# Entwurf einer Security-Infrastruktur für SOA auf Basis von Web Services und IBM WebSphere Developer

Diplomarbeit

vorgelegt von
Marc Nübling

Betreuer:
Prof. Dr.-Ing. Wilhelm G. Spruth

Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Lehrstuhl Technische Informatik

15. Februar 2007

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum Unterschrift

# Danksagung

Hiermit möchte ich mich bei allen Personen bedanken, die zur Erstellung dieser Diplomarbeit beigetragen haben. An erster Stelle stehen dabei meine Eltern, die mich während meines gesamten Studiums unterstützt haben. Mein besonderer Dank gilt außerdem meinem Betreuer Prof. Dr.-Ing. Wilhelm G. Spruth und meinem Ansprechpartner Michael Herrmann von der Universität Leipzig. Weiterhin möchte ich mich auch bei Hans-Jürgen Groß und Claudia Rauscher von DaimlerChrysler und Andreas Konecny von IBM bedanken.

Der größte Dank jedoch gebührt meiner Freundin Tina, die mir während dieser gesamten Zeit immer wieder unterstützend zur Seite stand.

Für Oma Mina.

# Zusammenfassung

Sicherheit ist komplex und teuer - sie herzustellen ist eine der großen Herausforderungen der IT-Branche. Sicherheit ist gleichermaßen aber auch eine der essentiellen Anforderungen beim Austausch von sensiblen Informationen und Daten.

Frühe Programmiermodelle wie CORBA oder DCE adressierten zahlreiche Sicherheitsproblematiken und spezifizierten umfangreiche Lösungen für sichere Kommunikation in verteilten Systemen. Mit dem wachsenden Trend durch XML als Format für den Datenaustausch und Web Services als Modell für verteiltes Programmieren wuchsen jedoch neue Problemstellungen heran die auch nach neuen Sicherheitsspezifikationen verlangten. Das Zusammenspiel dieser neu ausgerichteten Spezifikationen mit den alten und bewährten Technologien erlaubt nun auch die Umsetzung komplexer Anforderungen, wie sie bspw. von serviceorientierten Architekturen formuliert werden. Eine wichtige Rolle bei der Realisierung sicherheitssensitiver Szenarien spielen die Hersteller der Entwicklungssoftware, indem sie ihre Produkte den wachsenden Standards anpassen, und durch die Integration technischer Hilfsmittel die Implementierung komplexer Sicherheitstechnologien unterstützen.

Diese Diplomarbeit hat die Möglichkeiten untersucht, inwiefern die Kommunikation zwischen verschiedenen Parteien, die durch Web Services repräsentiert werden, mittels industrieller Standards gesichert werden kann. Dabei diente das abstrakte Modell der serviceorientierten Architektur als Vorlage für einen Entwurf, der mit dem Web Service Modell konkretisiert und anhand der IBM Websphere Developer Entwicklungsplattform veranschaulicht wurde. Neben den gängigen Methoden zur Sicherung der Transportschicht wurden besonders die Möglichkeiten des WS-Security Standards als Sicherheitsmechanismus für die Nachrichtenebene untersucht.

# Inhaltsverzeichnis

# Kapitel 1: Einleitung

Den Ansatz, Softwaresysteme als serviceorientierte Architekturen (SOA) zu entwerfen, existiert bereits seit Anfang der 90er Jahre. Die technische Evolution unterstützte die Adaption in Branchen wie dem Bankwesen und der Telekommunikation, wo Fusionen und Zusammenschlüsse zu einem erhöhten Bedarf an Integration heterogener Softwaresysteme führten. Durch den aufkommenden Bereich des E-Business genoß SOA an neuem Interesse und Relevanz. Alle größeren Unternehmen haben SOA Projekte in Gang gesetzt um die tatsächlichen Vorteile, aber auch die Herausforderungen einer SOA zu evaluieren. Die schnelle Realisierung von neuen und stetig veränderlichen Geschäftsprozessen mittels flexibler Softwaresysteme ist neben der Integration von Softwarebeständen und Altsystemen in heterogene Infrastrukturen die Hauptursache für den Bedarf von SOA [Pezz06].

Internationale Unternehmen wie bspw. die DaimlerChrysler AG, untersuchen den Einsatz von SOA, und die Möglichkeiten, Applikationen auf Basis von SOA zu erstellen, deren Services über eine Vielzahl von Standorten in der ganzen Welt verteilt werden können. Dies impliziert aber nicht nur die Komplexität verteilter Systeme, sondern auch die Frage nach einem fähigen Sicherheitsmodell. Die vertrauliche Kommunikation zwischen einem Serviceanbieter und einem Servicekonsumenten über das World Wide Web oder über Unternehmensnetzwerke hinweg wird nur dann in Erwägung gezogen, wenn die Sicherheit der auszutauschenden Nachrichten und Daten gewährleistet werden kann. Für verteilte Programmierkonzepte wie RPC oder CORBA wurden in der Vergangenheit bereits umfangreiche Sicherheitslösungen erarbeitet [OPEN05] [OMG07], jedoch haben sich im Laufe der Zeit die Schwerpunkte verlagert und die Technologien erneuert. Mit der wachsenden Beliebtheit von XML hat sich die Notwendigkeit neuer Standards auf Basis von XML herauskristallisiert, und das Konzept der Web Services auf dem Markt etabliert. Jedoch adressiert die Verwendung von XML als Format für den Datenaustausch auch neue Sicherheitsrisiken, welche die Unternehmen für ihre Web Service Szenarien in Betracht ziehen müssen.

## 1.1 Problemstellung

Serviceorientierte Architekturen zielen auf die Unterstützung einer flexiblen Vereinigung verschiedener Softwaresysteme zu verteilten Systemen. Neue Geschäftsprozesse können den wachsenden und veränderbaren Anforderungen schnell angepasst werden indem sie aus existierenden Softwarestücken zusammengesetzt und orchestriert werden. Unternehmen beschäftigen sich mit der Adaption zu einer serviceorientierten Architektur, um von der Entwicklung ihrer Applikationen als verteilte Systeme profitieren zu können, um bspw. Softwarestücke aus der Fertigung, dem Vertrieb und des Managements die auf der ganzen Welt verteilt sind, zu integrieren.

Eine geographische Verteilung der Services birgt jedoch auch Sicherheitsrisiken. Die Kommunikation zwischen Dienstanbieter und Dienstkonsument findet bei einer weltweiten Verteilung der Dienste auch über besonders unsichere Netzwerke wie das frei zugängliche World Wide Web statt. Um vor Angriffen auf vertrauliche Daten vorzubeugen ist es daher notwendig, entsprechende Sicherheitsmaßnahmen zu ergreifen welche die Nachrichten auf ihrem Weg zum Ziel vor fremden Zugriffen und Manipulationsversuchen schützen. Für verteilte Systeme gibt es mit SSL/TLS, dem Kerberos Authentifizierungssystem und der X.509 Public Key Infrastruