforderungen gerecht zu werden. In diesem Bereich fällt das Abfragen von Zuständen einer Ressource, die Erschaffung und Zerstörung von Ressourcen, das Speichern und Abrufen von Zustandswerten oder die Fehlerbehandlung.

Mit der Entwicklung des Webservice Resource Framework wurden schliesslich Standardkonventionen geschaffen, um auf die obengenannten Anforderungen eine Lösung bereitzustellen.

5.2 Web Service Resource Framework – Das Konzept

5.2.1 Allgemeines

Die Web Service Resource Framework(WSRF)-Spezifikationenfamilie stellt eine wichtige Erweiterung und Ergänzung für die bereits beschriebene WS-Technologie dar. Das WSRF wurde durch viele namhafte Firmen und Institutionen, wie Globus Alliance, IBM und HP entwickelt. Das Standardisierungsgremium Organization for the Advancement of Structured Information Standards (OASIS) hat WSRF als Standard aufgenommen. Derzeit liegt WSRF in der Version 1.2 vor.

Das Web Service Ressource Framework soll Konventionen bereitstellen, welche klare Richtlinien für den Zugang zu Ressourcen schafft. Dabei soll das Framework ein Standard sein, um Zustandsinformationen einheitlich abbilden zu können und dabei sowohl den Anforderungen des Grid Computing gerecht zu werden, als auch das Wesen der Webservice-Technologie unberührt zu lassen.

Das WSRF standardisiert sogenannte Patterns (Muster) und Mechanismen für

- die Definition von Operationen,
- die Gruppierung,
- den Lebenszyklus und
- die Fehlerbehandlung

von WS-Ressourcen.

Die mittlerweile ausgegliederte Web Sevice Notification (WSN)-Spezifikationenfamilie ist in Verbindung mit dem WSRF ein weiteres wichtiges Rahmenwerk, die in Kapitel 6 vorgestellt wird. Durch Verwendung von WS-Notification ist es Interessenten möglich, sich über die in den `Resourceproperties` bereitgestellten Zustandsinformationen bei Auftreten von Änderungen informieren lassen.

Implementationen, welche das WSRF bzw. WSN vollständig bzw. teilweise unterstützen sind u.a: Globus Toolkit , Muse, Websphere Application Server, WSRF , WSRF.NET und Unicore.

5.2.1.1 Die WSRF-Idee

Service-Requestoren interagieren mit den `stateful resources` mittels des sogenannten `Implied Resource Patterns`. (Joseph, Ernest, & Fellenstein, 2004)

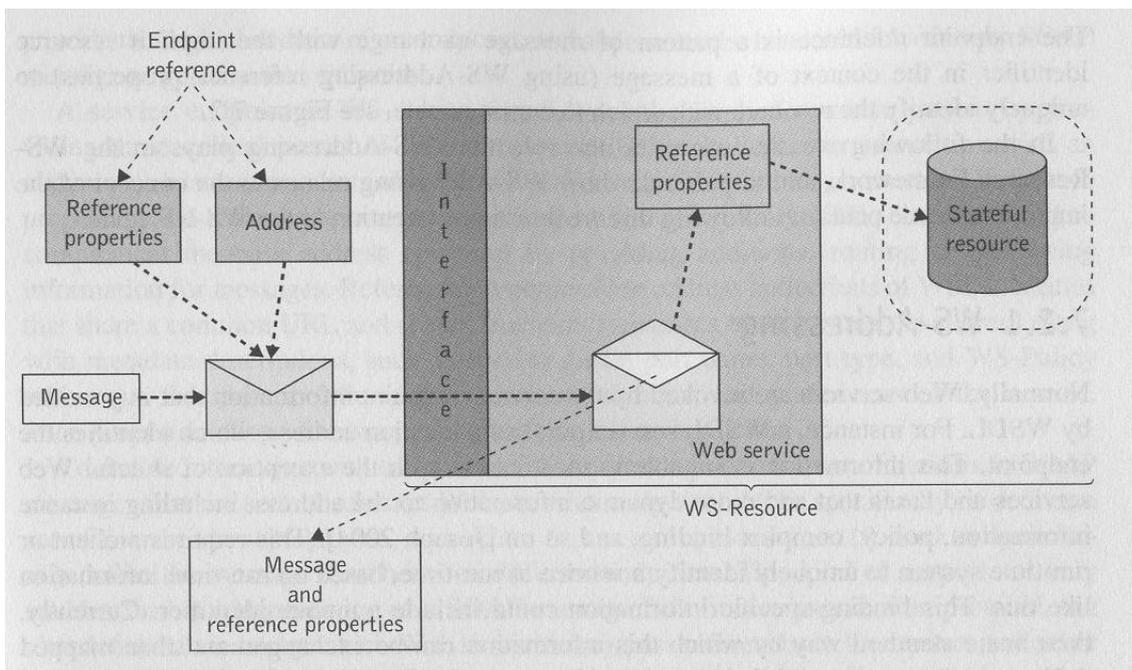


Abbildung 18 Implied Ressource Pattern (Papazoglou, 2007, S. 219)

Der Zustand wird nicht direkt in den Webservice integriert, sondern in einer separaten logischen Einheit gehalten. Es wird also eine strikte Trennung von zustandslosen Webservices und zustandsbehafteten Ressourcen praktiziert (Ressourcenansatz).

Der `Implied Resource Pattern` setzt sich aus einer Reihe an Webservice-Konventionen zusammen. Im Einzelnen ist das XML, WSDL und WS-Adressing. (Foster, et al., 2004b)

Die `WS-Resource` ist dabei jenes Konstrukt, welches die Trennung zwischen Webservice und dem Zustand repräsentiert. Jede `WS-Ressource` wird einem eindeutigen Schlüssel zugeteilt, über welche sie jederzeit angesprochen werden kann.

Die Adresse zu einer `WS-Ressource` wird als `Endpoint Reference (EPR)` bezeichnet. Diese beinhaltet ua. die `Endpoint Address` des Webservices und

den Identifikator der Ressource. Diese Angaben werden in den Referenzeigenschaften der `Endpoint Reference` festgelegt.

Das WSRF führt die Idee einer XML basierten Dokumentenbeschreibung ein (Resource Properties Dokument), in welchem der Zustand der Ressource beschrieben wird. (Banks, 2006, S. 7)

5.2.1.2 WSRF-Spezifikationen

Die Spezifikationen des WSRF sind über die verschiedenen Schichten des Webservice-Stapels verteilt.

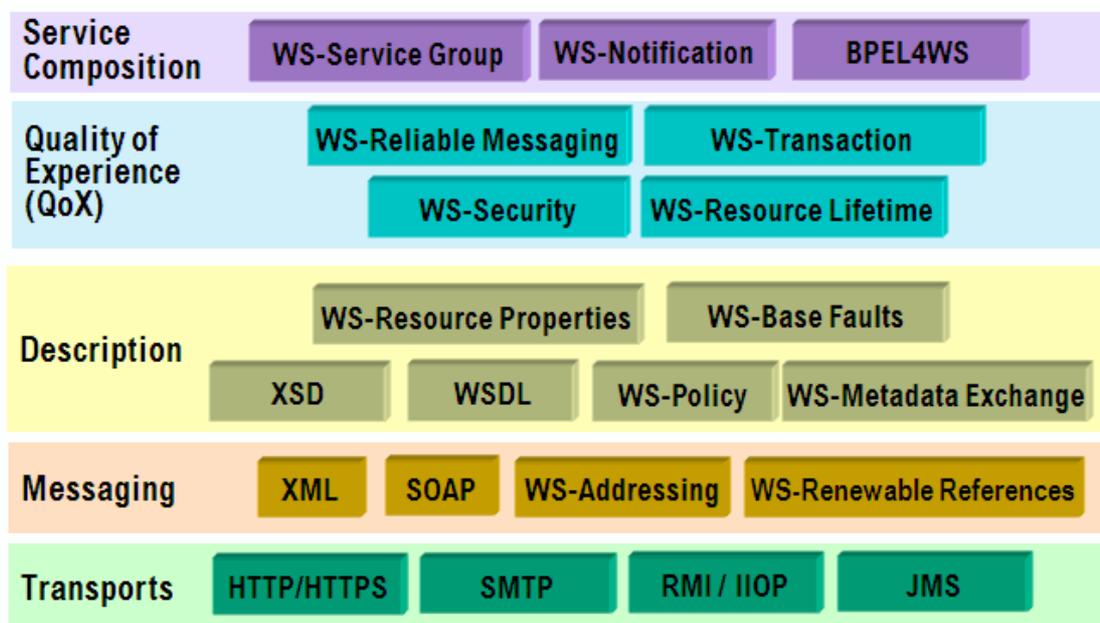


Abbildung 19 Einordnung in Bezug auf andere Webservice-Standards (Sabbah, 2004, S. 8)

In den folgenden Unterkapiteln werden die Spezifikationen von WSRF beschrieben.

5.2.1.2.1 WS-Resource

Die `WS-Resource`-Spezifikation definiert das Konstrukt der `WS-Resource`, welches eine Kombination aus einem Webservice und einer Ressource darstellt und als zentraler Bestandteil des WSRF gilt.

5.2.1.2.2 WS-ResourceProperties

Die WSRF-Teilspezifikation `WS-ResourceProperties` stellt ein Rahmenwerk zur Verfügung, mit welchem die Werte der `WS-Resource` abgefragt und upgedated werden können.

5.2.1.2.3 WS-ResourceLifetime

Die `WS-Lifetime` Spezifikation regelt die Lebensdauer einer `WS-Resource`. Entwickler können mit Hilfe dessen in ihren Applikationen festlegen, ob eine `WS-Resource` unmittelbar oder zu einem bestimmten Termin zerstört werden soll.

5.2.1.2.4 WS-ServiceGroup

Des weiteren ist es auch möglich, eine Gruppierung von mehreren `WS-Ressourcen` vorzunehmen. `WS-Resource ServiceGroup` ist die dafür vorgesehene Spezifikation.

5.2.1.2.5 WS-Basefault

Mithilfe von `WS-Resource Basefaults` wird die Fehlerbehandlung geregelt.

5.2.2 Definitionen

1: Der Zweck des WSRF ist die Definition eines allgemeinen und offenen Frameworks für die Modellierung und den Zugriff auf zustandsbehaftete Ressourcen (`stateful resources`) unter der Verwendung von Webservices. Dies inkludiert Mechanismen, um Sichtweisen des Zustands zu beschreiben, um das Management des Zustands unter Verwendung von Eigenschaften zu unterstützen, welche mit dem Webservice verknüpft sind, und um zu beschreiben, wie diese Mechanismen auf Webservice Gruppen erweiterbar sind. (OASIS, 2008)

2: Das WSRF definiert einen neuen Ansatz, um auf zustandsbehaftete Ressourcen zuzugreifen, welche die Infrastruktur des Grid Computing unterstützen und gleichsam zustandsbehaftete Web Services im allgemeinen zur Verfügung stellt - in anderen Worten: die Spezialfall-Lösung für das Gridcomputing (OGSI) wurde zum Vorteil für alle Web Services (mittels WSRF) verallgemeinert. (Antonopoulos, 2004)

3: Das WSRF ist eine Gruppe an Spezifikationen, welche einen standardisierten Ansatz bereitstellen, um mit zustandsbehafteten Ressourcen (`stateful resources`) in einer in hohem Maße zustandslosen („`stateless`“) Umgebung einer Webservice Applikation interagieren. Diese zustandsbehaftete Ressource kann in virtueller Hinsicht alles Mögliche sein: Angefangen von einer Datenbank bis zu einem elektronischen Hamster. Durch die Verwendung des WSRF ist es möglich, jegliche Einheit anzusprechen, welche durch Veränderung seiner Eigenschaften manipuliert werden kann. Dabei versteht man unter einer `WS-Resource` die Kombination einer zustandsbehafteten Ressource (zb. Datenbank oder eine Hardwareeinheit) und einem Webservice mit welchen diese interagiert. (Sundaram, 2005a, S. 2)

5.2.3 Stateful Resource

Um das Wesen einer `WS-Resource` zu verstehen ist es notwendig zu wissen, was eine sogenannte `stateful resource` ist.

Im Rahmen des WSRF wird eine Ressource in ein sogenanntes `WS-Resource`-Konstrukt eingebettet, über welche Nachrichten gesendet und Informationen abgefragt werden können.

Eine `Stateful Resource` kann beispielsweise folgendes sein:

- eine Tabelle,
- ein Datensatz in einer Datenbank
- Objekte wie beispielsweise Java Beans
- ein Temperaturprozessor
- ein virtueller Einkaufswagen

Eine `stateful Resource` kann als sogenannte „bag of state“ verstanden werden, in welcher die Information, die den Zustand formt, gebündelt wird. (Graham, et al., 2004, S. 384)

5.2.3.1 Merkmale einer Ressource

Folgende Merkmale einer Ressource existieren (Foster, et al., 2004b, S. 10):

- Die Ressource muss einen definierten Satz an Zustandsinformationen enthalten.

Die Darstellung von diesen Informationen soll im Rahmen eines XML-Dokumentes erfolgen.

- Für die Ressource wird ein Lebenszyklus festgelegt.

Ein Client kann die Erstellung der Ressource veranlassen. Dadurch ist eine serverseitige Speicherung der Information für diese Interaktion möglich. Später muss wieder die Freigabe der Ressource sichergestellt werden.

- Die Ressource muss mindestens einem Webservice zugeordnet sein, welcher sie kennt und mit ihr arbeiten kann.
- Die Identifikation der Stateful Resource erfolgt durch den Stateful Resource Identifier. Dieser ist in der Endpoint-Reference des sogenannten ResourceProperties-Dokumentes angegeben

5.2.4 WS-Resource

Eine `WS-Resource` ist die Komposition einer Ressource und eines Webservices mittels welchem auf die Ressource zugegriffen werden kann (Graham, Karmarkar, Mischkin, Robinson, & Sedukhin, 2006, S. 5).

Nachfolgend ist ein Webservice mit einer URL-Adresse (`http://www.example.com/service`) abgebildet. Dieser Webservice stellt einen Zugang zu drei Ressourcen zur Verfügung, die jeweils als „A“, „B“ und „C“ identifiziert werden. Im abgebildeten Fall erfolgt der Zugang auf die Ressource „C“ und das Konstrukt wird deshalb als `WS-Resource C` bezeichnet.

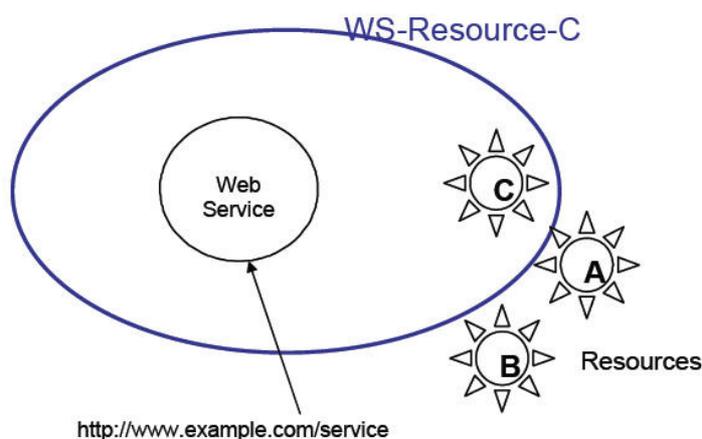


Abbildung 20 Webservice mit 3 Ressourcen (Graham, Karmarkar, Mischkin, Robinson, & Sedukhin, 2006, S. 6)

5.2.5 WS-Ressourcen Kommunikation (The Implied Resource Pattern)

Ein Client kann mit einer Ressource nur über die Webservices kommunizieren, welche mit der Ressource verbunden sind bzw. ein `WS-Resource`-Konstrukt bilden.

Um einen Zugriff auf eine Ressource zu haben, muss deshalb diese `WS-Resource` eindeutig identifizierbar sein. Die Lösung liegt in der Anwendung des folgenden Kommunikationsmusters, bei welchem die Identifikation der Ressource implizit im Header des SOAP-Protokolls angegeben wird. (Sprengel, 2006, S. 3)

Für die Requestor-Applikationen stellt sich der Webservice als zustandsbasiert dar, obwohl er selbst ein komplett zustandsloses Verhalten innehat. Diese Modellierungsart wird deshalb „`Implied Resource Pattern`“ genannt.

5.2.5.1 *The Implied Resource Pattern*

Der `Implied Resource Pattern` ist eine Konvention von XML, WSDL und WS-Adressing, welche definiert, wie eine einzelne `WS-Resource` im Zusammenhang mit der Abarbeitung einer Webservice-Nachricht referenziert wird. (Graham, et al., 2004, S. 393)

Jede an eine `WS-Resource` gesendete Nachricht muss einen Bezeichner enthalten, welcher die zu verwendete Ressource eindeutig identifiziert.

Mit implizit ist gemeint, dass die anfragende Instanz den `stateful resource identifier` nicht als einen expliziten Parameter im `Body` der `Request Message` besorgen muss.

Anstattdessen wird die `stateful Resource` mit der Ausführung des Nachrichtenaustausches implizit assoziiert. (Foster, et al., 2004b, S. 11)

Dabei wird der sogenannte `WS-Adressing-Mechanismus Endpoint Reference (EPR)` verwendet, um die Referenzen auf die `WS-Resource` darzustellen. Der Begriff `Pattern` wird verwendet um die Beziehungen zwischen Webservices und `stateful Resources` anzuzeigen, die kodifiziert sind durch eine Reihe existierender WS-Technologien (im speziellen XML, WSDL und `WS-Adressing`). (Foster, et al., 2004b, S. 11)

5.2.5.2 WS-Addressing

Normalerweise werden Webservices mit Hilfe der Service Endpoint Information aufgerufen, welche durch WSDL bereitgestellt wird. Beispielsweise hat ein WSDL Service Port eine Ortsadresse, welche den Endpunkt identifiziert. Diese Angabe der Information erweist sich in den meisten Fällen als geeignet. Die Ausnahme stellen jedoch zustandsbehaftete Webservices und jene Fälle dar, wo noch mehr dynamischere Information zur Adresse hinzugefügt werden soll (Policy, komplexes Binding, etc). (Papazoglou, 2007, S. 220)

Diesbezüglich wäre ein Client erforderlich, welcher ein Service zur Laufzeit eindeutig identifizieren kann (unter Verwendung eines eindeutigen Identifiers). Bisher hatte es keine einheitlichen Konventionen für diesen Informationsaustausch gegeben. Die WS-Addressing Spezifikation hat sich dieses Problems angenommen, indem ein Mechanismus bereitgestellt wurde, um die Endpoint Informationen zu identifizieren und zu beschreiben. Diese Information wird in den SOAP Headern angegeben. (Papazoglou, 2007, S. 220)

WS-Addressing stellt transportneutrale Mechanismen zur Verfügung, um Webservices und Messages zu adressieren. Die Spezifikation wurde eingeführt, um den Begriff eines Pointers auf ein Webservice zu standardisieren. (Graham, et al., 2004)

Der Hauptzweck von WS-Addressing ist es die Message Addressing Information in den Webservice Nachrichten einzubauen. Die SOAP selbst stellt keine Funktionen für die Identifizierung von Endpoints zur Verfügung. (Papazoglou, 2007, S. 220)

WS-Addressing definiert die Konstrukte EndpointReferences und MessageAddressingProperties. (Gudgin, Hadley, & Rogers, 2006)

5.2.5.2.1 EndpointReference

Die Endpoint-Reference (EPR) ist ein Konstrukt der Webservice-Spezifikation WS-Addressing und eine ganz wesentlicher Bestandteil im Rahmen des Implied Resource Patterns. Unter Endpoint Reference versteht man die XML-Darstellung eines netzwerkweiten Pointers auf einen Webservice. (Foster, et al., 2004b, S. 11)

Die `endpoint reference` beinhaltet (Joseph, Ernest, & Fellenstein, 2004, S. 632):

- eine Service-Adresse (`<wsa:Address>`),
- Metadaten, die mit dem dem Webservice assoziiert werden,
- Policy Informationen bezogen auf die Verwendung des Services
- Referenz Eigenschaften (`<wsa:ReferenceProperties>`)

Der `ReferenceProperties`-Teil kann generell unterschiedliche Informationen enthalten, welche zusätzlich zum Adressen-Teil angegeben werden.

5.2.5.2.2 MessageAddressingProperties

Die `MessageAddressingProperties` stellen Referenzen für die Endpunkte zur Verfügung, welche in einer Interaktion involviert sind. (Gudgin, Hadley, & Rogers, 2006)

Tabelle 10 XML Infoset (Gudgin, Hadley, & Rogers, 2006)

```

<wsa:To>xs:anyURI</wsa:To> ?
<wsa:From>wsa:EndpointReferenceType</wsa:From> ?
<wsa:ReplyTo>wsa:EndpointReferenceType</wsa:ReplyTo> ?
<wsa:FaultTo>wsa:EndpointReferenceType</wsa:FaultTo> ?
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:MessageID>xs:anyURI</wsa:MessageID> ?
<wsa:RelatesTo RelationshipType="xs:anyURI"?>xs:anyURI</wsa:RelatesTo>
*
<wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?

```

In den hier angeführten Beispiellistings werden hauptsächlich nur die folgenden zwei Elemente eingesetzt:

- `wsa:To` – gibt die URI des absoluten Nachrichtenempfängers an
- `wsa:Action` – dieses verpflichtende Element gibt durch Angabe einer URI die Absicht der Nachricht an.

5.2.5.3 WS-Resource Factory

Wenn das Verarbeiten eines Requests dazu führt dass eine Resource erzeugt wird, dann repräsentiert der Webservice eine sogenannte `WS-Resource Factory`. Dies liegt darin begründet, dass in der Antwort die `endpoint reference` der `WS-Resource` an den Service Requestor zurückgeschickt wird.

Diese endpoint reference ist das Resultat aus der Komposition der soeben neu erstellten stateful Resource und seinem assoziierten Webservice.

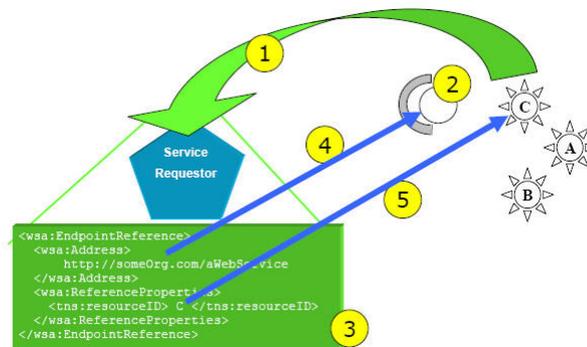


Abbildung 21 Reference enthält einen stateful Resource Identifier (Foster, et al., 2004b, S. 12)

5.2.5.4 WS-Resource qualified endpoint reference

Immer dann, wenn die an den Requestor zurückgesandte Endpoint Reference eine eindeutige Identifikation auf eine WS Resource darstellt, spricht man von einer sogenannten WS-Resource qualified endpoint reference.

Durch Verwendung dieser Ressourcen-Identifikation, ist gewährleistet, dass der Webservice weiterhin seine Zustandslosigkeit beibehält. Eine Abspeicherung welche Resource somit welchem Client zugeordnet ist, wird dadurch vermieden. (Foster, et al., 2004b)

Die Applikationen des Service Requestors veranlassen, dass diese Endpoint Reference in den Nachrichten-Headern der Anfragen hinzugefügt werden. Die folgende Abbildung 22 veranschaulicht eine SOAP-Anfrage, in welcher diese qualified endpoint reference verwendet wird.

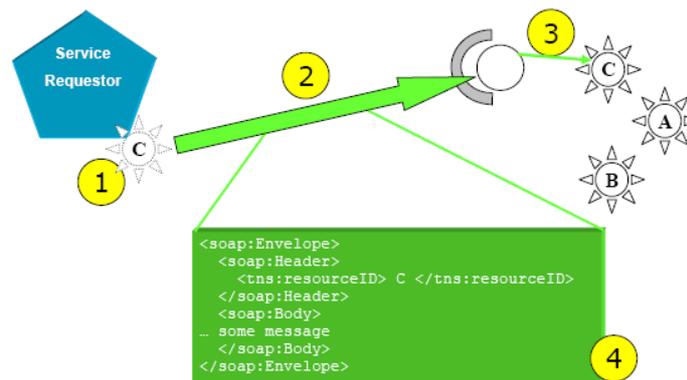


Abbildung 22 SOAP-Anfrage unter Verwendung einer qualified endpoint reference (Foster, et al., 2004b, S. 14)

Der Webservice (3) extrahiert den `stateful Resource identifier` von der SOAP Nachricht und verwendet diesen, um die `stateful Resource` zu lokalisieren, damit die die Request Nachricht verarbeitet werden kann. (Foster, et al., 2004b, S. 14)

5.2.6 WS-Resource Zustandsinformationen

Einer Requestor-Applikation wird vom Webservice die Möglichkeit zur Verfügung gestellt, die Zustandsinformationen einer Ressource über die angebotene Schnittstelle auszulesen und auch gegebenenfalls zu verändern.

Die Sicht auf den Zustand der Ressource kann hierbei jedoch begrenzt werden. Jene Zustandswerte, auf welche über die `WS-Resource` öffentlich zugegriffen werden kann, bezeichnet man als sogenannte `Resource Properties`.

Für die Requestor-Applikation wurde die Ressource bereits von der `Resource Factory` generiert. Um nun Informationen über die Struktur der Ressource zu bekommen, benötigt der Client das sogenannte `Resource Properties Document`.

Das `Resource Properties Document (RPD)` ist ein XML-Schema und beschreibt die Ressource. Es beinhaltet die Struktur der Informationen, welche den Zustand in den Ressourcen darstellt.

Das RPD wird referenziert durch die WSDL Beschreibung des Services und beschreibt beispielsweise einen Einkaufswagen, einen Drucker, einen Druckjob oder mit was auch immer der Client interagiert. (Banks, 2006, S. 7)

Da das Resource Properties Dokument mit dem WSDL Dokument verlinkt ist, sind diese Informationen für den Client bzw. den Entwicklern verfügbar. Die Spezifikation `WS-ResourceProperties` gibt nicht vor, wie ein Service die Implementierung des RPD vorzunehmen hat. Die Zustandsinformationen können beispielsweise in einer Datenbank abgespeichert werden. Für den Requestor bleibt die Implementierungsart jedoch verborgen.

5.2.7 Deklaration am Beispiel eines Shoppingservices:

Die Vorgehensweise (wie beispielsweise die Einbindung des RPD) lässt sich anhand eines einfachen Shopping Webservice veranschaulichen, welcher die Verwaltung mehrerer Ressourcen innehat.

Der konkrete Webservice, welcher unter der Adresse „<http://www.example.com/SimpleShoppingService>“ erreichbar ist, stellt mehrere Funktionalitäten bereit. Dazu gehören u.a. die Erstellung eines Einkaufswagens, das Hinzufügen von Artikel, das Entfernen von Artikeln, usw.

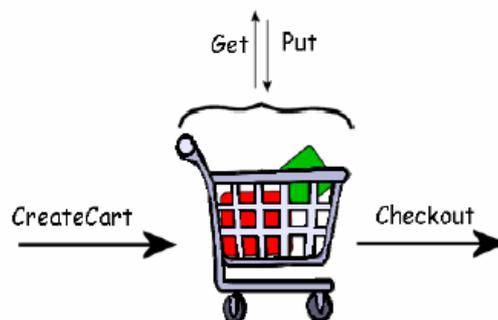


Abbildung 23 Einfacher Einkaufswagen (Banks, 2006, S. 13)

Das Shopping Service stellt einen Zugang zu einem durchsuchbaren Produktkatalog bereit. Ein *Produktcode* und eine *Bestellmenge* werden verwendet um einen Anfangseintrag in einem Einkaufswagen zu bilden, an welchen zusätzliche Produkte und Menge hinzugefügt werden können. Eine *Checkout*-Operation veranlasst die Kauf-Interaktion, um ein Bezahlservice zu starten. Die Inhalte des Einkaufswagens werden dann zu Inhalten einer Kundenbestellung. Danach wird der Einkaufswagen zerstört. (Banks, 2006, S. 13)

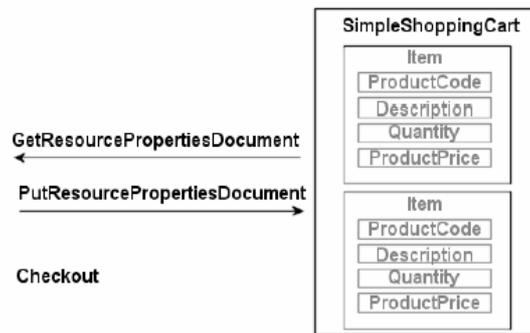


Abbildung 24 Einfaches Shopping Service (Banks, 2006, S. 10)

5.2.7.1 Resource Properties Document

Das Resource Properties Document hat beispielsweise folgendes Aussehen und repräsentiert die stateful Resource bzw. den Zustand des Einkaufswagens.

Tabelle 11 Resource Properties Document (Banks, 2006, S. 18)

```
<ssc:SimpleShoppingCart>
  <ssc:Item>
    <ssc:ProductCode>Cat-A2004-87968556</ssc:ProductCode>
    <ssc:Description>Garden String - 150m</ssc:Description>
    <ssc:Quantity>1</ssc:Quantity>
    <ssc:ProductPrice>1.59</ssc:ProductPrice>
  </ssc:Item>
</ssc:SimpleShoppingCart>
```

Das RPD muss als eine Global Element Declaration (GED) in XML-Schema definiert sein. Das GED bezieht sich auf die Definition des Wurzelements eines Resource Properties Document. Das RPD kann mehrere resource property-Elemente enthalten, welche als Kind-Elemente des Wurzelements zum Vorschein kommen. Jedes dieser Resource Properties müssen ebenso als GED definiert sein. (Graham, Karmarkar, Mischkin, Robinson, & Sedukhin, 2006, S. 11)

5.2.7.2 Import des Namespaces

Tabelle 12 WSDL-Import Element (Banks, 2006, S. 16)

```
<wsdl:import namespace="http://docs.oasis-open.org/wsrp-2"
location=" http://docs.oasis-open.org/wsrp-2"/>
```

5.2.7.3 Schema Declaration

Das Schema wird innerhalb des <types>-Elements von WSDL deklariert und kann durch die <import>-Funktion importiert werden.

Tabelle 13 ESDL-Schema für types (Banks, 2006, S. 16)

```

<wsdl:types>
<xsd:schema . . .>
<xsd:import . . . schemaLocation="./SimpleShoppingCart.xsd" />
</xsd:schema>
</wsdl:types>

```

5.2.7.4 PortType Deklaration

Um nun das Konstrukt der *WS-Resource* zu deklarieren, muss die Ressource an den Webservice angebunden werden. Dies geschieht, indem das RPD mit dem `<portType>` assoziiert wird. Die Anbindung des RPDs erfolgt durch die Nutzung der Attribut-Erweiterung des `<portType>`-Elements, mit welchem es möglich ist, zusätzliche Attribute an den `<portType>` zu binden. Es handelt sich dabei um das *Resource Properties*-Attribut aus dem zugehörigen Namensraum der *Resource Properties*-Spezifikation von WSRF, welche als Wert den qualifizierten Namen `<wsrf-rp:ResourceProperties>` enthält. (Sprengel, 2006)

Tabelle 14 WSDL-Schema für portType (Banks, 2006, S. 16)

```

<wsdl:portType name="SimpleShoppingCart"
wsrf-rp:ResourceProperties="ssc:SimpleShoppingCart">
. . .
</wsdl:portType>

```

Zur Vervollständigung ist hierbei noch anzumerken, dass ebenso noch weitere Deklarationen innerhalb von WSDL zu erfolgen haben. Natürlich muss auch das `<binding>` und `<service>` Element deklariert werden, welche ja mit dem `<portType>`-Element assoziiert sind. Diese geben die Wahl des für die Operationen zugehörige Transportprotokoll (HTTP, SMTP), sowie die Adresse an.

5.2.8 Vergleich zu Servicedefinitionen ohne WSRF-Nutzung

Es wurde bereits in der Einleitung erwähnt, dass die Einführung des WSRF wesentlich zu einer verbesserten Regelung bezüglich Zustandsmodellierung geführt hat. In den folgenden Unterkapitel werden jene für das WSRF charakteristische Punkte beschrieben (im Vergleich zu Servicedefinitionen, wenn kein WSRF eingesetzt wird).

5.2.8.1 Verzicht auf die explizite Angabe des Identifiers in WSDL

Einer der Unterschiede betrifft hierbei zunächst die Identifikation der Ressourcen.

WSRF verhindert es, den Identifier explizit in der WSDL Beschreibung anzuführen. Der Identifier ist Teil der EPR und wird implizit in allen Nachrichten gemäß der WS-Adressing-Regeln eingeschlossen.

Der Resource Identifier selber ist für die anfragende Applikation von keinem Interesse. Die Applikation nutzt nur den EPR als Referenz auf eine spezifische WS-Resource. WSRF führt dazu, dass der Resource Identifier aus dem Body der Nachricht entfernt wird. Im Falle eines „Non-WSRF“-Design wird typischerweise der Identifier meist in den Messages dargestellt (typischerweise als Parameter) und in WSDL beschrieben. (Banks, 2006, S. 7)

Die nachfolgende Gegenüberstellung zeigt den Unterschied beim Aufbau der SOAP-Nachrichten.



Abbildung 25 SOAP-Nachricht mit und ohne WSRF (Banks, 2006, S. 8)

Der Requestor muss nicht die Identität der Ressource als einen expliziten Parameter im Body der Request Message bereitstellen. Anstattdessen wird der Kontext mittels MessageHeaders aufgebaut. (Joseph, Ernest, & Fellenstein, 2004, S. 633)

5.2.8.2 Assozierung des RPDs mit dem portType

Das `ResourcePropertiesDocument` wird mit dem `<portType>` assoziiert. Standardisierte Nachrichtendefinitionen werden durch Verwendung der WSRF-Spezifikation importiert. (Banks, 2006, S. 14)

5.2.8.3 Standardisierung von gängigen Operationen

Die Definition von einfachen Operationen wie beispielsweise `<Get>` und `<put>` ist vollständig standardisiert. Nachrichtendefinitionen, die durch das `wsrf-rpw` Präfix referenziert werden, werden von der WSRF-Spezifikation zur Verfügung gestellt. (Banks, 2006, S. 8)

Tabelle 15 Verwendung von standardisierten Operationen (Banks, 2006, S. 15)

```
<wsdl:operation name="GetResourcePropertyDocument">
<wsdl:input message="wsrf-rpw:GetResourcePropertyDocumentRequest" . .
/>
<wsdl:output message="wsrf-rpw:GetResourcePropertyDocumentResponse" .
./>
```

5.2.8.3.1 Spezifische Operation „create“

Abseits der standardisierten Operationen ist es trotzdem notwendig spezifische Operationen in der WSDL-Beschreibung zu deklarieren. Im ShoppingService-Szenario ist es u.a. notwendig eine `<create>`-Operation zu definieren, um einen Einkaufswagen zu erstellen. Wird dieser Request gesendet, dann erzeugt der Server eine Referenz auf die neue `WS-Resource` und sendet den Identifier für den Einkaufswagen (S1) in Form einer `EndpointReference` zurück (siehe Tabelle 16 Response-Listing Tabelle 16). (Banks, 2006, S. 17)

Tabelle 16 Response-Listing (Banks, 2006, S. 17)

```
<ssc:CartCreateResponse>
<wsa:Address>http://www.example.com/SimpleShoppingService</wsa:Address
>
<wsa:ReferenceParameters>
<scImpl:CartId>S1</scImpl:CartId>
</wsa:ReferenceParameters>
</ssc:CartCreateResponse>
```

Dieser Identifier wird in allen weiterfolgenden Nachrichtenaufrufen angegeben, welche an das Webservice gesendet werden.

5.3 Abfrage und Update von Zustandsinformationen

Die Definition des zuvor genannten `ResourcePropertiesDocument` ist Teil der Spezifikation `WS-ResourceProperties`. Des Weiteren sind in dieser Spezifikation auch die Definition der Operationen angegeben, um die Zustandsinformationen einer `WS-Resource` abzufragen bzw. upzudaten.

Die Spezifikation stellt auch Standardwege zur Verfügung, mit welchem Requestors den Benachrichtigungsmechanismus `WS-Notification` benutzen können, um sich über Änderungen der Eigenschaftswerte der Ressourcen benachrichtigen zu lassen. (Banks, 2006, S. 5)

Folgende Abbildung zeigt eine Computersystem-Ressource, deren Zustandsinformation mittels einer definierten Reihe an Interfaces freigelegt wird (die Definition hierzu erfolgt mittels des `<portTypes>`-Element).

Figure 13 A computer system WS-Resource with WS-ResourceProperties interfaces

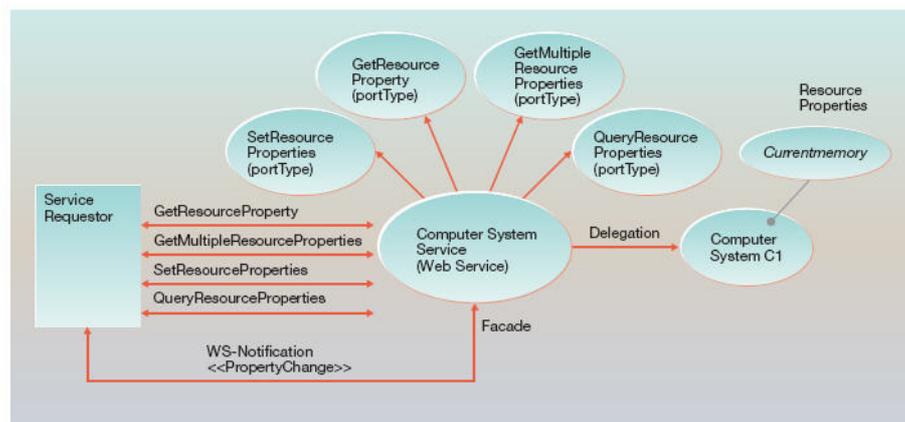


Abbildung 26 WS-Resource mit WS-ResourceProperties-Schnittstellen (Joseph, Ernest, & Fellenstein, 2004)

5.3.1 ResourceProperties-Operationen

Das WSRF definiert Standard-Operationen, um einzelne oder mehrere Zustandsinformationen auszulesen oder zu verändern. Es existieren folgende `ResourceProperties`-Operationen:

- `GetResourcePropertyDocument` wird verwendet, um das gesamte `ResourcePropertyDocument` von der `WS-Resource` zu erhalten. Es sind darin alle Properties der Ressource enthalten.

- PutResourcePropertyDocument sendet eine neue Version des ResourceProperties Dokument an die WS-Resource. Es ist als Gegenpart der Operation *GetResourcePropertyDocument* zu sehen.
- GetResourceProperty ermöglicht es die Werte einer WS-Resource einzeln abzufragen. Dies erfolgt durch Angabe des qualifizierten Namens der Eigenschaft (Property), welche abgefragt werden soll.
- GetMultipleResourceProperties hat eine ähnliche Funktionsweise wie *GetResourceProperty*. Es werden hierbei mehrere Eigenschaftswerte abgefragt.
- QueryResourceProperties erlaubt die Ausführung eines Abfrage-Ausdrucks auf dem Resource Properties Dokument.
- SetResourceProperties erlaubt eine Abfolge von mehreren Veränderungen bei einem Resource Properties Dokument innerhalb eines einzelnen Requests.

Innerhalb einer Nachricht kann zum Beispiel eine Eigenschaft neu eingefügt, eine andere abgeändert, und eine dritte gelöscht werden.

- InsertResourceProperties fügt ein oder mehrere Elemente in das Resources Properties-Element hinzu.
- UpdateResourceProperties kann verwendet werden, um die Wert einer Resource Property zu verändern.
- DeleteResourceProperties entfernt alle Werte für die Resource Property, die in dem Request genannt wird.

5.3.2 Beispielszenario Satellitenverwaltung

Die Operationen können anhand eines Systems veranschaulicht werden, welches das Management über eine Reihe von Satelliten innehat. Jeder Satellit stellt eine `stateful Resource` dar.

Ein zugehöriger Webservice stellt Funktionalitäten zur Verfügung, um den Satelliten zu steuern. Solche WS-Ressourcen können beispielsweise sein (Chase, 2007):

- Service zum Festsetzen und Abfragen der Höhe eines Satelliten
- Service zum Festsetzen und Abfragen der Position eines Satelliten

- Service, welches einen Zugang bereitstellt, um die Sterne zu zählen, die im Sichtbereich eines Satelliten liegen

5.3.2.1 Namensraum-Deklaration

An dieser Stelle werden zunächst die gängigen Namensraumdeklarationen angeführt, welche zu Beginn des Wurzelements `<SOAP-Env>` vor den Header und Bodyteilen angegeben sind.

Verwendete Namensraumbezeichner sind u.a:

- SOAP-Env
- sat
- wsa
- wsrp
- wsrl

Tabelle 17 Deklaration des Namensraums des Satellitensystems (Sundaram, 2005a, S. 8)

```
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/  
  
xmlns:sat=http://example.org/satelliteSystem  
  
xmlns:wsa=http://www.w3.org/2005/02/addressing  
  
xmlns:wsrp="http://docs.oasis-open.org/wsrp/2004/06/wsrp-  
WSResourceProperties-1.2-draft-01.xsd">  
  
xmlns:wsrl="http://docs.oasis-open.org/wsrp/2004/06/ws  
rf-WS-ResourceLifetime-1.2-draft-01.xsd">
```

Anmerkung:

Die Funktionsweise von WSRF wird in den nachfolgenden Kapiteln anhand von SOAP-Nachrichteninteraktionen bzw. WSDL-Definitionen veranschaulicht. An dieser Stelle ist anzumerken, dass in Bezug auf SOAP-Requests und SOAP-Responses aus Übersichtsgründen in den meisten Fällen auf die komplette Auflistung des gesamten SOAP-Umschlages verzichtet wird, da das Wurzelement samt Ihrer angegebenen XML-Namensräume immer den gleichen Aufbau darstellt. Es werden deshalb oft auszugsweise einerseits nur das Header-Element bzw. andererseits ausschliesslich das Body-Element des SOAP-Umschlages angeführt.

5.3.2.2 RPD-Deklaration

Das Resource Properties Document hat nun beispielsweise folgendes Aussehen und repräsentiert die `stateful resource` bzw. den Zustand des Satelliten.

Tabelle 18 Deklaration RPD des Satellitensystems (Sundaram, 2005a)

```
<satProp:GenericSatelliteProperties
xmlns:satProp="http://example.com/satellite">
<satProp:latitude>30.3</satProp:latitude>
<satProp:longitude>223.2</satProp:longitude>
<satProp:altitude>47700</satProp:altitude>
<satProp:pitch>49</satProp:pitch>
<satProp:yaw>0</satProp:yaw>
<satProp:roll>32</satProp:roll>
<satProp:focalLength>21999992</satProp:focalLength>
```

Beschreibung der ElementProperties:

- Position des Satelliten: latitude (Breitengrad), longitude (Längengrad), altitude (Höhe)
- Richtung des Satelliten: pitch (Neigung), yaw (Kursabweichung), roll (Rollbewegung)
- Distanz zum Punkt, zu welchem der Fokus besteht: focalLength, currentView

5.3.2.3 Identifier-Bekanntgabe im Zuge des Create-Requests

Voraussetzung für die Anwendung der Operationen ist es, dass der Requestor bereits über den Identifier der Ressource informiert worden ist (`WS-Resource qualified endpoint reference`) und diese Referenz auf die Ressource in seiner Anfrage mitsendet. Man spricht in diesem Zusammenhang von der Anwendung des `Implied Resource Pattern`. Somit kann der bearbeitende Webservice die Ressource richtig zuordnen und die Operationen dementsprechend anwenden.

Im Satellitenbeispiel wird diesbezüglich der Requester über die Identifikation der Ressource vom Webservice informiert. Dies geschieht im Rahmen eines `<create>`-Requests. Der Satelliten-Identifier mit dem Wert SAT9928 wird als Antwort zurückgeliefert.

Tabelle 19 Bekanntgabe des Identifier im Zuge des Create Requests (Sundaram, 2005a, S. 13)

```
<wsa:EndpointReference
```

```

xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
<wsa:Address>http://example.com/satellite</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatelliteId>SAT9928</sat:SatelliteId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>

```

5.3.2.3.1 Operation GetResourceProperty

Ein möglicher Kommunikationspattern wird nachfolgend anhand der Operation `GetResourceProperty` veranschaulicht. Mit diesem soll die `ResourceProperty <altitude>` abgefragt werden.

5.3.2.3.1.1 SOAP-Request (Headerteil)

Der Header des SOAP-Requests hat folgendes Aussehen:

Tabelle 20 Header des SOAP-Requests (Sundaram, 2005a, S. 22)

```

<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-ResourceProperties/
GetResourceProperty
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatelliteId>SAT9928</sat:SatelliteId>
</SOAP-ENV:Header>

```

Die Spezifikation WS-Adressing spezifiziert die Adressierungseigenschaften der Nachricht. Die `<Action>`-URI repräsentiert die Aktion, welche ausgeführt wird. Innerhalb des Elements `<To>` ist die URI angegeben, an welche der Request gesendet wird. (Linker, 2005)

5.3.2.3.1.2 SOAP-Request (Bodyteil)

Tabelle 21 Body des SOAP-Requests (Sundaram, 2005a, S. 22)

```

<SOAP-ENV:Body>
<wsrp:GetResourceProperty
xmlns:satProp="http://example.com/satellite">
satProp:altitude
</wsrp:GetResourceProperty>
</SOAP-ENV:Body>

```

Innerhalb der Standardoperation `GetResourceProperty` wird der Name der `Resourceproperty` angegeben, über dessen Wert der Requester informiert werden will.

5.3.2.3.1.3 SOAP-Response (Header)

Der Header-Bereich enthält die URI des Clients, welcher die Response erhalten soll. Meistens handelt es sich um jenen Client, welcher auch den Request ausgeführt hat. Für den Fall, dass die Antwort an eine andere URI gesendet werden soll, wäre das WS-Adressing spezifische Element `<wsa:Reply-To>` zu verwenden.

Tabelle 22 Header des SOAP Response (Sundaram, 2005a, S. 23)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WSResourceProperties/
GetResourcePropertyResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/myClient
</wsa:To>
</SOAP-ENV:Header>
```

5.3.2.3.1.4 SOAP-Response (Body)

Tabelle 23 Body des SOAP-Response (Sundaram, 2005a, S. 23)

```
<SOAP-ENV:Body>
<wsrp:GetResourcePropertyResponse
xmlns:satProp=
"http://example.com/satellite">
<satProp:altitude>
47700
</
satProp:altitude>
</wsrp:GetResourcePropertyResponse>
</SOAP-ENV:Body>
```

Der Body enthält den eigentlichen Payload. In diesem Fall wird der Wert des ResourceProperty `<altitude>` zurückgeliefert.

5.4 Fehlerbehandlung

Die Tatsache, dass unterschiedliche Implementierungen auf verschiedene Art und Weise mit Fehlermeldungen umgehen, kann generell zu Problemen führen. Um einen Dschungel an unterschiedlichen Fehlermeldungen zu vermeiden, ist eine einheitliche Regelung wünschenswert. Dadurch können auftretende Fehler auf Seiten des Requesters leichter gedeutet werden und der Aufwand für diesbezügliche Entwicklungsarbeit kann verringert werden.

Die Spezifikation `WS-BaseFaults` definiert einen XML-Schema Typ für standardisierte Fehlermeldungen und stellt Regeln für die Benutzung dieses

Typs zur Verfügung. Des weiteren können in einfacher Weise Ableitungen eigener Fehlermeldungen von diesem Standardtyp durchgeführt werden. Die Bindung hierzu erfolgt an die Operationen eines `<portType>`-Elements in der WSDL-Datei. Eigene Fehlermeldungen können in einfacher Weise von diesem Standardtyp abgeleitet werden und in einer WSDL-Datei an die Operationen eines `<portType>` gebunden werden. (Sprengel, 2006)

5.4.1 BaseFault-Funktionsweise (Liu & Meder, 2006, S. 6)

Folgende Subelemente können innerhalb des `BaseFaults` angegeben sein.

5.4.1.1 Timestamp

Das Element gibt jene Zeit an, an welcher der Fehler aufgetreten ist. Es muss ein Zeitstempel-Element vorhanden sein.

5.4.1.2 OriginatorReference (optional)

Das Element gibt die `WS-Addressing EndpointReference` des Webservices an, welches den Fehler erzeugt hat.

5.4.1.3 ErrorCode (optional)

Dieses Element stellt eine bequeme Unterstützung für die Fehlerberichterstattung von Legacyanwendungen (Altsystemen) zur Verfügung. Das `Dialekt`-Attribut auf dem `ErrorCode` muss eine URI sein, welche den Kontext definiert, in dem der `ErrorCode` interpretiert wird.

5.4.1.4 Description (optional)

Das Element enthält eine reine Sprachen Beschreibung des Fehlers. Diese Beschreibung ist für Benutzer sinnvoll, um den Fehler zu erklären.

5.4.1.5 FaultCause (optional)

Das Element ist ein `BaseFault`, welcher den darunterliegenden Grund dieses Fehlers beschreibt. Eine jegliche Anzahl an `FaultCause`-Elementen ist möglich.

5.4.1.6 Beispiel eines BaseFaults

[Tabelle 24 Beispiel für einen BaseFault \(Sundaram, 2005a, S. 38\)](#)

<code><BaseFault></code>

```
<Timestamp>2005-2-15T03:24:57</Timestamp>
<OriginatorReference>
<wsa:EndpointReference xmlns:wsa=
"http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatelliteId>
SAT9928
</sat:SatelliteId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</OriginatorReference>
<ErrorCode dialect=
"http://www.example.com/NeutronErrorMessages">
PolarityUnchangedError
</ErrorCode>
<Description>
Unable to reverse the polarity of the neutron flow!
</Description>
</BaseFault>
```

5.4.2 Erzeugung von neuen Fehlern

Prinzipiell kann ein `BaseFault` so eingesetzt werden, wie es in seiner Definition ursprünglich festgeschrieben ist. Das Parsen der Information und die Problembehandlung liegt dann in der Verantwortung der Clientapplikation.

Es ist jedoch eher so, dass der `BaseFault` erweitert wird, um mehrere spezifische Fehlertypen zu erzeugen. Um den neuen Typ zu generieren, wird ein XML Schema deklariert. (Sundaram, 2005a, S. 38)

Genauso wie alle anderen Nachrichten müssen auch Fehlernachrichten in der WSDL-Beschreibung definiert werden. Diese werden u.a im `operations-`Element des `<portTypes>` deklariert. (Sundaram, 2005a, S. 38)

5.4.2.1 Extension Mechanismus

Der Mechanismus `<extension>` erweitert den Fehler mit einem geeigneten Namen und kann spezifische Informationen hinzufügen. Ein neuer `<SatelliteNoRespondingFault>` kann als Antwort auf einen Request verwendet werden:

5.4.2.1.1 XML Schema für die neue Typ-Erzeugung

[Tabelle 25 XML-Schema für die Erweiterung von Fehlertypen \(Sundaram, 2005a, S. 47\)](#)

```

<xsd:complexType
name="SatelliteNotRespondingFaultType">
<xsd:complexContent>
<xsd:extension base="wsbf:BaseFaultType">
<xsd:sequence>
<xsd:element name="LastReboot"
type="xsd:dateTime" />
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>

</xsd:complexType>
<xsd:element name="SatelliteNotRespondingFault"
type="wsrl:SatelliteNotRespondingFaultType"/>

```

5.4.2.1.2 Fault-Return

Ein möglicher retournierter Fehler könnte demnach folgendes Aussehen haben:

Tabelle 26 XML-Schema für retournierten Fehler (Sundaram, 2005a, S. 47)

```

<SatelliteNotRespondingFault>
<Timestamp>2005-2-15T03:24:57</Timestamp>
<Description>
Cannot connect to satellite
</Description>
<LastReboot>2005-2-10T09:43:02</LastReboot>
</SatelliteNotRespondingFault>

```

Dem Client wurde hierbei zusätzliche Information bereitgestellt, welche im definierten Element `<LastReboot>` angegeben wird.

5.4.2.2 Fehlerdefinition im Zusammenhang mit WSRF-Spezifikationen

Es gibt noch eine Reihe weiterer Fehlermeldungen, welche in den jeweiligen WSRF-Spezifikationen definiert werden. Die Operation `SetResourceProperties` definiert beispielsweise den Fehler `UnableToModifyResourceProperty`. Dieser wird eingesetzt wenn eine `WS-Resource` das Abarbeiten dieser Operation nicht durchführen kann.

5.4.2.3 Fehlermeldungen *WS-ResourceProperties*

Die Spezifikation `ResourceProperties` definiert folgende Fehlermeldungen die als Antwort auf entsprechende Fehlerereignisse gesendet werden (Sundaram, 2005a, S. 39):

- `ResourceUnknownFault`
- `InvalidResourcePropertyQName`
- `InvalidSetResourcePropertiesRequestContent`

- UnableToModifyResourceProperty
- SetResourcePropertyRequestFailed
- UnknownQueryExpressionDialect
- InvalidQueryExpression
- QueryEvaluationError

5.4.2.4 Fehlermeldungen WS-ResourceLifetime

Die Spezifikation `WS-ResourceLifetime` definiert Fehler, welche die unmittelbare oder geplante Zerstörung einer `WS-Resource` betreffen (Sundaram, 2005a, S. 39):

- ResourceUnknownFault
- ResourceNotDestroyedFault
- UnableToSetTerminationTimeFault
- TerminationTimeChangeRejectedFault

5.4.2.5 Fehlermeldungen WS-ServiceGroup

Fehlermeldungen im Kontext mit der Spezifikation `WS-ServiceGroup` sind (Sundaram, 2005a, S. 39):

- UnsupportedMemberInterfaceFault
- ContentCreationFailedFault
- AddRefusedFault

5.5 Lebenszyklus von WS-Ressourcen

5.5.1 Allgemeines

`WS-Ressourcen` besitzen Lebenszeiten. Die Lebenszeit einer `WS-Resource` ist definiert als die Periode zwischen seiner Instanziierung und seiner Zerstörung. Die `WS-ResourceLifetime` Spezifikation gibt einen Standard vor, auf welche Art und Weise eine `WS-Resource` zerstört werden kann.

Die Spezifikation definiert ebenso, wie die Lebenszeit einer `WS-Resource` überwacht werden kann. Die Spezifikation schreibt jedoch nicht die Mittel vor, wodurch eine `WS-Resource` erzeugt wird. Normalerweise, ist das Interesse

eines Service Requestors für eine `WS-Resource` im bezug auf eine Zeitperiode eher undefinierbar.

In vielen Szenarien, ist es für Clients einer `WS-Resource` passend, seine sofortige Zerstörung zu veranlassen. Die sofortige Zerstörung einer `WS-Resource` kann erreicht werden, indem man die in dieser Spezifikation definierten Nachrichtenmuster verwendet. Zusätzlich, werden im Rahmen dieser Spezifikation auch Standardwege definiert, wie eine `WS-Resource` nach einer bestimmten Zeitperiode zerstört werden kann.

In einer verteilten Umgebung kann es sein, dass ein Client von dem Service-Provider Endpunkt getrennt wird, und er somit keine Fähigkeit mehr hat eine sofortige Zerstörung zu veranlassen. Diese Spezifikation definiert einen Standardweg, wodurch jeglicher Client einer `WS-Resource` die geplante Abbruchszeit einer `WS-Resource` errichten und erweitern kann. Wenn diese Zeit abläuft, dann kann sich die Ressource selbstzerstören ohne dass es notwendig ist, eine explizite Zerstörungsnachricht vom Client zu senden. (Frey, et al., 2004)

Der Lebenszyklus einer lässt sich in drei Abschnitte unterteilen (Sprengel, 2006):

- die Erstellung,
- die Identifier-Zuweisung zu einer `WS-Resource` und
- die Zerstörung.

5.5.2 Erstellung der `WS-Resource`

Wie bereits zuvor erwähnt gibt das WSRF für die Erzeugung der `WS-Resource` keine einheitlichen Richtlinien vor.

Der `factory pattern` gibt einen Weg vor, wobei mit Hilfe einer `WS-Resource factory` die Erstellung von einer oder mehrerer `WS-Resources` durchgeführt werden kann. In den meisten Fällen ist diese `WS-Resource factory` ein Web Service, welches die Ressource erzeugt, der Ressource eine Identität zuweist und die Beziehung zwischen der Ressource und dem verwaltendem Webservice herstellt. Die `WS-Resource` liefert als Ergebnis eine qualifizierte `endpoint reference` zurück. (Sprengel, 2006)

5.5.2.1 SOAP-Request: create

Der Erstellungsvorgang einer `WS-Resource` kann am Beispiel des bereits bekannten Satellitensystems erklärt werden (Die Definition der einfachen Operation `createSatellite` in WSDL wird diesbezüglich vorausgesetzt.).

Tabelle 27 SOAP-Request create (Sundaram, 2005a, S. 20)

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<createSatellite xmlns=
"http://example.com/satellite"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.5.2.2 SOAP-Response: Rücksendung der endpoint reference

In den WSDL-Nachrichtenmustern ist deklariert, dass als Antwort auf den Request eine Reference auf die neue `WS-Resource` zurückgesendet wird. (Banks, 2006, S. 19)

Die dazugehörige SOAP-Antwort enthält die *endpoint reference* für die erzeugte *WS-Resource*:

Tabelle 28 SOAP-Response der endpoint refernce (Banks, 2006, S. 20)

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Header/>

<SOAP-ENV:Body>
<wsa:EndpointReference
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite</
wsa:Address>
<wsa:ReferenceProperties>
<sat:SatelliteId>
SAT9928</sat:
SatelliteId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nach der erfolgten Erzeugung der `WS-Resource`, besitzt diese eine eindeutige Identität. Die Identität wird durch die `WS-Resource qualified endpoint`

`reference` dargestellt. Diese besteht aus der Identität der Ressource und der Adresse des Web Services. Es lassen sich folgende zwei Sichtweisen in Bezug auf die Identität der `WS-Resource` feststellen (Sprengel, 2006):

1. die Sicht der Service Implementierung auf die `WS-Resource`: Der Service Implementierung ist die Identität der Ressource bekannt und kann mit dieser operieren.
2. die Sicht des Service Nutzers (Clients) , welchem eine EPR zu einer `WS-Resource` zurückgegeben wird. Der Ressourcen Identifikator kann nicht interpretiert werden. D.h. es ist nicht möglich, die Identität der Ressource zu bestimmen.

5.5.3 Zerstörung der WS-Resource

Die Zerstörung der Ressource steht am Ende des Lebenszyklus und wird durch die Spezifikation `WS-Resource Lifetime` beschrieben. Im Gegensatz zur `WS-Resource`-Erzeugung stellt das WSRF hierfür einheitliche Nachrichtentypen zur Verfügung.

Es gibt zwei Mechanismen für die Terminierung (Beendigung) einer Ressource (Sprengel, 2006):

- **immediate termination**: die `WS-Resource` wird nach Erhalt der `destroy`-Nachricht sofort zerstört.
- **scheduled termination**: die Zerstörung der `WS-Resource` erfolgt erst nach Ablauf einer bestimmten Zeit. Dies erfolgt durch das Senden einer Nachricht des Typs `<setTermination Time>`.

Designer haben die Freiheit entweder eine von beiden oder beide dieser Mechanismen zu verwenden, um zu spezifizieren, wie deren Ressourcen zerstört werden können. Diese Entscheidung wird in der `portType` Definition (Interface) des Webservice festgelegt. (Graham, et al., 2004, S. 434)

5.5.3.1 Immediate Termination

Eine Ressource, die Teil des `Implied Resource Pattern` ist, kann eine `Message (DestroyRequest)` an das Service senden, um die Ressource sofort zu zerstören. (Joseph, Ernest, & Fellenstein, 2004, S. 636)

5.5.3.1.1 Destroy Operation

Das Format der Destroy Request-Message ist: `<wsrl:Destroy/>`

Wenn die WS-Resource die Destroy Request-Message akzeptiert, dann können die folgende beiden Fälle eintreten:

Fall 1 – Rückgabe der DestroyResponse:

Die Implied Stateful Ressource-Komponente der WS-Resource wird zerstört und die folgende DestroyResponse-Message retourniert:

```
<wsrl:DestroyResponse />
```

Fall 2 – Anzeige einer Fehlermeldung:

Eine der folgenden Fehler Messages wird retourniert:

- ResourceUnknownFault:

Die stateful Resource, die in der Message identifiziert wird und dem Implied Resource Pattern folgt, ist dem Webservice nicht bekannt.

- ResourceNotDestroyedFault:

Aus einem bestimmten Grund kann die WS-Resource nicht zerstört werden.

5.5.3.1.1.1 SOAP-Request (Header- und Body-Teil)

Um die WS-Resource zu zerstören, wird im zugehörigen SOAP-Request die Endpoint Referenez angegeben.

Tabelle 29 SOAP-Request für die Zerstörung einer WS-Resource(Sundaram, 2005b, S. 6)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-Resour
ceLifetime/Destroy
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatelliteId>SAT9928</sat:SatelliteId>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsrl:Destroy />
```

5.5.3.1.1.2 SOAP-Response (Header- und Body-Teil)

Die Bestätigung des Requests wird zurückgesendet.

Tabelle 30 SOAP-Request für die Zerstörung einer WS-Resource(Sundaram, 2005b, S. 7)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-Resourc
eLifetime/DestroyResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/myClient
</wsa:To>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsrl:DestroyResponse />
</SOAP-ENV:Body>
```

5.5.3.2 Scheduled Termination

Ein alternativer Ansatz einer WS-Resource-Terminierung wird Scheduled Termination genannt. Dabei handelt es sich um einen zeit-basierten Mechanismus, um die Lebenszeit einer WS-Resource zu managen. Die Erzeugung einer WS-Ressource erfolgt mittels einer „initialen termination time“. Das ist jene Zeit in der Zukunft, welche markiert, wann die WS-Resource beendet wird. (Graham, et al., 2004, S. 436)

Entitäten in dem System, sowie User-Applikationen, welche das Arbeiten mit der Ressource beibehalten wollen, werden periodisch versuchen die Termination Time zurückzusetzen, um die Lebenszeit der WS-Resource zu erweitern. (Graham, et al., 2004, S. 436)

Wenn die Termination Time abläuft, weil keine andere Entität an der Erweiterung der Lebenszeit interessiert ist, dann wird die WS-Ressource zerstört. Scheduled Termination ist etwas komplizierter als die immediate Termination. (Graham, et al., 2004, S. 436)

Um die Lebenszeit in dieser zeitbasierten Art und Weise zu managen, muss der Webservice zwei Resource Property-Elemente (CurrentTime , TerminationTime) und eine Operation (SetTerminationTime) implementieren.

5.5.3.2.1 ResourceProperties für die ScheduldTermination

5.5.3.2.1.1 ResourceProperty CurrentTime

Das Resource Property-Element `CurrentTime` erlaubt es den Requestoren die akute Zeit bezüglich einer bestimmten WS-Resource abzufragen (durch Verwendung der WS-ResourceProperties Operationen wie `GetResourceProperty`)

Dieses Resource Property hilft den Requestoren zu verstehen, wieviel Zeit noch übrig bleibt bevor es geplant ist eine WS-Resource zu beenden. Man beachte, dass das Resource Property `<wsrl:CurrentTime>` als `read-only` deklariert ist. Es kann also nicht durch die Operation `SetResourceProperties` gesetzt werden. Die Zeiten in der `WS-ResourceLifetime` folgen dem `dateTime`-Type, welches im XML Schema definiert ist. Interpretiert wird die Zeit im universal time format (UTC). (Graham, et al., 2004, S. 436)

Das Format dieses Resource Property Elements ist:

```
<wsrf-rl:CurrentTime>xsd:dateTime</wsrf-rl:CurrentTime>
```

5.5.3.2.1.2 ResourceProperty TerminationTime

Das Resource Property-Element `TerminationTime` erlaubt es den Requestoren, abzufragen wann es geplant ist, dass die WS-Resource zerstört wird.

Dieses ist jenes Resource Property, welches upgedatet wird, wenn die Requestoren versuchen die Termination Time der WS-Resource zurückzusetzen. Wenn die Termination Time abläuft, dann kann die WS-Resource durch ihre hosting environment zerstört werden. Das Timing sowie die Art und Weise der Zerstörung ist implementationsabhängig.

Einige Ressourcen haben den Wert `<xsi:nil>` für ihre Termination Time angegeben. Dieser Wert zeigt an dass die WS-Resource eine unbegrenzte Lebenszeit besitzt. Es kann durch eine zeitbasierten Mechanismus nicht zerstört werden. Die Ressource kann nur mittels der Operation `immediate termination` zerstört werden. (Graham, et al., 2004, S. 436)

Das Format des Resource Property-Element ist:

```
<wsrf-rl:TerminationTime
xsi:nil="xsd:boolean"?>xsd:dateTime</wsrf-rl:TerminationTime>
```

5.5.3.2.1.3 Beispiel-Listing

Beispielhaft sind nachfolgend die oben genannten Elemente im ResourcePropertiesDocument des Satellitensystems angegeben:

5.5.3.2.1.3.1 ResourcePropertiesDocument

Tabelle 31 Elemente des ResourcePropertiesDocument des Satellitensystems (Sundaram, 2005b, S. 7)

```
<satProp:GenericSatelliteProperties
xmlns:satProp="http://example.com/satellite"
xmlns:counterProp="http://example.com/satellite/CounterSatelliteProperties"
xmlns:wsrf="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd">
<satProp:latitude>30.3</satProp:latitude>
<satProp:longitude>223.2</satProp:latitude>
<satProp:altitude>47700</satProp:altitude>
<satProp:pitch>49</satProp:pitch>
<satProp:yaw>0</satProp:yaw>
<satProp:roll>32</satProp:roll>
<satProp:focalLength>
21999992
</satProp:focalLength>
<satProp:currentView>
http://example.com/satellite/2239992333.zip
</satProp:currentView>
<counterProp:currentCount>
92828
</counterProp:currentCount>
<wsrf:TerminationTime>
2005-12-31T12:00:00
</wsrf:TerminationTime>
<wsrf:CurrentTime>2005-2-15T03:24:57</CurrentTime>
</satProp:GenericSatelliteProperties>
```

5.5.3.2.1.3.2 SOAP-Request (Header- und Body-Teil)

Die ResourceProperties CurrentTime und TerminationTime können nun mit folgendem Request abgefragt werden:

Tabelle 32 Header-Teil (Sundaram, 2005b, S. 7)

```
<SOAP-ENV:Header>

<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-ResourceProperties/GetMultipleResourceProperties
```

```

</wsa:Action>

<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>

<sat:SatelliteId>SAT9928</sat:SatelliteId>

</SOAP-ENV:Header>

```

Tabelle 33 Body-Teil (Sundaram, 2005b, S. 7)

```

<SOAP-ENV:Body>
<wsrp:GetMultipleResourceProperties>
<wsrp:ResourceProperty>
wsrl:CurrentTime
</wsrp:ResourceProperty>
<wsrp:ResourceProperty>wsrl:TerminationTime</wsrp:ResourceProperty>
</wsrp:GetMultipleResourceProperties>
</SOAP-ENV:Body>

```

5.5.3.2.1.3.3 SOAP-Response (Header- und Body-Teil)

An den Client wird folgender Response mit den entsprechenden Informationen gesendet:

Tabelle 34 Header-Teil (Sundaram, 2005b, S. 8)

```

<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-ResourceProperties/GetMutlipleResourcePropertiesResponse
</wsa:Action>

<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/myClient
</wsa:To>
</SOAP-ENV:Header>

```

Tabelle 35 Body-Teil (Sundaram, 2005b, S. 8)

```

<SOAP-ENV:Body>
<wsrp:GetMultipleResourcePropertiesResponse>
<wsrl:CurrentTime>
2005-2-15T03:24:57
</wsrl:CurrentTime>
<wsrl:TerminationTime>
2005-2-15T03:30:00
</wsrl:TerminationTime>
</wsrp:GetMultipleResourcePropertiesResponse>
</SOAP-ENV:Body>

```

5.5.3.2.2 Operation SetTermination

Um die `TerminationTime` der `WS-Resource` zurückzusetzen, ist es notwendig dass der `Requestor` die `SetTerminationTime` Operation verwendet. (Graham, et al., 2004, S. 436)

Zum Festlegen einer neuen `TerminationTime` sendet der `Client` folgenden Request an den `Webservice`.

Tabelle 36 SOAP-Request für eine neue `TerminationTime` (Sundaram, 2005b, S. 9)

```
<SOAP-ENV:Header>

<wsa:Action>
http://docs.oasis-open.org/wsr/2004/06/WS-ResourceLifetime/SetTerminationTime
</wsa:Action>

<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatelliteId>SAT9928</sat:SatelliteId>

</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsr:SetTerminationTime>
<wsr:RequestedTerminationTime>
2005-2-15T07:24:00
</wsr:RequestedTerminationTime>
</wsr:SetTerminationTime>
</SOAP-ENV:Body>
```

Wird der Request akzeptiert dann enthält der zugehörige Response die neue `TerminationTime` und die `CurrentTime`.

Tabelle 37 SOAP-Response für eine neue `TerminationTime` (Sundaram, 2005b, S. 10)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsr/2004/06/WS-ResourceLifetime/SetTerminationTimeResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/myClient
</wsa:To>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsr:SetTerminationTimeResponse>
<wsr:NewTerminationTime>
2005-2-15T07:24:00
```

```
</wsrl:NewTerminationTime>
<wsrl:CurrentTime>
2005-2-15T03:25:08
</wsrl:CurrentTime>
</wsrl:SetTerminationTimeResponse>
</SOAP-ENV:Body>
```

5.5.3.3 Ressourcen-Zerstörung im Zusammenhang mit WSN

WS-ResourceLifetime standardisiert ebenso die Möglichkeit WS-Notification zu verwenden, um interessierte Teilnehmer über die Beendigung einer WS-Resource zu informieren. (Graham, et al., 2004, S. 440)

Eine WS-Resource kann also den sogenannten Pattern of notifying interested parties unterstützen. Dazu wird die Spezifikation WS-Notification verwendet, um diesen Pattern zu implementieren. Wenn die Ressource gemäß der WS-BaseNotification-Spezifikation ein NotificationProducer ist, dann sollte ein <topic> (WS-Topics) bereitgestellt werden, um Requestoren zu erlauben die Benachrichtigung bezüglich der Zerstörung zu abonnieren. Die Notification ist sowohl auf die immediate und die scheduled destruction anwendbar. (Sundaram, 2005b, S. 17)

5.6 WS-Ressourcen Gruppierung

Fallweise kann es sein, dass unterschiedliche WS-Ressourcen und Webservices domän-spezifisch organisiert werden müssen. Daraus leitet sich das Bedürfnis ab, die WS-Ressourcen zu Kollektionen zusammenzufassen. (Sprengel, 2006)

Wenn eine Applikation gross und komplex wird, dann kann es vorteilhaft sein die vorhandenen Ressourcen zu gruppieren, um beispielsweise den Zugang für gewisse Benutzergruppen zu beschränken. WSRF hat eine Lösung für diese Problemstellung und stellt deshalb die Spezifikation ServiceGroups zur Verfügung. (Sundaram, 2005b)

Generell versteht man unter ServiceGroup eine Art Verzeichnis, welches aus sogenannten ServiceGroupEntries besteht. Jeder Entry besitzt eine EPR und verweist auf einen Webservice bzw. eine WS-Resource. Des weiteren ist

auch eine Kurzbeschreibung angegeben. Eine `ServiceGroup` ist selbst wiederum eine `WS-Resource`. (Sprengel, 2006)

5.6.1 Inhalt der Spezifikation

Die Spezifikation stellt `Messages Exchanges` zur Verfügung, welche die Schnittstellen `ServiceGroup`, `ServiceGroupEntry` und `ServiceGroupRegistration` definiert. (Maguire, Snelling, & Banks, 2006, S. 10)

- **ServiceGroup:**

Eine `ServiceGroup` ist eine `WS-Resource`, die sich gemäß dem `Implied Resource Pattern` verhält und eine Sammlung zu anderen Webservices darstellt.

- **Member:**

`Members` bzw. `ServiceGroup Members` werden jene individuellen `Services` genannt, die einer Servicegruppe zugehörig sind.

- **ServiceGroupEntry:**

Ein Eintrag in einer `ServiceGroup` assoziiert einen `Member` mit einer `ServiceGroup`. Ein `ServiceGroupEntry` beinhaltet auch Informationen, mit welcher die Teilnahme von `Members` in der `ServiceGroup` angezeigt wird.

- **ServiceGroupRegistration:**

Die Registrierung stellt Standardwege zur Verfügung, um es Benutzern zu ermöglichen neue `Members` einzufügen.

Die nachfolgende Grafik veranschaulicht, wie eine `ServiceGroup` zugehörige Webservices verwaltet (die Grafik beinhaltet aus Gründen der Vollständigkeit auch das `Member-Interface`, welches allerdings nicht Teil der `WS-ServiceGroup` Spezifikation ist):

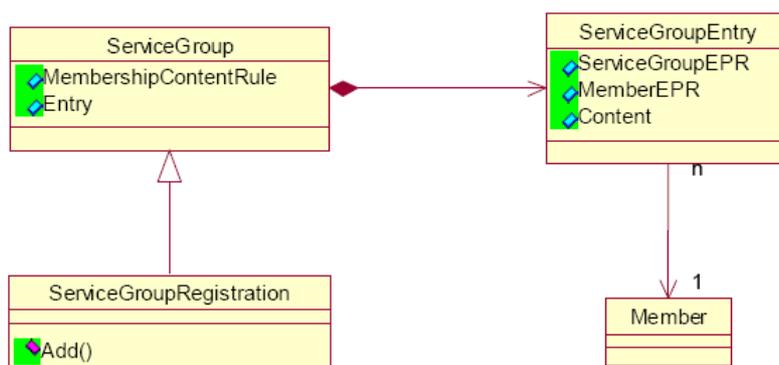


Abbildung 27 ServiceGroup (Maguire, Snelling, & Banks, 2006, S. 11)

5.6.2 ServiceGroup

Wie bereits erwähnt stellt die `ServiceGroup` eine WS-Resource dar. Dh. es handelt sich um eine Kombination aus einem Webservice und einer `stateful resource`. In diesem Fall wird diese `stateful resource` durch die Ansammlung der `Entrys` dargestellt, welche die `ServiceGroup` enthält. So wie jede andere WS-Resource ist eine `Servicegroup` durch eine `Endpoint Reference` identifizierbar und es können ebenso die gleichen Operationen angewendet werden.

Das für die erzeugte `ServiceGroup` vorhandene `Resource Property Document` enthält die Elemente `MembershipContentRule` und `Entry`.

5.6.2.1 Erzeugung einer ServiceGroup

Mit einer `Create-Nachricht` (`createSatGroup`) kann die WS-Resource erzeugt werden, worauf der Server wiederum eine `Endpoint-Reference` an den Requester zurückliefert. Nachfolgend wird das `Servicegroup-Szenario` am Beispiel des Satellitensystems erklärt. Als `Resource Identifier` wird der Wert `<SATGRP3>` verwendet.

Tabelle 38 SOAP-Request für eine neue ServiceGroup (Sundaram, 2005b, S. 15)

```

<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsr/2004/06/WS-Resour
ceLifetime/SetTerminationTimeResponse
</wsa:Action>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
  
```

```

<SOAP-ENV:Body>
<create SatGroup
xmlns="http://example.com/satellite"/>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

```

Die zugehörige Response-Nachricht enthält die Endpoint Reference für die neue WS-Resource.

Tabelle 39 SOAP-Response für eine neue ServiceGroup (Sundaram, 2005b, S. 10)

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>

<SOAP-ENV:Body>
<createSatGroupResponse
xmlns="http://example.com/satellite">
<wsa:EndpointReference
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat=
"http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatGroupId>
SATGRP3
</sat:SatGroupId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</createSatGroupResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

5.6.2.2 MembershipContentRule

Die MembershipContentRule Elemente definieren die Kriterien, welche eine ServiceGroupEntry zu erfüllen hat, bevor diese zu einer ServiceGroup hinzugefügt werden kann. (Sundaram, 2005b, S. 16)

Somit können also die Beschränkungen bezüglich der Servicegroup-Mitgliedschaft angegeben werden. Folgende zwei Attribute werden beschrieben:

- Das Attribut MemberInterface dient dazu, die Mitgliedschaft auf jene Teilnehmer (Members) zu begrenzen, die bestimmte <portTypes> implementieren.

- Das Attribut `ContentElements` fordert das Vorhandensein von bestimmten Child-Elementen in der `ServiceGroupEntry`.

Tabelle 40 Format des `MembershipContentRule` resource property Elements (Maguire, Snelling, & Banks, 2006, S. 13)

```
<wsrf-sg:MembershipContentRule
  MemberInterfaces="list of QName"?
  ContentElements="list of QName"
 />
```

Im Resource Properties Document der `ServiceGroup` können beispielsweise folgende drei `MembershipContentRules` angegeben werden.

Tabelle 41 Beispiel-Listing RPD der `ServiceGroup` (Sundaram, 2005b, S. 21)

```
<wssg:MembershipContentRule
  MemberInterface="sat:AuditedSatellitePortType" />
<wssg:MembershipContentRule
  MemberInterface="sat:CounterPortType"
  ContentElements="sat:audited" />
<wssg:MembershipContentRule
  ContentElements="sat:approvedBy" />
```

5.6.2.3 Entry

Jeder Entry identifiziert die `ServiceGroupEntry`-Ressource und stellt einen Zeiger auf das konkrete Service bereit. (Sundaram, 2005b, S. 21)

Eine Entry Resource Property hat die folgende Struktur:

Tabelle 42 Struktur einer Entry Resource Property (Maguire, Snelling, & Banks, 2006, S. 13)

```
<wssg:Entry>

  <wssg:ServiceGroupEntryEPR>
  wsa:EndpointReferenceType
  </wssg:ServiceGroupEntryEPR>

  <wssg:MemberServiceEPR>
  wsa:EndpointReferenceType
  </wssg:MemberServiceEPR>

  <wssg:Content> {any} </wssg:Content> ?
</wssg:Entry>
```

Zur Veranschaulichung ist beispielhaft folgender Entry-Eintrag angeführt (natürlich enthält ein RPD in der Regel mehrere Entry-Elemente. Aus Übersichtsgründen ist nur ein Auszug des Listings dargestellt):

Tabelle 43 Beispiel-Listing RPD (Sundaram, 2005b, S. 17)

```

<wssg:Entry>
<wssg:ServiceGroupEntryEPR>
<wsa:EndpointReference
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatGroupEntryId>
SATGRPENTRY28981
</sat:SatGroupEntryId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</wssg:ServiceGroupEntryEPR>

<wssg:MemberServiceEPR>
<wsa:EndpointReference
xmlns:wsa=
ttp://www.w3.org/2005/02/addressing"
xmlns:sat=
"http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatelliteId>
SAT9928
</sat:SatelliteId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</wssg:MemberServiceEPR>

<wssg:Content>
<sat:approvedBy>
BossManBing
</sat:approvedBy>
</wssg:Content>

</wssg:Entry>

```

5.6.3 ServiceGroupEntry

Eine `ServiceGroupEntry` ist ebenso eine WS-Resource. Dies bedeutet, dass es auch ein RPD vorhanden ist, welches den Zustand und eine WS-Adresse enthält. Eine `ServiceGroupEntry` definiert die Beziehung zwischen einer `ServiceGroup` und einem Service. Es kann auch optionaler Content enthalten sein. (Graham, et al., 2004, S. 14)

5.6.3.1 Struktur des RPD

Der Aufbau des Resource Property-Elements besteht aus folgenden Elementen:

- ServiceGroupEPR
- MemberEPR
- Content

Die ServiceGroupEPR ist die Endpoint Reference für die ServiceGroup. Die MemberServiceEPR ist eine Referenz auf das Webservice bzw. die WS-Resource, auf welche sich die ServiceGroupEntry bezieht. Im Content-Element ist zusätzliche Information enthalten (Approval Code). (Sundaram, 2005b, S. 17)

Tabelle 44 Beispiel Listing eines RPD der ServiceGroupEntry (Sundaram, 2005b, S. 16)

```
<wssg:ServiceGroupEntryRP>
<wssg:ServiceGroupEPR>
<wsa:EndpointReference
xmlns:wsa=
"http://www.w3.org/2005/02/addressing"
xmlns:sat=
"http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatGroupId>
SATGRP3
</sat:SatGroupId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</wssg:ServiceGroupEPR>
<wssg:MemberEPR>
<wsa:EndpointReference xmlns:wsa=
http://www.w3.org/2005/02/addressing"
xmlns:sat=
"http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
```

5.6.4 ServiceGroupRegistration

Das ServiceGroupRegistration-Interface ist eine Erweiterung des ServiceGroup-Interface. Die ServiceGroupRegistration definiert die Nachrichtenaustauschmuster, welche es einem Requestor erlauben, Entries

zu einer ServiceGroup WS-Ressource explizit hinzuzufügen. (Maguire, Snelling, & Banks, 2006, S. 17)

5.6.4.1 Hinzufügen einer ServiceGroup mittels der Operation Add

Ein Requestor kann beispielsweise einen Satelliten (SAT8557) zu einer ServiceGroup hinzufügen. Mittels Angabe von add wird ein Request an den Server geschickt. Um dies zu bewerkstelligen, muss er den ResourceIdentifier der ServiceGroup kennen (SATGRP3).

Tabelle 45 Header-Teil (Sundaram, 2005b, S. 20)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-ServiceGroup/Add
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatGroupId>SATGRP3</sat:SatGroupId>
</SOAP-ENV:Header>
```

Tabelle 46 Body-Teil (Sundaram, 2005b, S. 20)

```
<SOAP-ENV:Body>
<wssg:Add>

<wssg:MemberEPR>
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatelliteId>
SAT8557
</sat:SatelliteId>
</wsa:ReferenceProperties>
</wssg:MemberEPR>

<wssg:Content>
<sat:approvedBy>
BossManBing
</sat:approvedBy>
</wssg:Content>

</wssg:Add>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Das Add-Element, welches im Body des Requests angegeben wird, beinhaltet folgende Child-Elemente:

- MemberEPR
- Content
- InitialTerminationTime (optional)

Es kann ein InitialTerminationTime-Element hinzugefügt werden, um eine TerminationTime für die zurückgelieferte ServiceGroupEntry anzugeben.

Das Webservice, das mit dem ServiceGroupEntry assoziiert ist und durch AddResponse zurückgeliefert wird, muss also die Nachrichtenaustauschmuster und ebenso die resource properties implementieren, die durch das ScheduledResourceTermination-Interface und die ImmediateResourceTermination-Interface spezifiziert sind (WS-ResourceLifeTime). (Maguire, Snelling, & Banks, 2006, S. 17)

5.6.4.1.1 SOAP-Response (Header- und Body-Teil)

Wenn die ServiceGroupRegistration den Request akzeptiert um einen Teilnehmer (Member) hinzuzufügen, dann muss er mir einer AddResponse Message antworten. Der Inhalt einer AddResponse Message ist eine EndpointReference. Diese EndpointReference zeigt zu der ServiceGroupEntry WS-Resource, die von der ServiceGroup erzeugt wird, um die Assoziation des Members innerhalb der ServiceGroup zu repräsentieren. (Maguire, Snelling, & Banks, 2006, S. 18)

Tabelle 47 Header-Teil (Sundaram, 2005b, S. 20)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsrf/2004/06/WS-ServiceGroup/AddResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/myClient
</wsa:To>
</SOAP-ENV:Header>
```

Tabelle 48 Body-Teil (Sundaram, 2005b, S. 20)

```
<SOAP-ENV:Body>
<wsrp:AddResponse>
</wsa:EndpointReference>
<wsa:Address>
http://example.com/satellite
```

```
</wsa:Address>
<wsa:ReferenceProperties>
<sat:SatGroupEntryId>
SATGRPENTRY29003
</sat:SatGroupEntryId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</wsrp:AddResponse>
</SOAP-ENV:Body>
```

5.6.5 Kombinierbarkeit mit anderen Webservice-Standards

Ein mit einem `ServiceGroup Entry` assoziierter Webservice kann aus einer Vielfalt von Webservice-Standards entworfen werden. Diesbezüglich sind u.a. folgende Standards zu nennen: (Maguire, Snelling, & Banks, 2006, S. 21)

- `WS_ResourceLifetime`, mit dessen Standard Patterns WS-Ressourcen im Zusammenhang mit `WS-ServiceGroups` zerstört werden können.
- `WS_BaseNotification` gibt eine Definition vor, auf welche Art und Weise Dritt-Instanzen eine Registrierung vornehmen können, um sich über Änderungen von `ServiceGroups` informieren zu lassen. Die Webservice Komponente der `ServiceGroup WS-Resource` kann auch die `NotificationProducer Schnittstelle (WS-BaseNotification)` implementieren. Für diesen Fall, muss die Komponente ein `<Topic>` (`WS-Topics`) bereitstellen. Damit können sich Requestoren für eine `Notification` registrieren, um über Änderungen innerhalb der `ServiceGroup` informiert zu werden.
- `WS_ResourceProperties` ermöglicht es die `Properties` einer `ServiceGroup` und seiner `Entires` über eine Webservice Schnittstelle zugänglich zu machen.

Beispielsweise beinhaltet eine bestimmte `ServiceGroup` mehrere `WS-Ressourcen`, welche ein Requestor nutzen möchte. Durch Verwendung der Operation `GetResourceProperty` kann das Element `Entry` abgefragt werden. In der zurückgelieferten Antwort ist die jeweilige `MemberServiceEPR` enthalten, welche der Client verwenden kann um mit dem jeweiligen Service zu interagieren. (Sundaram, 2005b)

5.7 Geschichtlicher Hintergrund von WSRF

Die Entwicklung des WSRF ist ursprünglich aus dem Refactoring und der Evolution des gridspezifischen OGSI-Standard hervorgegangen. Diese Erweiterung war gewünscht, da den Webservices viele Fähigkeiten, welche beim Grid Computing gebraucht werden, fehlten.

Dazu zählt beispielsweise die Möglichkeit eine Ressource darzustellen und eine Lebensdauer festzulegen. OGSI führte den Begriff des *Grid Service* ein, welches eine Variante des Webservice Konzeptes bzw. Erweiterung darstellte.

OGSI wurde von der Grid-Community (GGF ,Global Grid Forum) entwickelt, mit dem Wunsch das aufkommende Phänomen der Webservices zu adaptieren und zu erweitern. Basierend auf XML Schema Definitionen und WSDL, stellte OGSI ein nützliches Werkzeug für Webservices-Entwickler zur Verfügung. Der Standard ermöglichte es, Webservices mit einem Zustand zu modellieren und auch gleichsam für Gridsysteme kompatibel zu machen. Eine Referenzimplementierung von OGSI stellt das Globus Toolkit 4 da, welches auch den WSRF Standard unterstützt. (Graham, et al., 2004, S. 386)

5.7.1 Gridkonzept

5.7.1.1 Allgemeines

Ein "Grid" bezeichnet eine nach dem Grid-Computing-Ansatz aufgebaute Rechner-, Netzwerk- und Software-Infrastruktur, welche es möglich macht, dass heterogene, lose gekoppelte und geographisch verteilte IT-Ressourcen über Organisationen hinweg kooperativ genutzt werden.

Die IT Ressourcen, welche von ihren physikalischen Grenzen ungebunden sind, werden als Services angeboten und je nach Bedarf dem Anwender zugänglich gemacht. Diese können beinahe jede IT-Komponente umfassen: Rechenleistung, Speicherplatz, Datenbanken, Anwendungen, Dateien, Sensoren oder wissenschaftliche Instrumente.

(Foster, Kesselman, Nick, & Tuecke, 2002)

Gridsysteme werden durch die Verwendung von Grid-Middleware (Globus Toolkit) ermöglicht. Es handelt sich dabei um Softwarekomponenten und Protokolle, welche den kontrollierten Zugang zu Grid-Ressourcen bereitstellen.

Grid Computing Implementationen basierten hauptsächlich bislang auf der Open Grid Services Infrastructure (OGSI).

(Foster, Kesselman, Nick, & Tuecke, 2002)

5.7.1.2 Definition

Eine frühere Definition aus dem Jahr 1998 lautet wie folgt (Kesselmann & Foster, 1999):

„A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities“

Diese frühere Definition des Gridkonzeptes aus dem Jahr 1998 lässt sich noch durch folgenden Zusatz erweitern (Foster, Kesselman, & Tuecke, 2001, S. 2):

“... coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.”

Der Unterschied zur früheren Definition liegt darin, dass die gemeinsame Ressourcennutzung durch Virtuelle Organisationen festgelegt wird.

5.7.1.3 Kriterien für ein Grid (Foster, 2002)

- Koordination von Ressourcen welche nicht unter zentralisierter Kontrolle stehen
- Einsatz von standardisierten, offenen, allgemein gültigen Protokollen und Schnittstellen
- Bereitstellung von nichttrivialen „Quality of Services“

5.7.1.4 OGSA

Definiert werden die Services in der OGSA (Open Grid Services Architecture) , welche die grundlegende service-orientierte Systemarchitektur des Grid darstellt.

Die OGSA stellt eine Definition für ein Komponentenmodell bereit, bei welchem den Applikationen die Möglichkeiten geschaffen werden, die im Grid angebotenen Dienste auf einfache Weise miteinander zu nutzen. Folgende

Thematiken werden in der Definition in Bezug auf Grid-Umgebungen behandelt (Reinefeld & Schintke, 2004, S. 295):

- Identifikation, Authentifikation, Autorisierung von Teilnehmern und Services
- Auffinden und Aufrufen von Services
- Ausverhandlung von Verfahren
- Überwachung und Kontrolle verteilter Applikationen
- Einrichtung und Unterstützung dynamischer Arbeitsgruppen (VO)

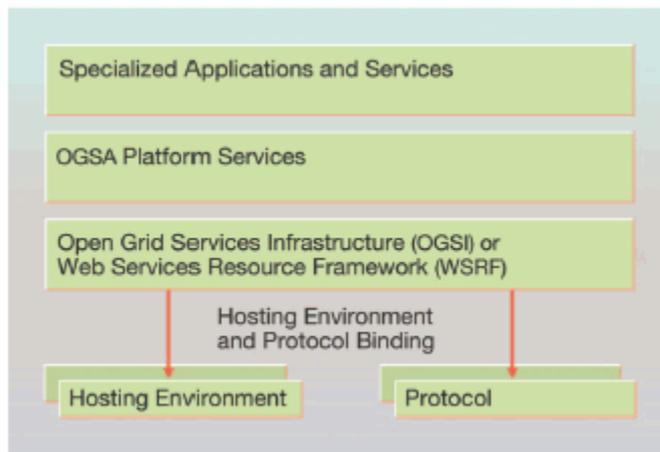


Abbildung 28 OGSA Plattform Architektur (Joseph, Ernest, & Fellenstein, 2004, S. 628)

5.7.2 OGSI

Während einerseits OGSA eine Beschreibung vorgibt, wie der allgemeine Rahmen für die Gridstruktur und die Dienstorganisation auszusehen hat, erfolgt mittels OGSI die Spezifikation der sogenannten Grid Services.

Die OGSI Spezifikation, welche in der Version 1.0 im Jahr 2003 freigegeben wurde, definiert eine Reihe an Konventionen und Erweiterungen, welche für die Verwendung von WSDL und XML Schema ausgelegt ist, und zustandsgebundene Webservices ermöglichen.

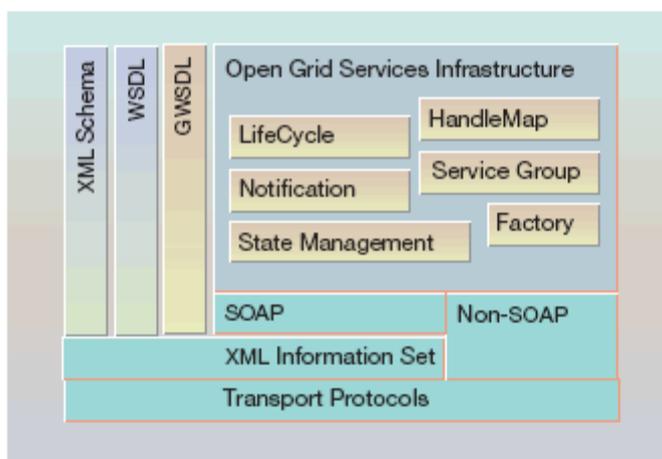


Abbildung 29 Komponenten der OGSI (Joseph, Ernest, & Fellenstein, 2004, S. 631)

5.7.2.1 OGSI-Konzept

OGSI definiert einen Ansatz (Foster, et al., 2004):

- für das Erstellen, das Benennen und die Lebensdauer-Verwaltung von Serviceinstanzen
- für das Deklarieren und Untersuchen der Zustandsdaten der Services
- für die asynchrone Benachrichtigung von Änderungen der Zustandsdaten
- für die Darstellung und das Verwalten von Sammlungen der Serviceinstanzen
- für das allgemeine Fehlerhandling bei Dienstaufrufen

5.7.2.2 Begriff Grid Services

Im Kontext mit WSRF wird ein Grid Service als ein formloses Webservice definiert, welches entworfen wurde um in einer Gridumgebung zu operieren und den Anforderungen des Grids, an dem es teilnimmt, gerecht zu werden. Grid Ressourcen werden mittels Webservices durch Verwendung des sogenannten `Implied-Resource Pattern` dargestellt und kontrolliert.

Das WSRF Framework betrachtet ein Grid-Service als ein `stateless` Webservice, welches eine `stateful resource` verwaltet. (Tsalgatidou, 2005, S. 11)

5.7.2.3 Kritik an OGSi

Es gab einige Schwachpunkte bei OGSi, welche von der Webservice-Community kritisiert wurden. Aus diesem Grund fand OGSi auch nicht die allgemein verbreitete Akzeptanz, was schließlich dazu führte, dass das WSRF entwickelt wurde.

Die Kritikpunkte waren u.a (Foster, et al., 2004, S. 606):

- dass die Spezifikation zu überladen ist, weil die Spezifikation zu lange und nicht sauber aufgeteilt ist
- dass OGSi nicht gut mit existierenden Webservices und mit XML Toolkits zusammenarbeitet, da XML zu aggressiv und WSDL nicht standardkonform verwendet werden kann
- dass die OGSi-Konzepte zu objekt-orientiert sind

5.7.3 WSRF als gridspezifischer Ansatz

Eine der Besonderheiten von WSRF und WSN ist es, dass der Standard nicht nur von der Webservice-Community, sondern auch von der Grid-Community akzeptiert wird. Es wird somit dadurch die Möglichkeit geschaffen, dass Webservices und Gridsysteme noch näher zusammenwachsen können.



Abbildung 30 WSRF als Schnittstelle zwischen Webservices und GridComputing (Foster, 2004a, S. 7)

In den letzten Jahren haben sich die Ziele von Gridsystemen mit den Vorteilen von SOA-basierten Webservices immer mehr überlappt und die Entwicklung von WSRF und WSN hat natürlich dazu viel beigetragen. Somit können die Vorteile aus beiden Bereichen zusammengeführt und die jeweiligen Stärken gemeinsam genutzt werden.

6 WSN

Generell wird in vielen Service-orientierten Architekturen (SOA) der „Request/Response“ Pattern verwendet. Dabei identifiziert ein Service Requestor einen Service, welchen er nutzen will, und sendet daraufhin eine Request Nachricht. Der Service Provider akzeptiert die Request Nachricht, verarbeitet diese, und sendet eine Response Nachricht. Dieser Pattern entspricht dem Wesen eines einfachen Prozeduraufrufes. (Niblett & Graham, 2005, S. 869)

Ein weiteres Programmiermodell, welches es an dieser Stelle zu erwähnen ist, ist das sogenannte event-basierte Programmieren („Event-based Programming“), welches in „Publish/Subscribe“ Systemen zum Einsatz kommt. User-Interface Systeme sind hierbei ein häufiges Anwendungsgebiet, indem beispielsweise Komponenten auf Benutzerinteraktionen wie beispielsweise Mausklicks oder Tastatureingaben reagieren. (Niblett & Graham, 2005, S. 869)

Event-basiertes Programmieren stellt Entitäten in den Vordergrund, welche ein Vorkommnis darstellen (ein Ereignis, das eingetreten ist). In objekt-orientierten Systemen wird hierbei von einem Ereignisobjekt (Event Object) gesprochen. (Niblett & Graham, 2005, S. 869)

In nachrichten-orientierten Systemen bezieht sich diese Entität auf eine Ereignisnachricht. Die Applikationen produzieren und konsumieren Events. Dabei existiert keine Kopplung der Events zu den Anwendungen. Ein Consumer kann für Events, an denen er interessiert ist, eine Registrierung vornehmen, um in weiterer Folge über auftretende Ereignisse benachrichtigt zu werden. Das Interaktionsmuster wird in diesem Zusammenhang „Notification Pattern“ genannt. Dabei wird eine Beziehung zu dem auftretenden Ereignis bzw. dem Event aufgebaut, und nicht direkt zum Service Provider, so wie es beim Request/Response Pattern der Fall ist. (Niblett & Graham, 2005, S. 870)

Der Ansatz, dass das Ereignis eingesetzt wird, um eine Entkoppelung zwischen dem Event Producer und Consumer vorzunehmen, gilt als signifikanter Unterschied zwischen dem „Request/Response Pattern“ und dem „Notification Pattern“. Durch dieses Entkoppeln ist es möglich, dass auch ein „1-to-n“ und „n-to-1“ Nachrichtenaustausch möglich ist. (Niblett & Graham, 2005, S. 870)

6.1 Allgemeines

6.1.1 Konzept

Die WSN-Spezifikation ist ein ergänzendes Benachrichtigungswerkzeug, welches das WSRF erweitert. Dieser asynchrone Mechanismus wird zum Auslösen von Ereignissen in IT-Infrastrukturen verwendet. Benutzer können sich zu bestimmten Interessens-Themen anmelden, um Informationen über relevante Zustandsänderungen oder Ereignisse zu erhalten. Im Bereich Grid-Computing können beispielsweise potentielle Nutzer standardisierte Infos über die Komponenten, die Verfügbarkeit und freie Kapazitäten von Grids abrufen.

So wie bei all den anderen Spezifikationen werden bei WSN spezifische Nachrichtenaustausch-Muster definiert. (`Message-Exchange-Pattern`). Im Fall von WSN kann mit diesen Mustern das Publizieren und Eintragen von Interaktionen mit dem WSRF ermöglicht werden.

Das WSN-Framework stellt Konventionen bereit, damit sich interessierte Dienste über sich ändernde Zustände oder über auftretende Events benachrichtigen lassen können (wie zB. bei Auftreten von Änderungen in den `Resource Properties`).

WSN definiert mehrere Rollen. Besonders erwähnenswert ist für die WSN-Spezifikation der `NotificationConsumer` und der `NotificationProducer`. In einem einfachen Szenario ist es so, dass der `NotificationConsumer` einen `NotificationProducer` kontaktiert, um eine `Subscription` (Abonnement) für ein bestimmtes `Topic` (Thema) anzumelden. Wenn der `NotificationProducer` über eine Nachricht zu diesem `Topic` verfügt, dann sendet er diese Nachricht an den `NotificationConsumer`. (Niblett & Graham, 2005, S. 870)

6.1.3 Begriffserklärung

Im Zusammenhang mit WSN gibt es eine Vielzahl von Begriffen, die in einem Benachrichtigungsszenario verwendet werden. Im den folgenden Teilkapiteln werden zunächst die grundlegenden Begriffe erörtert.

6.1.3.1 Situation

`WS-Notification` verwendet den Begriff `Situation`, um sich auf eine Sache zu beziehen, die von Interesse ist und welche sich ereignet hat. Es existieren viele Arten von Situationen. Dies ist der Fall, wenn sich der Zustand ändert; beispielsweise wenn eine Systemressource (bspw. ein Server) von einem Online-Zustand in den Offline-Zustand übergeht. (Graham, et al., 2004, S. 413)

WSN definiert nicht, was eine Situation sein kann oder nicht. Es wird nur festgestellt, dass eine Situation existiert, und dass es Entitäten im System gibt, welche daran interessiert sind, Nachrichten zu erhalten, wenn eine Situation auftritt. (Graham, et al., 2004, S. 413)

6.1.3.2 Publisher

Ein `Publisher` ist eine Entität, die `NotificationMessage`-Instanzen erzeugt. (Niblett & Graham, 2005, S. 872)

6.1.3.3 NotificationProducer

Ein `NotificationProducer` ist ein `Web Service`, welches die Fähigkeit hat herauszufinden, dass eine Situation aufgetreten ist. Der `NotificationProducer` erzeugt ein XML-Artefakt (eine Benachrichtigung oder `NotificationMessage`), welches wichtige Details über dieses Auftreten erfasst. (Graham, et al., 2004, S. 414)

6.1.3.4 NotificationConsumer

Ein `NotificationConsumer` ist ein `Endpoint`, dargestellt durch eine `WS-Addressing Endpoint Reference`, welcher designiert ist Benachrichtigungen zu erhalten, die durch einen `NotificationProducer` als ein Ergebnis einer `Subscription` produziert wurden. (Graham, Hull, & Murray, 2006, S. 9f)

6.1.3.5 Subscriber

Ein `Subscriber` ist im allgemeinen die Instanz, welche die Registrierung durchführt bzw. die `subscribe operation` ausführt. Ein `Subscriber` ist eine Entität, welche die `SubscribeRequest`-Nachricht zu einem `NotificationProducer` sendet. Man beachte dass ein `Subscriber` eine unterschiedliche Entität zu dem `NotificationConsumer` sein kann, für welche die `Notifications` aktuell erzeugt werden. (Graham, Hull, & Murray, 2006, S. 10)

6.1.3.6 SubscriptionManager

In den meisten Fällen handelt es sich beim `Subscriber` und beim `NotificationConsumer` um dieselbe Instanz. Aber es gibt Situationen, wo diese Rollen von verschiedenen Einheiten übernommen werden. Die Verwaltung von Registrierungen kann also von einem sogenannten Registrierungsverwalter (`SubscriptionManager`) übernommen werden. (Graham, et al., 2004, S. 413)

Ein solches Szenario könnte sein, dass beispielsweise ein Webservice eine Verkaufsauftrag in Auftrag gibt und die `subscribe operation` ausführt. Eine weitere Instanz in der Rolle des `NotificationConsumers` erhält dann Benachrichtigungen über Statusänderungen der Verkaufsauftrag. (Graham, et al., 2004, S. 413)

Ein `SubscriptionManager` ist eine WS-Resource, welche die Übersicht über eine `Subscription` behält. Das zugehörige `ResourcePropertyDocument` beinhaltet die `Endpoint Reference` des `NotificationConsumers`, die `Topics` und Lieferangaben. (Sundaram, 2005c, S. 17)

6.1.3.7 Topic

Ein `Topic` ist das Konzept, welches verwendet wird um `Notifications` und ihre zugehörigen `Notification schemas` zu kategorisieren.

(Vambenepe, Graham, & Niblett, 2006, S. 7)

6.1.3.8 Subscription

Eine `Subscription` repräsentiert eine Beziehung zwischen einem `NotificationConsumer` und dem `NotificationProducer`, wobei die Beziehung Filterparameter (ein `Topic` und weitere optionale Filterausdrücke) beinhaltet. (Graham, Hull, & Murray, 2006, S. 10)

6.1.4 Notification

Der Begriff `Notification` wird verwendet um auf eine „one-way“ Nachricht Bezug zu nehmen, welche eine Information über eine Situation an andere Services befördert. (Niblett & Graham, 2005, S. 871)

6.1.4.1 Aufbau

Das Standardformat für `Notifications` ist folgendermaßen aufgebaut. Folgende Inhalte sind in der Nachricht enthalten:

- das Thema (`Topic`), für welche die `Notification` produziert wird
- die Adresse der Nachrichtenquelle
- der Nachrichtenteil

6.1.4.2 Standardformat für eine Notification

Das Standardformat für `Notifications` bzw. die generelle Funktionsweise von WSN lässt sich anhand des bereits vorgestellten Satellitensystems erörtern. Es wird hierbei angenommen, dass zwei Ereignisse unterschieden werden.

- Die `ThresholdReached Situation` wird ausgelöst, wenn der Satellit eine bestimmte Schwelle (`Threshold`) erreicht.
- Das `CounterReset Event` wird ausgelöst, wenn die Software im Satelliten einem `Reset` unterzogen werden muss.

Desweiteren existieren zwei `NotificationConsumer` (Sundaram, 2005c, S. 5):

- ein Wissenschaftler, der nur an der `thresholdReached Situation` interessiert ist
- der Systemadministrator ,der alle `Notifications` von allen `Topics` erhalten möchte

Tabelle 49 Listing: „Notify“ – Standardformat für Notifications am Beispielszenario Satellitensystem (Sundaram, 2005c, S. 5)

```

<wsnt:Notify>
<wsntw:NotificationMessage>
<wsnt:Topic Dialect="http://docs.oasis-
open.org/wsn/2004/06/TopicExpression/Simple">
sat:thresholdReached
</wsnt:Topic>
<wsnt:ProducerReference>
<wsa:EndpointReference
xmlns:wsa=
"http://www.w3.org/2005/02/addressing"
xmlns:sat=
"http://example.org/satelliteSystem">
<wsa:Address>
http://example.com/satellite
</wsa:Address>
<wsa:ReferenceProperties>
<sat:ResourceId>
RES2883
</sat:ResourceId>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</wsnt:ProducerReference>
<wsnt:Message>
<sat:ThresholdReachedMessage>
<sat:MessageTitle>
Threshold reached
</sat:MessageTitle>
<sat:MessageContent>
The satellite has reached a new
milestone in counting stars. Please view
the current data for status.
</sat:MessageContent>
</sat:ThresholdReachedMessage>
</wsnt:Message>
</wsntw:NotificationMessage>
</wsnt:Notify>

```

6.1.4.3 Topic Deklaration

NotificationConsumer können auswählen, welche Benachrichtigung sie erhalten möchten, indem bei der Subscription das Topic angegeben wird, an welchem Interesse besteht. Die WSN-Teilspezifikation definiert hierfür die sogenannten TopicSpaces, wie folgendes Listing veranschaulicht. Hierin werden die Topics spezifiziert, um die Situationen darzustellen.

Tabelle 50 TopicSpace (Sundaram, 2005c)

```

<wstop:topicSpace name="SatelliteTopics"
targetNamespace="http://example.com/satellite"
xmlns:wstop="http://docs.oasis-open.org/wsn/2004/06/ws

```

```
n-WS-Topics-1.2-draft-01.xsd">
<wstop:topic name="counterReset" />
<wstop:topic name="thresholdReached" />
</wstop:topicSpace>
```

6.1.4.4 Nachrichtenablauf

Das folgende Diagramm veranschaulicht eine mögliche Abfolge der Operationen:

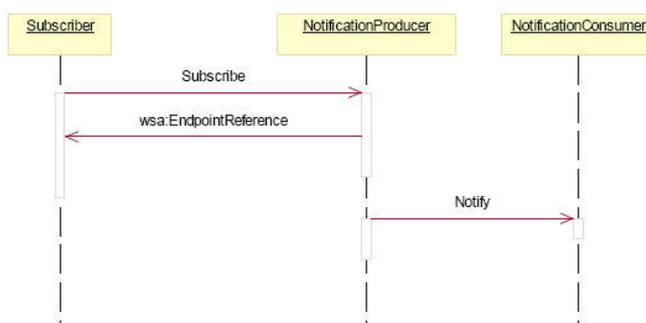


Abbildung 32 Abfolge der Operationen zwischen Subscriber, NotificationProducer und NotificationConsumer

Der Subscriber sendet eine Subscribe Request Message an den NotificationProducer. Angezeigt wird dabei die Adresse des NotificationConsumers, die Nachrichtenarten für die Subscription, und andere Subscription-bezogene Information. Als Antwort auf diese Message, erzeugt der NotificationProducer eine Subscription Resource und returniert eine EndpointReference (WS-Adressing). Einige Zeit später, bringt der NotificationProducer eine Notification hervor, welche der Subscription zugeordnet wird. Der NotificationProducer verwendet die Notify Message, um dies an den NotificationConsumer zu liefern oder sendet, falls es in der Subscription angezeigt wird, die NotificationMessage als eine applikationsdefinierte Message (ohne Verwendung des allgemeinen Notify Message Exchange). (Niblett & Graham, 2005)

6.2 WS-Base Notification

6.2.1 Allgemeines

Die WS-BaseNotification Spezifikation definiert die Konstrukte NotificationConsumer, NotificationProducer und SubscriptionManager.

Die WS-Base Notification stellt das Fundament für die gesamte WSN-Familie bereit. In dieser Spezifikation werden die grundlegenden Rollen und Message Exchanges definiert, welche verwendet werden, um den Notification Pattern auszudrücken. Die Spezifikation kann alleine für sich oder in Kombination mit der WS-Topics- und WS-Brokered-Spezifikation eingesetzt werden. (Niblett & Graham, 2005, S. 873)

6.2.1.1 Beispielszenario:

Im vorangegangenen Kapitel wurde bereits ein eher einfaches Szenario im Bezug auf eine WSN-Benachrichtigung angeführt.

Ein etwas komplizierteres Szenario ist dann gegeben, wenn der NotificationConsumer nicht direkt die Subscription durchführt. In diesem Fall führt ein sogenannter Subscriber die Subscription für ihn durch. Des weiteren kann es auch einen sogenannten Publisher geben, der alle Aufgaben des NotificationProducers übernimmt. Die Nachrichten können dann von diesem an einen NotificationBroker gesendet werden, welcher wiederum diese an die NotificationConsumer weiterleitet. (Sundaram, Understanding WSRF, Part 3, 2005c)

Die folgende Abbildung zeigt einen Subscriber, welcher im Auftrag eines NotificationConsumers einen subscribeRequest an einen NotificationProducer durchführt.

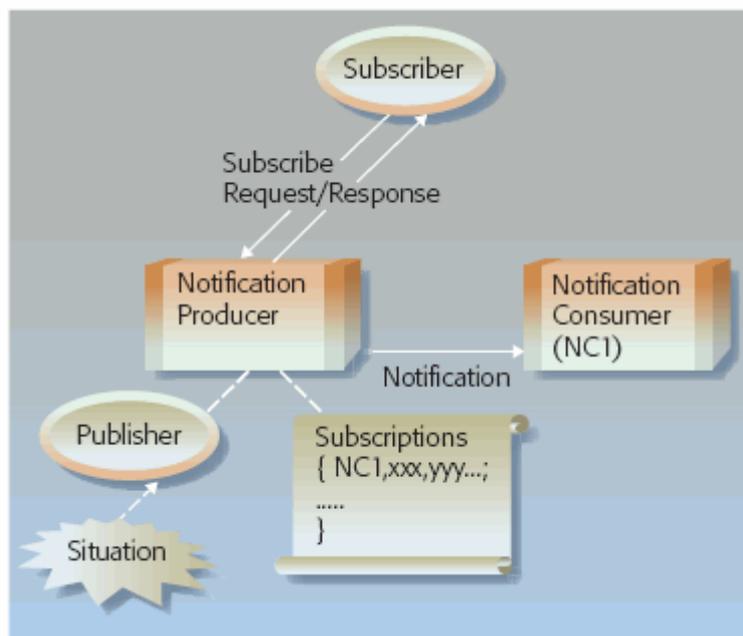


Abbildung 33 subscribeRequest eines NotificationConsumers (Niblett & Graham, 2005, S. 873)

Als Ergebnis des Requests, fügt der NotificationProducer eine Subscription zu seiner Subscriptionsliste hinzu und sendet einen Response an den Subscriber. Jede Subscription Entry protokolliert den NotificationConsumer (siehe Abbildung 33: NC1) – zusammen mit weiteren Properties der Subscription (siehe Abbildung 33: xxx, yyy). In der Praxis handelt es sich dabei beispielsweise um Inhalte wie die Termination Time der Subscription und um Filterausdrücke. Schliesslich nimmt der Publisher eine Situation wahr und der NotificationProducer sendet eine Notification an den NotificationConsumer. (Niblett & Graham, 2005, S. 873)

Üblicherweise agiert ein NotificationProducer auch als sogenannter Publisher. Die WS-BaseNotification unterscheidet nicht zwischen den beiden Rollen. Die Entscheidung ob gewisse Aufgaben wie zB das Senden von NotificationMessages an eine separate Publisher-Instanz hinter dem NotificationProducer-Interface delegiert wird oder nicht, wird in der WS-BaseNotification-Spezifikation trotzdem nicht explizit definiert. (Niblett & Graham, 2005, S. 878)

6.2.1.2 Direct Notification-Pattern

Die sogenannte `direct notification` ist dann gegeben, wenn der `NotificationProducer` auch als `Publisher` agiert. (Niblett & Graham, 2005, S. 878)

Im direkten Fall implementiert das `publishing Webservice` die `Message Exchanges`, welche mit dem `NotificationProducer-Interface` assoziiert sind. Es ist verantwortlich für das Akzeptieren von `SubscribeMessages` und das Senden von `Notifications` an interessierte Teilnehmer. (Chappell & Liu, 2004, S. 12)

Im Gegensatz dazu kommt in der `WS-BrokeredNotification-Spezifikation` die „`Brokered Notification`“ zum Einsatz, bei welchem das Konzept des `Notification Brokers` definiert ist, welcher als ein „`Intermediary Webservice`“ die Instanzen `Publisher` und `NotificationProducers` entkoppelt. (Niblett & Graham, 2005, S. 879)

6.2.1.2.1 Beispiel Printer Verwaltung

Es wurde bereits erwähnt, dass der `NotificationPattern` nicht nur „1-to-1“, sondern auch „1-to-n“ bzw. „n-to-1“ Nachrichtenbeziehungen unterstützt. Zur Veranschaulichung kann folgendes Druckerverwaltungsszenario angeführt werden.

6.2.1.2.1.1 Abbildung: Ein Producer/mehrere Consumer

In einem `Printer Management Szenario` können beispielsweise mehrere `Producer` und ein `Consumer` definiert werden.

Ein `NotificationProducer Webservice` repräsentiert hierbei jeden Drucker in einer Abteilung. Ein einzelnes `Manager-Programm` überwacht die Drucker und agiert als `NotificationConsumer`. Jeder Drucker kann eine Reihe von Zuständen einnehmen (`offline`, `printing`, `out of paper`, `paper jammed`, etc.). Wenn die Zustände sich ändern, dann werden `Notifications` erzeugt. Die überwachende Anwendung registriert sich bei jedem Drucker in der Abteilung (durch die Operation `subscribe`). Der Zustand eines jeden Druckers kann überwacht werden, wenn deren Zustand sich ändert. Somit ist es nicht

notwendig kontinuierlich eine Abfrage bei jeden Drucker durchzuführen. (Niblett & Graham, 2005, S. 873)

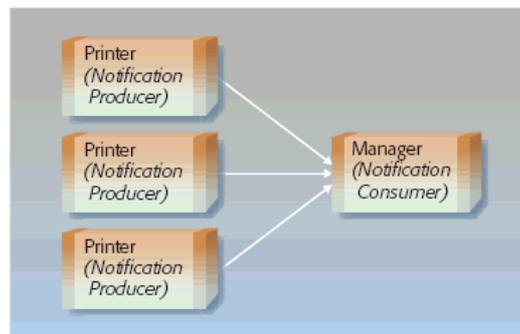


Abbildung 34 Szenario Printer-Management (Niblett & Graham, 2005, S. 873)

6.2.1.2.1.2 Abbildung : Mehrere Producer/ein Consumer

Genauso ist ebenso ein Szenario denkbar, wo nur ein `Producer` jedoch mehrere `Consumer` definiert werden.

Wie nachfolgende Abbildung zeigt, können sich beispielsweise mehrere Händler, welche die Rolle der `NotificationConsumer` einnehmen, benachrichtigen lassen, wenn es zu Änderungen von Aktienkursen kommt.

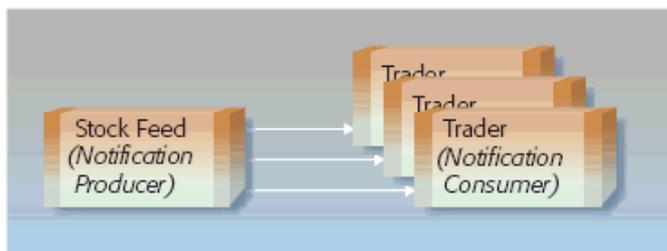


Abbildung 35 Beispiel Aktienkursinformation: Benachrichtigung an mehrere `Notification Consumer`(Niblett & Graham, 2005, S. 873)

6.2.2 Schnittstelle Notification Consumer

Jene Instanzen, welche aufgrund von Interesse an diesen Situationen daran interessiert sind, eine `Notification Message` zu erhalten, bezeichnet die Spezifikation als sogenannte `NotificationConsumer`.

Der Nachrichtenkonsument kann eine Registrierung für ein bestimmtes `Topic` (Thema) beim Nachrichteproduzenten anmelden. (Graham, et al., 2004)

6.2.2.1 Definition

Ein `NotificationConsumer` ist ein Endpoint, dargestellt durch eine WS-Addressing Endpoint Reference, welcher designiert ist Notifications zu erhalten, die durch einen `NotificationProducer` als ein Ergebnis einer `Subscription` produziert wurden. (Graham, Hull, & Murray, 2006)

6.2.2.2 WSDL-Deklaration

Es ist nicht notwendig, dass der `NotificationConsumer` eine WS-Ressource darstellt. Notwendig ist es ein WSDL-File zu erzeugen, welches den Webservice definiert.

Tabelle 51 WSDL-Deklaration am Beispiel Satellitensystem (Sundaram, 2005c)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SatelliteAdministrator"
targetNamespace="http://example.com/satellite"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://example.com/satellite"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

<wSDL:import namespace="http://docs.oasis-open.org/w
srf/2004/06/wsrif-WS-ResourceProperties-1.2-draft-01.wsdl"
location="WS-BaseN.wsdl" />

<types>
<xsd:schema
targetNamespace="http://example.com/satellite"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import namespace=
"http://schemas.xmlsoap.org/ws/2004/03/addressing"
schemaLocation="WS-Addressing.xsd" />
</xsd:schema>
</types>

<binding name="ConsumerSoapBinding"
type="wsntw:NotificationConsumer">
<soap:binding style="document"
transport=
"http://schemas.xmlsoap.org/soap/http"/>
<operation name="Notify">
<input> <soap:body use="literal"/> </input>
<output> <soap:body use="literal"/> </output>
</operation>
</binding>
<service name="ConsumerService">
<port name="ConsumerPort"
binding="tns:ConsumerSoapBinding">
<soap:address location=
```

```
"http://example.com/satelliteMessageConsumer"/>
</port>
</service>
</definitions>
```

Im obigen Listing wird das `ConsumerService` erzeugt, welche das `ConsumerSoapBinding` verwendet. Darin wird der `NotificationConsumer` `portType` implementiert.

6.2.3 Filterverwendung (Niblett & Graham, 2005)

Der übliche Weg ist es, dass alle Nachrichten, die ein `NotificationProducer` erzeugt, an den `NotificationConsumer` gesendet werden. Es gibt jedoch Szenarien, wo ein `NotificationConsumer` nur Nachrichten mit von ihm gewünschten bestimmten Kriterien erhalten möchte. Die `WS-BaseNotification` Spezifikation hat hierfür die Möglichkeit geschaffen, Filterausdrücke zu definieren, um den Nachrichtenerhalt einzuschränken.

Folgende 3 Arten von Filterausdrücken können in der `Subscribe Message` angegeben werden:

- `Topic Filters`,
- `Message Filters` und
- `Producer State Filters`

`Topic Filters` stellen einen geeigneten Weg zur Verfügung die Nachrichten zu kategorisieren bzw. nach einem bestimmten Thema (`Topic`) zu ordnen. Es werden alle Benachrichtigungen ausgeschlossen, welche nicht zu dem angegebenen Thema korrespondieren.

`Message Filters`: Durch Verwendung von booleschen Ausdrücken werden bei diesen Filter nur bestimmte Nachrichten versendet.

`Producer State Filters`: Um diese Art des Filterausdrucks zu verwenden, muss der `Subscriber` über Eigenschaften des `NotificationProducers` informiert sein. Der Filter basiert auf dem Status des `Producers`. Dh die Nachrichtensendung erfolgt, wenn ein bestimmter Status vorliegt (bspw. `DebugMode=ON`).

Der `NotificationConsumer` erhält nur dann die Nachricht vom `NotificationProducer`, wenn alle Bedingungen der Filterausdrücke als wahr ausgewertet werden.

6.2.4 Schnittstelle Notification Producer

Ein `NotificationProducer` kann eine `Notify-Message` erzeugen, die eine oder mehrere `Notifications` enthält.

Der `NotificationProducer` ist etwas komplexer als der `NotificationConsumer`. Mehrere `Resource Properties` sind erforderlich, deshalb muss er den Status einer `WS-Resouce` innehalten. Diese `Resource Properties` definieren die `Topics`, für welche der `Producer Subscriptions` akzeptiert. (Sundaram, 2005c)

6.2.4.1 Resource Properties

Die folgenden Kapitel beschreiben die `ResourceProperties` im `ResourcePropertiesDocument` eines `NotificationProducers`.

6.2.4.1.1 wsnt:TopicExpression

Dieses `ResourceProperty` beinhaltet eine Sammlung von `Topics` die vom `NotificationProducer` unterstützt werden. Die Reihe der `Topics` wird ausgedrückt durch Verwendung von einem oder mehreren `<wsnt:TopicExpression>` `resource property-Elementen`.

6.2.4.1.2 wsnt:FixedTopicSet

Das Element zeigt an, wenn sich die Ansammlung der `Topics` die innerhalb des `<wsnt:TopicExpression>` `resource property` enthalten ist, ändert. Der Wert „True“ wird verwendet, wenn die `Collection` der `Topics`, die vom `NotificationProducer` unterstützt werden, sich nicht ändert. Der Wert „false“ wird verwendet, wenn der `NotificationProducer` es erlaubt, dass die `Collection` geändert wird (beispielsweise wenn zusätzliche `Topics` unterstützt werden, sollen `Publisher` oder `Subscriber` diese abfragen).

6.2.4.1.3 wsnt:TopicExpressionDialect

Zeigt an ein oder mehrere TopicExpression Dialekte, die vom NotificationProducer unterstützt werden.

6.2.4.1.4 wstop:TopicSet

Diese Resource Property beinhaltet die Collection an Topics, die vom NotificationProducer unterstützt werden. Sie werden als einzelnes XML Element ausgedrückt (siehe WS-Topics).

6.2.4.1.5 Beispiel RPD

Ein Resource Properties Document eines NotificationProducers könnte beispielsweise folgendermaßen ausschauen:

Tabelle 52 Beispiel für ein RPD eines NotificationProducers (Sundaram, 2005c)

```
<wsn:NotificationProducerRP
xmlns:wsn="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd">
<wsn:Topic>thresholdReached</wsn:Topic>
<wsn:Topic>counterReset</wsn:Topic>
<wsn:FixedTopicSet>
false
</wsn:FixedTopicSet>
<wsn:TopicExpressionDialects>
http://docs.oasis-open.org/wsn/2004/06/TopicExpression/Simple
</wsn:TopicExpressionDialects>
</wsn:NotificationProducerRP>
```

6.2.4.2 WSDL-Deklaration

Um die Fähigkeiten des NotificationProducers zum bereits vorhandenen Satellitenservice hinzuzufügen, ist es notwendig die WSDL-Datei bezüglich der producerbezogenen ResourceProperties anzupassen.

6.2.4.2.1 Angabe der Namensräume

Einerseits werden die WSN-spezifischen Namensräume deklariert.

Tabelle 53 Angabe der Namensräume

```
xmlns:wsnt="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd"
xmlns:wsntw= "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"

<wsdl:import namespace="http://docs.oasis-open.or
```

```
g/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
location="WS-BaseN.wsdl" />
```

6.2.4.2.2 Angabe des Topic-spezifischen Schemas

Andererseits wird im `<types>` Element das Topic-spezifische ResourceProperties-Schema angegeben:

Tabelle 54 Angabe des Topic-spezifischen Schemas (Graham, Hull, & Murray, 2006, S. 14)

```
<xsd:element ref="wsnt:Topic"
minOccurs="1" maxOccurs="unbounded"/>
<xsd:element
ref="wsnt:FixedTopicSet"
minOccurs="1" maxOccurs="1"/>
<xsd:element
ref="wsnt:TopicExpressionDialects"
minOccurs="1" maxOccurs="unbounded"/>
```

6.2.4.2.3 Angabe der Nachrichtenaustauschmuster

Da ein NotificationProducer als WS-Resource deklariert ist, müssen ebenso die erforderlichen Nachrichtenaustauschmuster bezüglich der Resource Properties-Specification bereitgestellt werden. (bspw. GetResourceProperty).

6.2.4.2.4 Angabe der Operationen

Des Weiteren werden von seiten des NotificationProducers folgende Operationen zur Verfügung gestellt (Sundaram, 2005c).

- Subscribe
- GetCurrentMessage

6.2.4.3 Operation Subscribe (Erstellen einer Subscription)

Um eine Subscription (dt: Abonnement) für den NotificationConsumer durchzuführen, wird eine Subscribe-Nachricht an den NotificationProducer gesendet.

Die Subscribe-Operation wird von NotificationProducer zur Verfügung gestellt, um es anderen Instanzen zu erlauben, Interesse dafür anzumelden, dass erzeugte Notifications empfangen werden können. (Graham, et al., 2004, S. 415)

Eine Subscription repräsentiert die Beziehung zwischen einem NotificationConsumer und einem NotificationProducer, einschließlich irgendwelcher FilteringParameter so wie ein Topic oder verschiedener anderer optionaler Filter-Ausdrücke. (Graham, Hull, & Murray, 2006)

6.2.4.3.1 Resource Properties

Im Body des Subscribe-Requests sind neben der aktuellen Subscribe Message, auch folgende ResourceProperties angeführt (Graham, et al., 2004, S. 420):

- ConsumerReference bezeichnet die EPR des Consumers, welcher jene NotificationMessages erhält, die mit der Subscription assoziiert sind.
- TopicExpression ist der Ausdruck, welcher die Reihe an Topics anzeigt, die mit der Subscription assoziiert sind. Im Listing ist spezifiziert, dass der Consumer eine Subscription für das counterReset Topic wahrnehmen will.
- UseNotify zeigt an, ob die NotificationMessage mit zusätzlichen Metadaten gesendet wird.
- TerminationTime ist definiert durch die Spezifikation WS-ResourceLifetime. Diese ResourceProperties beschreiben die Lebensdauer der Ressource.

6.2.4.3.2 Subscribe-Soap-Request (Header-Teil)

Nachfolgend ist der Request angeführt, welcher bei der Erstellung einer Subscription gesendet wird:

Tabelle 55 Header-Teil - Request bei der Erstellung einer Subscription (Sundaram, 2005c, S. 16)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-BaseNo
tification/Subscribe
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatelliteId>
SAT9928
```

```
</sat:SatelliteId>
</SOAP-ENV:Header>
```

Im Header ist die WS-Adressing Information der WS-Resource angegeben. Dadurch wird der Satellit repräsentiert, welcher als NotificationProducer agiert und gleichsam Messages erhalten kann. (Graham, et al., 2004, S. 420)

6.2.4.3.3 Subscribe-Soap-Request (Body-Teil)

Tabelle 56 Body-Teil - Request bei der Erstellung einer Subscription (Sundaram, 2005c, S. 16)

```
<SOAP-ENV:Body>
<wsnt:Subscribe>
<wsnt:ConsumerReference>
<wsa:EndpointReference>
<wsa:Address>
http://example.com/satelliteMessageConsumer
</wsa:Address>
</wsa:EndpointReference>
</wsnt:ConsumerReference>
<wsnt:TopicExpression Dialect="http://docs.oasis-open.org/wsn/2004/06/TopicExpression/Simple">
sat:counterReset
</wsnt:TopicExpression>
<wsnt:UseNotify>true</wsnt:UseNotify>
<wsnt:InitialTerminationTime>
2005-12-31T12:00:00
</wsnt:InitialTerminationTime>
</wsnt:Subscribe>
</SOAP-ENV:Body>
```

6.2.4.3.4 Subscribe-Soap-Response (Header- und Body-Teil)

Es wird ein Fault retourniert, falls die Subscription nicht behandelt werden kann.

WS-Notification modelliert die Subscription als eine WS-Resource. Um die Assoziierung durchzuführen, wird also eine neue WS-Resource erzeugt: die sogenannte Subscription WS-Resource. Als Antwort auf den Subscribe-Request wird eine WS-Resource-qualified endpoint reference bezüglich der Subscription Resource an den Subscriber zurückgeliefert. Deshalb kann man den Subscribe-Request als sogenannte WS-Resource Factory betrachten. (Graham, et al., 2004, S. 416)

Tabelle 57 Response bei der Erstellung einer Subscription(Sundaram, 2005c, S. 17)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-BaseNotification/SubscribeResponse
</wsa:Action>
```

```

<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satelliteMessageConsumer
</wsa:To>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsnt:SubscribeResponse>
<wsnt:SubscriptionReference>
<wsa:Address>
http://example.com/satelliteSubscriptions
</wsa:Address>
<wsa:ReferenceProperties>
<sat:ResourceID>
SUB3772
</sat:ResourceID>
</wsa:ReferenceProperties>
</wsnt:SubscriptionReference>
</wsnt:SubscribeResponse>
</SOAP-ENV:Body>

```

Der Header enthält die Information für den NotificationConsumer. Der Body enthält die Endpoint Referenz für die aktuelle Subscription.

6.2.4.4 Operation *getCurrentMessage*

Wenn ein Consumer einmal eine Subscription für ein bestimmtes Topic vorgenommen hat, dann möchte er öfter die letzte Message zu diesem Topic einsehen. Wenn der NotificationProducer die aktuelle Nachricht in seinem Cache abspeichert, dann kann der Consumer diese abfragen, indem er die Nachricht `GetCurrentMessage` verwendet. (Sundaram, Understanding WSRF, Part 3, 2005c)

Diese ist für neu-angemeldete NotificationConsumer sinnvoll, damit diese die jüngsten Notifications erhalten können, die die anderen NotificationConsumers bereits erhalten haben. Falls der NotificationProducer die letzten Notifications aufgrund von gewissen Umständen nicht gecacht hat, dann muss er mit einer Fault-Message antworten, die anzeigt, dass die aktuelle Message im Bezug auf das Topic nicht verfügbar ist. (Graham, Hull, & Murray, 2006, S. 22)

6.2.4.4.1 Soap-Request (Header- und Body-Teil)

Tabelle 58 SOAP-Request (Header- und Body-Teil) (Sundaram, 2005c)

```

<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-BaseNot
ification/GetCurrentMessage

```

```

</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satellite
</wsa:To>
<sat:SatelliteId>
SAT9928
</sat:SatelliteId>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<wsnt:GetCurrentMessage>
<wsnt:Topic Dialect=
"http://docs.oasis-open.org/wsn/2004/06/Topic
Expression/Simple">
sat:counterReset
</wsnt:Topic>
</wsnt:GetCurrentMessage>
</SOAP-ENV:Body>

```

6.2.4.4.2 Soap-Response (Header- und Body-Teil)

Tabelle 59 SOAP-Response (Header- und Body-Teil) (Sundaram, 2005c)

```

<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-BaseNot
ification/GetCurrentMessageResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satelliteConsumer
</wsa:To>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsnt:GetCurrentMessageResponse>
<sat:MessageTitle>
Satellite Reset
</sat:MessageTitle>
<sat:MessageContent>
The satellite has been rebooted due to an
administrator request.
</sat:MessageContent>
</wsnt:GetCurrentMessageResponse>
</SOAP-ENV:Body>

```

6.2.5 Subscription

Vor allem in internetbasierten Umgebungen kann folgendes Szenario eintreten. Eine Applikation sendet zunächst einen Subscription Request an den NotificationProducer und akzeptiert auch die erhaltenen Nachrichten einige Zeit lang. Dann kann es jedoch sein, dass die Applikation nicht mehr in Erscheinung tritt und verschwindet. Aufgrund dieses Falles muss von seiten des NotificationProducers entschieden werden, ob die Subscription aufrechterhalten wird oder gelöscht wird. (Niblett & Graham, 2005)

Die `WS-BasedNotification` Spezifikation stellt für `NotificationProducer` die Möglichkeit bereit, dass eine `Subscription` in ihrer Lebenszeit begrenzt werden kann. Um dies zu bewerkstelligen, wird in der `Subscribe Request Message` ein Zeitparameter angegeben, welcher den Zeitpunkt angibt, in dem die `Subscription` abläuft. (Niblett & Graham, 2005)

Der Parameter kann die folgende Formen annehmen (Niblett & Graham, 2005):

- ein absoluter Zeitpunkt (UTC), ab welchem die `Subscription` beendet wird.
- eine relative Zeitangabe, ab wann die `Subscription` beendet werden soll.
- der Wert nil, für den Fall, dass die `Subscription` weitergeführt werden soll

Die Lebensdauer einer `Subscription` wird festgesetzt, wenn sich ein `NotificationConsumer` bei einem `NotificationProducer` im Zuge einer `Subscription Request Message` unter Einbeziehung der Zeitangabe registriert. Es besteht jedoch auch die Möglichkeit die Lebenszeit einer `Subscription` nachträglich abzuändern, indem ein Erneuerungsrequest (`Renew`) gesendet wird. (Niblett & Graham, 2005)

6.2.5.1 Schnittstelle Subscriber bzw. Subscription Manager

Die `Subscription WS-Resource` wird mit einem Webservice assoziiert, bei welchem der `SubscriptionManager` `<Porttype>` implementiert wird. Dieser `<Porttype>` definiert eine Ansammlung an `Resource Properties`, die zu den Komponenten im `Subscribe Request` korrespondieren. Dem Requestor ist es erlaubt die Werte der `Subscription-bezogenen Resource Properties` durch Verwendung von `WS-ResourceProperties-Operationen` auszulesen und zu beschreiben. (Graham, et al., 2004, S. 416)

Der `<Porttype>` definiert ebenso eine `CreationTime Resource Property`, welche aufzeichnet wann die `Subscription` erzeugt wurde. Der `SubscriptionManager` unterstützt des weiteren die von `WS-ResourceLifetime` definierten Operationen, um die Lebensdauer der `Subscription` zu managen. Requestoren haben also die Fähigkeit die

Ressource zu zerstören oder auch einen geplanten Beendigungs-Mechanismus zu verwenden. (Graham, et al., 2004, S. 416)

Man unterscheidet den Base SubscriptionManager und Pausable SubscriptionManager, welche jeweils verschiedene Operationen anbieten.

6.2.5.1.1 Base Subscription Manager

Die grundlegende Handlungsweise des Base SubscriptionManagers ist es, die Zeitdauer einer Subscription Resource zu erneuern und eine Subscription zu beenden.

6.2.5.1.1.1 Operation Renew

Um die derzeitige Lebenszeit einer Subscription zu modifizieren, sendet ein Requestor eine Renew Request Message zum SubscriptionManager.

6.2.5.1.1.2 Operation Unsubscribe

Um eine Subscription zu beenden, sendet ein Requestor eine Unsubscribe Request Message an den SubscriptionManager.

6.2.5.1.2 Pausable Subscription Manager

6.2.5.1.2.1 Operation PauseSubscription

Um die Produktion von Notifications an einer gegebenen Subscription temporär zu unterbrechen, kann ein Requestor eine PauseSubscription Request Message an den SubscriptionManager senden.

Für den Fall dass beispielsweise die Subscription bezüglich des Topic thresholdReached für eine gewissen Zeit unterbrochen werden soll, wird eine PauseSubscription Nachricht an den SubscriptionManager gesendet.

Tabelle 60 SOAP-Request (Header- und Body-Teil)(Sundaram, 2005c, S. 24)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-
BaseNotification/PauseSubscription
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satelliteSubscriptions
</wsa:To>
```

```
<sat:ResourceID>SUB3775</sat:ResourceID>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsnt:PauseSubscription />
</SOAP-ENV:Body>
```

Der SubscriptionManager sendet daraufhin einen einfachen PausSubscriptionResponse.

Tabelle 61 SOAP-Request (Header- und Body-Teil)(Sundaram, 2005c, S. 24)

```
<SOAP-ENV:Header>
<wsa:Action>
http://docs.oasis-open.org/wsn/2004/06/WS-BaseNo
tification/PauseSubscriptionResponse
</wsa:Action>
<wsa:To SOAP-ENV:mustUnderstand="1">
http://example.com/satelliteConsumer
</wsa:To>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
<wsnt:PauseSubscriptionResponse />
</SOAP-ENV:Body>
```

6.2.5.1.2.2 Operation Resume Subscription

Wenn ein Requestor die Produktion von Notifications an einer gegebenen Subscription wieder aufnehmen will, dann muss er eine ResumeSubscription Request Message senden. Das Nachrichtenaustauschmuster ist genauso einfach aufgebaut wie bei der PauseSubscription-Operation.

6.3 WS-Topics

Nachfolgend wird die Spezifikation `WS-Topics` näher vorgestellt, welche ein weiterer Bestandteil der Spezifikationenfamilie `WSN` ist. Auf das Konzept der `Topics` wurde bereits im Zusammenhang mit Filtern (siehe Kapitel 6.1.4.3) verwiesen.

Die `WS-Topics` Spezifikation definiert nun im besonderen die Konstrukte `Topic` und `Topicspace`, welche in Verbindung mit der `WS-BaseNotification` verwendet werden. Das Ziel der `WS-Topics` Spezifikation ist es, Mechanismen zu definieren, um Interessens-Einheiten (`Topics`) für eine `Subscription` zu organisieren und kategorisieren. (Vambenepe, Graham, & Niblett, 2006, S. 4)

6.3.1 Begriff `Topic`

Die `WS-Notification` Spezifikation erlaubt es sogenannte `Topics` zu verwenden, damit eine Reihe von `Notifications` organisiert und kategorisiert werden können. Der `Topics`-Mechanismus stellt eine bequeme Vorgangsweise zur Verfügung, die es `Subscribern` erlaubt, ihre `Notification` welche für sie von Interesse sind, zu veranlassen. Es ist Teil der Veröffentlichung einer `Notification`, dass ein `Publisher` die `Notification` mit einen oder mehreren `Topics` assoziiert. Beim Erzeugen einer `Subscription` durch den `Subscriber`, wird eine `Topic Filter Expression` angegeben, die mit einen oder mehreren `Topics` in Beziehung steht. Der `NotificationProducer` verwendet diese Reihe an `Topics` für den Übereinstimmungsprozess bei der Versendung von `Notifications`. (Vambenepe, Graham, & Niblett, 2006, S. 8)

6.3.1.1 Begriff `TopicTree`

`Topics` sind hierarchisch organisiert und bilden dabei einen sogenannten `TopicTree`. Ein `TopicTree` organisiert eine Familie von zugehörigen `Topics` und `ChildTopics`. Jeder `TopicTree` hat einen `RootTopic`. (Graham, et al., 2004, S. 420)

6.3.1.2 Beispielszenario CRM-System

Anhand eines CRM-Systems (Customer Relationship Management) lässt sich beispielhaft veranschaulichen, wie die Struktur eines `TopicTree` auf eine XML-Struktur abgebildet wird.

Das System ist dafür verantwortlich, verschiedene Kundenkontaktsituationen ausfindig zu machen. Verschiedene andere Applikationen sind daran interessiert `Notifications` zu erhalten, wenn dann gewisse Situationen auftreten. Die `Topics` werden verwendet, um diese Situationen in wiedererkennbare Kategorien zu organisieren. Dies führt dazu, dass der `Subscribe`-Vorgang vereinfacht wird. In Abbildung 36 `TopicTree` wird der `telephoneContact` `Topic` modelliert. Dieser ist der `Root-Topic` für alle anderen `Topics`, die sich auch Telefonkontakte beziehen. Die Unterkategorien des Telefonkontaktes werden als `Child Topics` modelliert. (Graham, et al., 2004, S. 421)

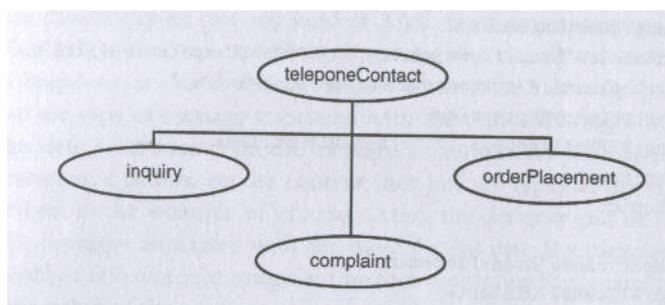


Abbildung 36 `TopicTree telephoneContact` . (Graham, et al., 2004, S. 421)

Die hierarchische Beziehung zwischen den `Topics` wird nun durch folgende XML-Struktur dargestellt.

Tabelle 62 XML Darstellung der `Topics telephoneContact` (Graham, et al., 2004, S. 421)

```

<wstop:Topic name="telephoneContact">
  <wstop:Topic name="inquiry" ... />
  <wstop:Topic name="complaint" ... />
  <wstop:Topic name="orderPlacement" ... />
</wstop:Topic>
  
```

6.3.1.2.1 Attributangaben

Das XML-Model für `Topics` kann mit zusätzlicher Information ausgestattet werden.

6.3.1.2.2 messageTypes

Das `messageTypes` Attribut beschreibt die Nachrichtenart, welche mit dem `Topic` assoziiert wird.

6.3.1.2.3 final

Das `final` Attribut ist als boolescher Ausdruck deklariert und zeigt an, ob es möglich ist, `Topic Childs` zum `Topic` dynamisch hinzuzufügen. (Graham, et al., 2004, S. 424)

6.3.2 Begriff TopicNamespace

Die `WS-Topics` Spezifikation bietet die Möglichkeit an, dass jedes `Topic` in einem XML-Namespace definiert wird.

Jene `Topics`, die mit demselben XML-Namespace in Verbindung stehen, werden als `TopicNamespace` bezeichnet. Ein `TopicNamespace` stellt somit eine Ansammlung von `TopicTrees` dar. Des weiteren kann der `TopicNamespace` zusätzliche Metadaten enthalten die sich auf seine teilnehmenden `Topics` beziehen. Diese Zuordnung erfolgt, um Namenskollisionen zu vermeiden. Dadurch wird die Zusammenarbeit zwischen den Applikationen der `NotificationProducer` und `NotificationConsumer` vereinfacht. (IBM, 2008)

Aufgrund der voneinander unabhängigen Entwicklung der Applikationen könnten außerdem Namenskollisionen prinzipiell von vornherein nicht ausgeschlossen werden. Ein Entwickler, der ein Benachrichtigungssystem implementiert, kann somit eine `Topic`-Struktur festsetzen, die er für diesen Namespace für passend hält. Beispielsweise könnte die gewählte Sprache der Veröffentlichungen definiert werden (bspw. Auswahlmöglichkeit zwischen Englisch und Französisch). Ein `NotificationConsumer` kann somit die Benachrichtigungen in der gewünschten Sprache erhalten. (IBM, 2008)

6.3.2.1 Beispiel TopicNamespace

Der `TopicNamespace` (<http://example.org/topicSpace/example1>) definiert insgesamt 6 `Topics` (2 `RootTopics` und deren 4 `ChildTopics`). Der `TopicNamespace` hat zwei `TopicTrees` mit den Wurzeln `<t1>` und `<t4>`:

- Topic <t1> verfügt über die ChildTopics <t2> und <t3>.
- Topic <t4> verfügt über die ChildTopics <t5> und <t6>.

Die folgende Abbildung veranschaulicht die Darstellung dieses TopicNamespaces.

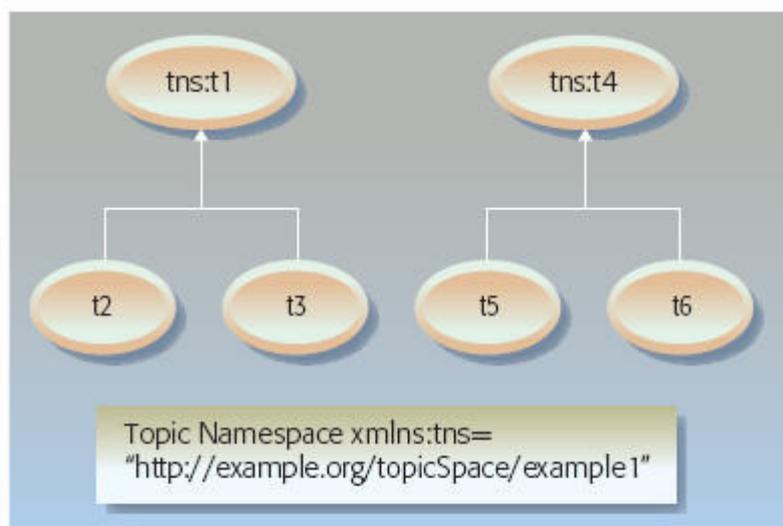


Abbildung 37 Darstellung eines TopicNamespaces (Vambenepe, Graham, & Niblett, 2006, S. 10)

Das folgende XML-Dokument veranschaulicht, wie der TopicNamespace dargestellt wird.

Tabelle 63 XML-Darstellung des TopicNamespaces (Vambenepe, Graham, & Niblett, 2006, S. 10)

```

<?xml version="1.0" encoding="UTF-8"?>
<wstop:TopicNamespace name="TopicSpaceExample1"
targetNamespace="http://example.org/topicSpace/example1"
xmlns:tns="http://example.org/topicSpace/example1"
xmlns:xyz="http://example.org/anotherNamespace"
xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/wsn/t-1
http://docs.oasis-open.org/wsn/t-1.xsd" >

<wstop:Topic name="t1">
<wstop:Topic name="t2" messageTypes="xyz:m1 tns:m2"/>
<wstop:Topic name="t3" messageTypes="xyz:m3"/>
</wstop:Topic>

<wstop:Topic name="t4">
<wstop:Topic name="t5" messageTypes="tns:m3"/>
<wstop:Topic name="t6"/>
</wstop:Topic>
</wstop:TopicNamespace>
  
```

6.3.3 Begriff TopicSet

Der Begriff `TopicSet` wird im Zusammenhang mit dem `NotificationProducer` verwendet und bezeichnet die Ansammlung der `Topics`, die von diesem unterstützt werden. Es ist hierbei insbesondere der Unterschied zum Begriff `TopicNamespace` anzumerken. Ein `TopicNamespace` ist eine abstrakte Reihe an `Topic` Definitionen. Das `TopicSet` eines `NotificationProducers` kann darüber hinaus `Topics` von mehreren verschiedenen `TopicNamespaces` enthalten. (Vambenepe, Graham, & Niblett, 2006, S. 8)

6.3.3.1 Beispiel TopicSet

Als Beispiel soll nochmals der zuvor erläuterte `TopicNamespace` herangezogen werden, welcher insgesamt sechs `Topics` beinhaltet.

Es wird nun angenommen, dass der `NotificationProducer` folgende 3 `Topics` nutzen möchte:

- `RootTopic` `<tns:t1>`
- `<t2>` `ChildTopic` von `<tns:t1>`
- `<t5>` `ChildTopic` von `<tns:t4>`

Der `NotificationProducer` unterstützt diese `Topics` durch Hinzufügen zu seinem `TopicSet`.

Die Darstellung des `TopicSet` Dokuments sieht folgendermaßen aus:

Tabelle 64 XML-Darstellung TopicSet(Vambenepe, Graham, & Niblett, 2006, S. 11)

```
<?xml version="1.0" encoding="UTF-8"?>
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:tns="http://example.org/topics/example1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/wsn/t-1
http://docs.oasis-open.org/wsn/t-1.xsd">

<tns:t1
wstop:topic="true">
<t2
wstop:topic="true"/>
</tns:t1>

<tns:t4>
<t5
wstop:topic="true"/>
</tns:t4>
```

```
</wstop:TopicSet>
```

6.3.4 Begriff TopicExpression

Topics werden durch TopicExpressions referenziert. Diese Expressions können an folgenden Stellen aufscheinen (Vambenepe, Graham, & Niblett, 2006, S. 18):

- Als Komponente der Subscribe Message Request an den NotificationProducer.
- Als Komponente einer Notification Message, die an einen NotificationConsumer oder NotificationBroker gesendet wird.
- Im TopicExpression Resource PropertyElement, die mit der NotificationProducerRole assoziiert sind.

Die Expressions enthalten eine große Verschiedenartigkeit an ExpressionTypes. Der Typ der Expression wird durch das Dialekt Attribut in einer Topic Expression bestimmt. (Graham, et al., 2004, S. 426)

Die Dialekt Komponente enthält eine URI, welche den Grammatik-Typ identifiziert, der in der TopicExpression verwendet wird. Der Zweck der TopicExpression ist es eine Reihe von einem oder mehreren Topics zu identifizieren. Diese Topics können einem oder mehreren Namespaces zugeordnet sein. (Vambenepe, Graham, & Niblett, 2006, S. 18)

In den nachstehenden Kapitel sind die zur Auswahl stehenden Dialekte beschrieben.

6.3.4.1 Simple TopicExpression

Dieser Dialekt wird verwendet werden, um nur auf RootTopics zu referenzieren. Ein NotificationProducer welcher nur RootTopics akzeptiert, kann nur Subscriptions wählen, die Simple TopicExpressions beinhalten. (Niblett & Graham, 2005, S. 876)

Die nachfolgende TopicExpression definiert den RootTopic <t1> innerhalb des TopicNamespaces (tns). Die Namespace-Deklaration ist:

```
xmlns:tns="http://example.org/topics/example1"
```

Tabelle 65 XML-Beispiel für Simple TopicExpression (Vambenepe, Graham, & Niblett, 2006, S. 18)

```
<wsnt:TopicExpression
Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple">
tns:t1
</wsnt:TopicExpression>
```

6.3.4.2 Concrete TopicExpression

Dieser Dialekt wird verwendet, um ein einzelnes Topic innerhalb des TopicNamespaces zu identifizieren. (Vambenepe, Graham, & Niblett, 2006, S. 18)

Die folgende TopicExpression identifiziert den Topic namens <t3>, das ChildTopic von <tns:t1>. Das Forward Slash-Schriftzeichen dient hierbei als Separator (Vambenepe, Graham, & Niblett, 2006, S. 20).

Tabelle 66 XML-Struktur für Concrete TopicExpression (Vambenepe, Graham, & Niblett, 2006, S. 20)

```
<wsnt:TopicExpression
Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete">
tns:t1/t3
</wsnt:TopicExpression>
```

6.3.4.3 Full TopicExpression

Dieser Ausdruck ist eine Erweiterung der Concrete TopicExpression, um mehrere Topics herauspicken zu können. (Niblett & Graham, 2005, S. 876)

Der Dialekt wird durch folgende URI identifiziert:

```
Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Full"
```

Nachfolgend sind beispielhafte Ausdrücke angeführt. Der Wildcard Character * wird verwendet um eine Knotenreihe zu identifizieren, welches aus einer Ansammlung von ChildTopics besteht (Vambenepe, Graham, & Niblett, 2006, S. 22):

- „tns:t1/*“
Identifiziert alle ChildTopics von dem RootTopic <t1>
- „tns:t1/*/t3“
Identifiziert alle GrandChilds von <tns:t1>, welche den Namen <t3> haben.

- „tns:*“
Identifiziert alle `RootTopics` in dem `<tns:TopicNamespace>`.
- „tns:t1/t3//.“
Hier wird der Sub-Tree identifiziert, welcher aus `<tns:t1/t3>` und all seiner Nachfolger besteht. Wenn also die `TopicExpression` mit den Charakter `</>` endet, dann wird angezeigt, dass die `TopicExpression` einem `TopicSubtree` entspricht.

6.4 WS-Brokered Notification

Generell hat ein `NotificationProducer` hauptsächlich folgende Aufgaben zu bewältigen:

- das Ermitteln von Situationen und
- das Erzeugen von `NotificationMessages`.

Zusätzlich dazu ist er auch dafür verantwortlich die `Notifications` mit der `SubscriptionListe` abzugleichen und die `NotificationMessages` an jeden `SubscribedConsumer` zu senden. Diese Zusatzaufgabe kann jedoch an eine andere Instanz delegiert werden. Die Spezifikation `WS-Brokered Notification` definiert hierfür die Schnittstellen für einen sogenannten `NotificationBroker`. (Niblett & Graham, 2005)

Grundsätzlich unterscheidet die `WS-Notification-Spezifikation` folgende zwei Fälle, wie die Verteilung der `Notifications` erfolgt:

- `direct` und
- `brokered`

Bei der `BaseNotification-Spezifikation` wird der `DirectNotificationPattern` implementiert. Die Spezifikation `WS-Brokered Notification` wendet den `BrokeredNotificationPattern` an:

6.4.1 BrokeredNotificationPattern

Die folgende Abbildung veranschaulicht das Konzept der `BrokeredNotification`:

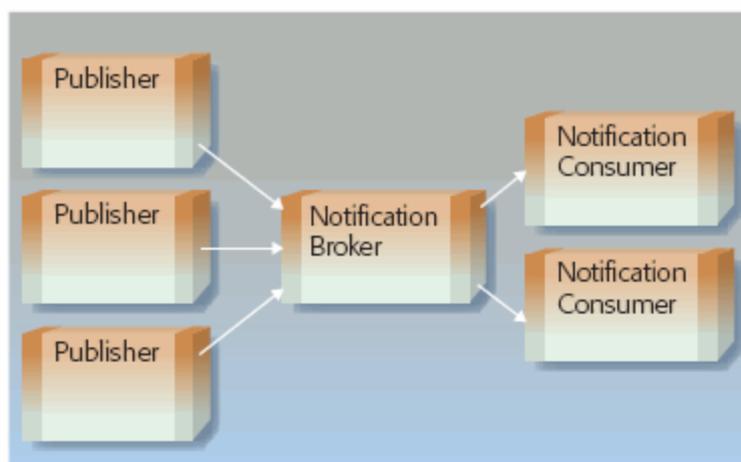


Abbildung 38 Konzept der BrokeredNotification (Niblett & Graham, 2005, S. 879)

Bei der sogenannten `BrokeredNotification` ist ein Vermittler – nämlich ein `NotificationBroker` – für das Verteilen der Nachrichten verantwortlich, welche von einem oder mehreren `Publishern` an die `NotificationConsumer` verteilt werden. (Chappell & Liu, 2004, S. 12)

Bei der `BrokeredNotification` kommt es also im Vergleich zur `DirectNotification` zur Trennung der Instanzen (`NotificationProducer` und `Publisher`).

Der `NotificationBroker` führt die Aufgaben eines `NotificationProducers` im Auftrag eines `Publishers` durch. Der `NotificationBroker` ist selber ein `Webservice`, welches sowohl von `Publishern` als auch von `NotificationConsumern` bedient werden kann. (Niblett & Graham, 2005, S. 879)

Die `WS-Brokered Notification` Spezifikation definiert die Rolle eines `NotificationBrokers` als vermittelndes `Webservice`, welches sogenannte `Publisher` und `NotificationProducers` entkoppelt. Die Spezifikation `WS-BrokeredNotification` baut auf den grundlegenden `Benachrichtigungsmechanismen` von `WS-BaseNotification` auf, indem das Konzept eines vermittelnden `NotificationBrokers` hinzugefügt wird. Des weiteren werden auch zusätzliche Varianten in Bezug auf die Rolle des `Publishers` beschrieben. (Niblett & Graham, 2005, S. 879)

Grundsätzlich kann ein NotificationBroker sowohl die Rolle des NotificationProducers als auch des NotificationConsumers übernehmen. (Chappell & Liu, 2004, S. 8)

Das bedeutet, dass ein Subscriber eine Subscription bei einem NotificationBroker erzeugt, indem er den exakt gleichen Subscribe Request verwendet, als wenn er diesen bei einem Standard NotificationProducer durchführen würde. (Niblett & Graham, 2005, S. 879)

6.4.2 Deklarationen und Funktionalitäten von WS-Brokered

Die WS-BrokeredNotification definiert den NotificationBroker als eine Erweiterung eines NotificationProducers. Somit ist ein NotificationBroker auch alles das, was auch ein NotificationProducer ist. Dazu gehört dass er also als ein Webservice agiert, welches Subscribe Requests akzeptiert oder eine Liste von Topics mittels des wsnt:Topics resource properties Element bereitstellt usw. (Graham, et al., 2004, S. 433)

6.4.2.1 ResourceProperty Element RequiresRegistration

Es muss das ResourceProperty Element RequiresRegistration deklariert werden.

```
<xsd:element name="RequiresRegistration" type="xsd:boolean"/>
```

Ist diese Resource Property mit <true> belegt, dann müssen sich Publisher vorregistrieren, bevor sie einen Publish-Vorgang unternehmen können. (Chappell & Liu, 2004, S. 15)

6.4.2.2 Standard Operationen

Das NotificationBroker Interface muss u.a. Operationen wie Notify, Subscribe und GetCurrentMessage unterstützen.

6.4.2.3 RegisterPublisher Operation

Zusätzlich unterstützt der NotificationBroker auch die RegisterPublisher Operation. Diese Operation erlaubt es Publishern

sich schon früher für `Notifications` mit bestimmten `Topics` zu registrieren. (Graham, et al., 2004, S. 434)

Es wird jedoch nicht von allen `Brokern` eine Vorregistrierung verlangt. Die Notwendigkeit der Vorregistrierung hängt vom Zustand des `RequiresRegistration` `resource` `Property` des `NotificationBrokers` ab. (Graham, et al., 2004, S. 434)

Auf Anfrage eines `RegisterPublisher Request` durch den `Publisher` , erzeugt der `NotificationBroker` eine `PublisherRegistration Resource`. Es wird die `PublisherRegistrationReference` (Ressourcen-Identifizier der `PublisherRegistration Resource`) und eine `ConsumerReference` an den `Publisher` zurückgesendet. (Chappell & Liu, 2004, S. 20)

6.4.2.4 *PublisherRegistration Interface*

Die `WS-BrokeredNotification` stellt des weiteren das sogenannte `PublisherRegistration Interface` zur Verfügung.

Mit dieser Schnittstelle ist es möglich die `PublisherRegistration WS-Resources` zu manipulieren. Mit Hilfe der `Operation DestroyRegistration` können die `WS-Resources` unverzüglich zerstört werden. (Chappell & Liu, 2004, S. 24)

6.4.3 Vorteile der Brokered Notification (im Vergleich zum direct notification pattern)

6.4.3.1 *Delegierung von bestimmten Aufgaben an den Broker*

Die `BrokeredNotification` hat für den `Publisher` den Vorteil, dass er alle `Message Exchanges`, welche mit dem `NotificationProducer` assoziiert sind, implementieren muss. Der `NotificationBroker` übernimmt dabei Aufgaben wie die Verwaltung von `Subscriptions` und die Verteilung von Nachrichten. Er agiert somit in der Rolle eines `SubscriptionManager` bzw eines `NotificationProducers` . (Niblett & Graham, 2005, S. 879)

6.4.3.2 Reduzierung der Beziehungen zwischen den Services

Als Beispiel kann ein Printer Monitoring Szenario herangezogen werden, in welchem es 100 Drucker und 2 Überwachungsapplikationen gibt.

Im Falle einer direkten Notification müssten beide Monitor-Applikationen 100 Subscriptions erteilen. Bei der Statusänderung eines jeden Druckers würden die Messages immer an beide Consumer geschickt werden.

Im Falle der BrokeredNotification würde jede Monitoring Application einmal eine Subscription an den Broker durchführen. Die Drucker würden bei Statusänderungen nur eine Nachricht an den Broker senden.(Niblett & Graham, 2005, S. 879)

6.4.3.3 Funktion als Suchservice

Publisher und Subscriber können über den Broker zueinander finden.(Niblett & Graham, 2005, S. 879)

6.4.3.4 Möglichkeit der anonymen Benachrichtigung

Es ist nicht notwendig, dass ein Publisher und ein Consumer ihre Anonymität aufgeben. Es gibt Szenarien wo es für einen NotificationConsumer nicht wichtig ist zu wissen, wer die Notification produziert hat.(Niblett & Graham, 2005, S. 879)

6.4.3.5 Bereitstellung von value-added functions

Ein NotificationBroker kann zusätzliche Funktionen durchführen. Dazu gehört beispielsweise das Aufzeichnen von Notifikationsnachrichten, um spätere Analysen durchzuführen.(Niblett & Graham, 2005, S. 879)

6.4.4 Publisher-Interaktion mit dem Notification Broker

Es gibt verschiedene Arten, auf welche Weise ein Publisher mit einem NotificationBroker interagiert:

6.4.4.1 Simple Publishing

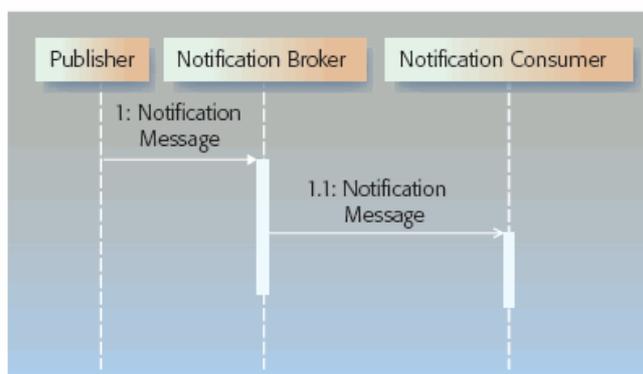


Abbildung 39 Nachrichtenablauf bei Simple Publishing (Niblett & Graham, 2005, S. 880)

Ein Publisher, der eine Situation ausfindig macht, erzeugt eine Nachricht und versendet sie an den NotificationBroker. Der Broker sendet die Nachrichten an die Consumer, da er ja die Subscriptions erfüllen muss. Nachrichten kann er von einem oder mehreren Publishern empfangen. Zusätzlich kann der Broker eine Vorregistrierung veranlassen. Dh. Publisher müssen vor der ersten Nachrichtengenerierung eine Registrierung durchführen. Somit kann der Broker eine Filterung der Publisher durchführen und entscheiden, ob Nachrichten an den Publisher verteilt werden sollen oder nicht. (Niblett & Graham, 2005, S. 880)

Aufgrund der Tatsache, dass das Notifyformat von WS-BaseNotification verwendet wird, wird die Publisher-Nachricht mit einem Topic versehen. Die Nachrichten werden vom Broker mit den jeweiligen Subscriptions verglichen und überprüft, ob eine Übereinstimmung zwischen registrierten Topics und Filtern vorhanden ist. Jene Consumer, deren Subscriptions passen, erhalten dann vom Broker eine Kopie der Nachricht. (Niblett & Graham, 2005, S. 880)

6.4.4.2 Demand-basiertes Publishing

Eine Möglichkeit um unnötigen Nachrichtenverkehr zu vermeiden, bietet der demand-basierter Publishing Ansatz.

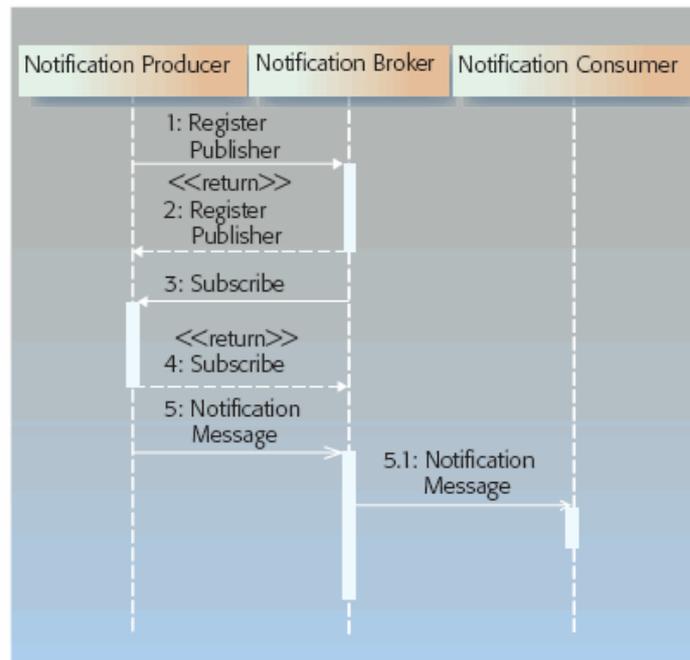


Abbildung 40 Nachrichtenablauf bei Demand basierter Publisher (Niblett & Graham, 2005, S. 881)

Der Nachteil des Simple Publishing ist es, dass im Falle von auftretenden Situationen auch Nachrichten selbst bei nicht relevanten Subscriptions erzeugt werden und diese an den Broker gesendet werden. (Niblett & Graham, 2005, S. 880)

Beim Demand-basierten Publishing kombiniert der Publisher die Rolle des NotificationProducers und Publishers, hat aber nur eine Subscription zu verwalten. Der Publisher implementiert den subscribe Message Exchange vom Notification Producer-Interface und der NotificationBroker verwendet diesen Exchange, um eine Subscription mit dem Publisher einzurichten. Der Publisher sendet dann Benachrichtigungen an den Broker, indem er die standardisierte Notify Message verwendet. Wenn der Broker herausfindet, dass keine relevanten Subscriptions existieren, kann er die Subscription mit dem Publisher unterbrechen und danach wiederaufnehmen, wenn relevante Subscriptions vorhanden sind. (Niblett & Graham, 2005, S. 880)

7 Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde der Aufbau und die Funktionsweise von wesentlichen Standards im Bereich der Webservicearchitekturen betrachtet. Nur aufgrund von intensiven Standardisierungsbemühungen der letzten Jahre konnten Webservicestandards wie WSRF oder WSN einen hohen Standardisierungsgrad erreichen.

Die Interoperabilität ist ein wichtiges Kriterium bei der Realisierung verteilter Systeme, welche im allgemeinen durch die Entwicklung von Standards erreicht werden kann. Die Internettechnologie am Beispiel von Standardprotokollen wie HTTP hat schon um einiges früher bereits vorgezeigt, dass man letztlich nur durch einen hohen Grad an Standardisierung eine allgemeine Verbreitung von Technologien erreichen kann.

Zu einer hohen Akzeptanz der Webservices haben natürlich allen voran die etablierten Kernstandards SOAP und WSDL beigetragen. Umso wichtiger war es auch, dass nun auch Konventionen für die Bereiche der Zustandsmodellierung und des Event-Handlings geschaffen wurden.

Mit der Entwicklung der im Rahmen dieser Diplomarbeit vorgestellten Standards WSRF und WSN wurden wichtige Anforderungen erfüllt, welche bei vielen Computersystemen eine nicht zu vernachlässigbare Notwendigkeit darstellen.

Das WSRF stellt Konventionen bereit, um klare Richtlinien für den Zugang zu zustandsbehafteten Ressourcen zu schaffen. Aus den sonst zustandslosen Webservices können zustandsbehaftete Webservices erzeugt werden und es können Zwischenergebnisse anderen Clients zugänglich gemacht werden. Das WSN stellt einen Mechanismus zur Verfügung, welcher für das Auslösen von Ereignissen in IT-Infrastrukturen verwendet werden kann. Potenzielle Nutzer können damit standardisierte Informationen über die Komponenten, Verfügbarkeit und freie Kapazitäten abrufen.

Möglicherweise wird auch der WSDM-Standard in Zukunft für IT-Systeme eine größere Rolle einnehmen. WSDM stellt auf Basis von Webservices eine komplette Architektur zur Verwaltung von verteilten Ressourcen bereit. Das Ziel

dabei ist es, dass bestehende Applikationen einheitlicher und kosteneffektiver integriert und verwaltet werden können.

Mittels Webservicetechnologien lässt sich somit die Virtualisierung von Ressourcen realisieren, da sie allgemeine, öffentlich zugängliche, interoperable Schnittstellen zur Verfügung stellen.

Der Zugang und die Nutzung von geografisch verteilten heterogenen Ressourcen ist generell eine der zentralen Herausforderungen und wird wohl auch die Zukunft der IT-Systeme maßgeblich beeinflussen. Dieser Ansatz spielt ja bekanntlich in Gridsystemen eine große Rolle.

Die neuen Webservice Standards haben dazu beigetragen, dass hierbei eine größere Flexibilität und Effizienz des Zusammenspiels mit Grid-Komponenten gewährleistet ist. Durch die Entwicklung von WSRF und WSN konnte die Webservice- und Grid community noch näher zusammenwachsen.

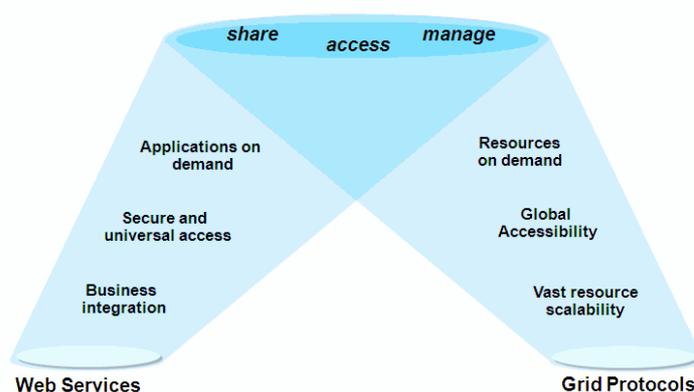


Abbildung 41 Das beste aus den beiden Welten WebServices und Grid(Sabbah, 2004, S. 2)

Die Verbreitung von Webservices hat diesbezüglich auch dazu geführt, dass die ursprünglich rein wissenschaftlich ausgerichteten Gridkonzepte auch immer mehr für die Unternehmenswelt interessant geworden sind. Im Allgemeinen gibt das allgemein bekannte Konzept der service-orientierten Architektur den Weg vor, in welche Richtung sich in Zukunft die IT-Systeme entwickeln werden. Zukünftige Systemlandschaften sollen eine hohe Flexibilität, Wandlungsfähigkeit, Wirtschaftlichkeit und Adaptierbarkeit hinsichtlich dynamischer wechselnder Umgebungen aufweisen. Webservices bieten aufgrund ihrer flexiblen Schnittstellen eine gute Voraussetzung, um den

Aufwand für die Administration und Instandhaltung von IT-Systemen zu vermindern oder um die Wiederverwendbarkeit von Komponenten zu erhöhen. Auch die Auslagerung von Geschäftsprozessen oder der Zukauf von Ressourcen kann mit Webservices vereinfacht werden. Es ist somit notwendig dass Software-Lösungen in Verwendung sind, mit welchen für Geschäftspartner trotzdem ein sicherer Zugriff auf interne Anwendungen und Ressourcen möglich ist.

Es gibt generell eine Menge an Zukunftsinitiativen, die sich im Zusammenhang mit dem service-orientierten Webserviceansatz nennen lassen. An dieser Stelle können u.a folgende Begriffe angeführt werden:

On Demand Computing, Autonomic Computing, Cloud Computing, Software as a Service (SaaS), Service Science on Demand

Das Webservicekonzept ist jedenfalls eine wichtige Weiterentwicklung in der Evolution der Webtechnologien. Im Vergleich zum herkömmlichen Webansatz erfolgt ja die Kommunikation bei Webservices völlig automatisiert - dh. ohne menschliche Interaktion. Insofern ist für Webservices beträchtliches Potenzial für künftige IT-Lösungen vorhanden, da die Automatisierung von IT-Prozessen in der Zukunft eine immer größere Rolle einnehmen wird.

Um eine vollständige Automation zu erreichen, ist es jedoch notwendig, dass auch die semantische ("bedeutende") Komponente hinzukommt. Webservice-Spezifikationen haben bislang hauptsächlich einen syntaktischen Rahmen definiert.

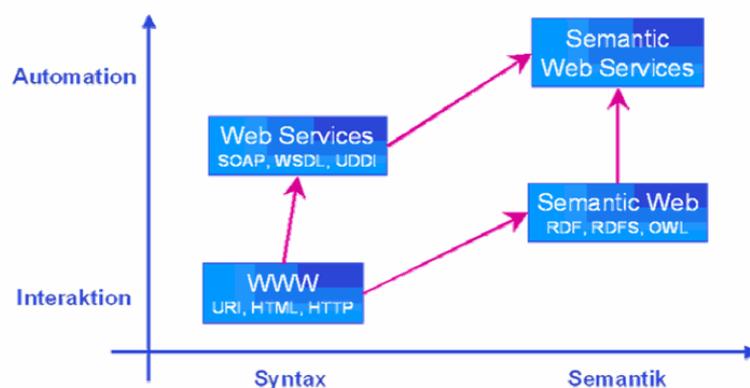


Abbildung 42 Einordnung bezüglich Automation und Semantik (Dostal & Jeckle, 2004)

Eine Service-orientierten Architektur kann letztendlich nur dann verwirklicht werden, wenn Syntax (Web Services) und Semantik (Semantic Web) eine Symbiose bilden. (Dostal & Jeckle, 2004)

Webservices werden mit semantischen Inhalten angereichert und mit Techniken aus dem Bereich des Semantic Web erweitert. Erst wenn eine Ablage, ein Zugriff und eine Auswertung der Semantik in einer maschinenlesbaren Form durch die beteiligten Komponenten erreicht wird, kann eine vollständige Automatisierung mittels Webservices erreicht werden.

(Dostal & Jeckle, 2004)

Diese sogenannten Semantic Web Services (SWS) haben das Potential die IT-Systeme völlig neuartig zu revolutionieren. Die derzeitigen aktuellen Standards bieten jedenfalls eine gute Basis, auf welcher man aufbauen kann. Vieles hängt natürlich davon ab inwieweit die weiteren Standardisierungsprozesse vorangehen. Die heutige IT-Welt sieht sich einer Vielzahl von interessanten Herausforderungen gegenübergestellt. Tatsache ist jedoch, dass die automatisierte Vernetzung von verteilten Komponenten auf Basis von Web Services bzw. Service-orientierten Architekturen weiterhin ein spannendes Thema bleiben wird.

LITERATURVERZEICHNIS

Alonso, G., Casati, F., Harumi, K., & Machiraju, V. (2004). *Web Services - Concepts, Architectures and Applications*. Heidelberg: Springer Verlag.

Antonopoulos, A. M. (11. 05 2004). *Web Services Resource Framework brings together grids, Web services*. (Network World) Abgerufen am 28. 08 2008 von <http://www.networkworld.com/newsletters/datacenter/2004/0510datacenter1.html>

Banks, T. (23. 05 2006). *Web Services Resource Framework (WSRF) - Primer v1.2*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von <http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf>

Benkner, S. (2006). *Advanced Software Engineering (VO-Skriptum)*. Institut für Scientific Computing, Wien.

Bianco, P., Kotermanski, R., & Merson, P. (09 2007). *Evaluating a Service-Oriented Architecture*. (C. M. University, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.sei.cmu.edu/publications/documents/07.reports/07tr015.html>

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., et al. (11. 02 2004). *Web Services Architecture*. (W3C) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., et al. (11. 02 2004). *Web Services Architecture*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

Chappel, D., & Jewell, T. (2003). *Java Web Services*. O'Reilly.

Chappell, D., & Liu, L. (03. 05 2004). *Web Services Brokered Notification*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf

Chase, N. (2007). *Understanding Web Services Specifications, Part1: SOAP*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <https://www6.software.ibm.com/developerworks/education/ws-understand-web-services1/ws-understand-web-services1-a4.pdf>

- Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S. (27. 06 2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. (W3C, Herausgeber) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2007/REC-wsdl20-20070626>
- Dhesiaseelan, A. (20. 05 2004). *What's New in WSDL 2.0*. Abgerufen am 28. 08 2008 von <http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html>
- Dostal, W., & Jeckle, M. (1 2004). *Semantik, Odem einer Service-orientierten Architektur*. Abgerufen am 28. 08 2008 von http://www.sigs.de/publications/js/2004/01/dostal_JS_01_04.pdf
- Dustdar, S., Gall, H., & Hauswirth, M. (2003). *Software-Architekturen für Verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software*. Springer Verlag.
- Foster, I. (20. 07 2002). *What is the Grid? A Three Point Checklist*. Abgerufen am 28. 08 2008 von <http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf>
- Foster, I. (22. 01 2004a). *WS-Resource Framework: Globus Alliance Perspectives*. (GLOBUS, Hrsg.) Abgerufen am 28. 08 2008 von <http://xml.coverpages.org/FosterWSRF200401.pdf>
- Foster, I., Czajkowski, K., Ferguson, D. F., Frey, J., Graham, S., Maguire, T., et al. (01. 03 2004). *Modeling and Managing State in Distributed Systems: The Role of OGSF and WSRF*. Abgerufen am 28. 08 2008 von <http://www.unigrids.org/papers/wsrp.pdf>
- Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., et al. (03. 05 2004b). *Modeling Stateful Resources with Web Services*. (IBM, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.ibm.com/developerworks/webservices/library/specification/ws-resource/ws-modelingresources.html>
- Foster, I., Kesselman, C., & Tuecke, S. (2001). *The Anatomy of the Grid*. (GLOBUS, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.globus.org/alliance/publications/papers/anatomy.pdf>

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002). *The Physiology of the Grid*. (GLOBUS, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.globus.org/alliance/publications/papers/ogsa.pdf>

Frey, J., Graham, S., Czajkowski, K., Ferguson, D. F., Foster, I., Leymann, F., et al. (03. 05 2004). *Web Services Resource Lifetime (WS-ResourceLifetime)*. (IBM, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf>

Graham, S., Davis, D., Simeonov, S., Daniels, G., Brittenham, P., Nakamura, Y., et al. (2004). *Building Webservices with Java* (Second Edition Ausg.).

Graham, S., Hull, D., & Murray, B. (01. 10 2006). *Web Services Base Notification 1.3 (WS-BaseNotification)*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

Graham, S., Karmarkar, A., Mischkin, J., Robinson, I., & Sedukhin, I. (01. 04 2006). *Web Services Resource 1.2 (WS-Resource)*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

Gudgin, M., Hadley, M., & Rogers, T. (09. 05 2006). *Web Services Addressing 1.0 - Core*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., et al. (27. 04 2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., et al. (27. 04 2007a). *SOAP Version 1.2 Part2: Adjuncts (Second Edition)*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2007/REC-soap12-part2-20070427>

Hauser, T. (2004). *Web Services - Die Standards* (1. Auflage Ausg.). Bonn: Galileo Press.

Hauser, T., & Löwer, U. M. (2004). *Web Services - Die Standards* (1. Auflage Ausg.). Bonn: Galileo Press.

- Heinz, B. (2005). *Leitfaden Web Services*. (BITKOM, Hrsg.) Abgerufen am 28. 08 2008 von http://www.bitkom.org/files/documents/Leitfaden_Web_Services__V_2.0_4.10.2005.pdf
- IBM (Hrsg.). (2008). *WS-Topics*. Abgerufen am 28. 08 2008 von http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/concepts/cjwsn_topics.html
- Ilg, R., & Fischer, T. (2003). *Innovationsbeschleuniger in globalen Netzen*. (SAP) Abgerufen am 21. 07 2007 von <http://www.sap.info/de/go/21898/>
- Joseph, J., Ernest, M., & Fellenstein, C. (2004). *Evolution of grid computing architecture and grid adoption models*. (J. IBM SYSTEMS, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.research.ibm.com/journal/sj/434/joseph.pdf>
- Kesselmann, C., & Foster, I. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann.
- Kodali, R. R. (13. 06 2005). *What is service-oriented architecture?* (JavaWorld.com) Abgerufen am 28. 08 2008 von An introduction to SOA: <http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html>
- Kossmann, D., & Leymann, F. (2004). Web Services. *Informatik Spektrum* (27/2), S. 117-128.
- Kuschke, M., & Wölfel, L. (2002). *Web Services kompakt*. Heidelberg: Spektrum Akademischer Verlag.
- Linker, B. (2005, 01 31). *Introduction to WS-Addressing*. (Oracle Technology Network) Retrieved 06 30, 2007, from http://dev2dev.bea.com/pub/a/2005/01/ws_addressing_intro.html
- Liu, L., & Meder, S. (01. 04 2006). *Web Services Base Faults 1.2*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf
- MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P., & Metz, R. (07. 02 2006). *Reference Model für Service Oriented Architecture*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf>

- Maguire, T., Snelling, D., & Banks, T. (01. 04 2006). *Web Services Service Group 1.2 (WS-ServiceGroup)*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf
- Manhart, K. (14. 12 2006). *Merkmale von Web Services | Web Services - Grundlagen, Aufbau und Struktur*. (Tecchannel) Abgerufen am 28. 08 2008 von <http://www.tecchannel.de/webtechnik/soa/457051/index4.html>
- Niblett, P., & Graham, S. (2005). *Events and service-oriented architecture: The OASIS Web Services Notification specifications*. (J. IBM Systems, Hrsg.) Abgerufen am 28. 08 2008 von www.research.ibm.com/journal/sj/444/niblett.pdf
- Nilo, M., & Lafon, Y. (27. 04 2007). *SOAP Version 1.2 Part 0: Primer (Second Edition)*. (W3C, Hrsg.) Abgerufen am 28. 08 2008 von <http://www.w3.org/TR/2007/REC-soap12-part0-20070427>
- OASIS, T. C. (2008). *Web Services Resource Framework (WSRF) TC*. (OASIS, Herausgeber) Abgerufen am 28. 08 2008 von http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
- Öztürk, M., & Stiller, M. (2004). *Grenzenlose Kommunikation mit Web-Services*. Abgerufen am 28. 08 2008 von http://www.sigs.de/publications/js/2004/02/oeztuerk_JS_02_04.pdf
- Papazoglou, M. P. (2007). *Web Services: Principles and Technology*. Prentice Hall.
- Parys, D., & Mauerer, J. (14. 11 2004). *Web Services Standards: SOAP, UDDI und WSDL*. (Microsoft Developer Network) Abgerufen am 28. 08 2008 von <http://msdn.microsoft.com/de-de/library/bb979447.aspx>
- Reimers, S. (08. 07 2005). *Technische Grundlagen von Web Services*. Abgerufen am 28. 08 2008 von <http://pvs.uni-muenster.de/pvs/lehre/SS05/seminar/stefan.pdf>
- Reinefeld, A., & Schintke, F. (2004). *Dienste und Standards für das Grid Computing*. Abgerufen am 28. 08 2008 von <http://www.zib.de/csr/Publications/2004-reinefeld-lni.pdf>

- Sabbah, D. (20. 01 2004). *Bringing Grid & WebServices Together*. (Globus, Hrsg.) Abgerufen am 28. 08 2008 von http://www.globus.org/wsrf/sabbah_wsrf.ppt
- Skonnard, A. (03 2003). *Understanding SOAP*. (Microsoft Developer Network) Abgerufen am 28. 08 2008 von <http://msdn2.microsoft.com/en-us/library/ms995800.aspx>
- Sotomayor, B. (2005). *The Globus Toolkit 4 Programmer's Tutorial*. Abgerufen am 28. 08 2008 von <http://www.chinagrid.net/grid/paperppt/progtutorial.pdf>
- Sprengel, M. (2006). *Einführung in das Web Services Resource Framework*. Abgerufen am 28. 08 2008 von http://wendtstud1.hpi.uni-potsdam.de/sysmod-seminar/SS2006/elaborations/03_WSRF_Web_Services_Resource_Framework.pdf
- Srinivasan, L., & Treadwell, J. (03. 11 2005). *An Overview of Service-oriented Architecture, Web Services and Grid Computing*. (HP, Hrsg.) Abgerufen am 28. 08 2008 von <http://h71028.www7.hp.com/ERC/downloads/SOA-Grid-HP-WhitePaper.pdf>
- Sundaram, B. (15. 03 2005a). *Understanding WSRF, Part 1*. (IBM, Hrsg.) Abgerufen am 28. 08 2008 von <https://www6.software.ibm.com/developerworks/education/gr-wsrf1/gr-wsrf1-a4.pdf>
- Sundaram, B. (22. 03 2005b). *Understanding WSRF, Part 2*. (IBM, Hrsg.) Abgerufen am 28. 08 2008 von <https://www6.software.ibm.com/developerworks/education/gr-wsrf2/gr-wsrf2-a4.pdf>
- Sundaram, B. (29. 03 2005c). *Understanding WSRF, Part 3*. (IBM, Hrsg.) Abgerufen am 28. 08 2008 von <https://www6.software.ibm.com/developerworks/education/gr-wsrf3/gr-wsrf3-a4.pdf>
- Tilkov, S., & Roth, R. (16. 10 2006). *An overview of the Web services standards landscape*. (www.innoq.com) Abgerufen am 28. 08 2008 von www.innoq.com/soa/ws-standards

Tilkov, S., & Starke, G. (2007). *SOA Expertenwissen* (1. Auflage Ausg.). Heidelberg: dpunkt.

Tsalgatidou, A. (10. 11 2005). *Web & Grid Services - Interoperability and the SODIUM Approach*. Abgerufen am 28. 08 2008 von ftp://212.190.71.100/pub/ist/docs/directorate_d/st-ds/fse_tsalgatidou.pdf

UDDI.org. (14. 11 2001). *UDDI Executive White Paper*. Abgerufen am 28. 08 2008 von Universal Description, Discovery and Integration: http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf

Vambenepe, W., Graham, S., & Niblett, P. (01. 10 2006). *Web Services Topics 1.3 (WS-Topics)*. (OASIS, Hrsg.) Abgerufen am 28. 08 2008 von http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf

Lebenslauf

von Michael Jäger

Persönliche Daten

Name: Michael Jäger
Wohnort: Dr. Helmut Czink-Gasse 2/2/30; 2230 Gänserndorf
Geburtsdaten: Wien am 15. Juni 1973
Nationalität: Österreich
Familienstand: ledig

Schulbildung

1979 – 1983 Besuch der Volksschule in Wien bzw. Gänserndorf
1990 – 1995 Besuch des Naturwissenschaftliches Realgymnasium in
2230 Gänserndorf

Universitäre Bildung

1991 – 1999 Universität Wien, Studienrichtung Wirtschaftsinformatik – 1.
Abschnitt (Spezielle: Grundzüge Privatrecht; Englisch)
2000 – 2008 Universität Wien, Studienrichtung Wirtschaftsinformatik – 2.
Abschnitt (Spezielle: Organisation&Führung, BWL der
öffentlichen Wirtschaftsunternehmen); Besondere
Vertiefung in Software-Engineering
(Webtechnologiekonzepte)

Berufserfahrung

ab 1991: Tätigkeiten im Einkauf, Beratung und Verkauf bei
internationalen IT-Unternehmen und Projektmitarbeit auf
selbständiger Basis.
Aktive Mitarbeit in einem gemeinnützigen Verein

Wien, am 29.09.2008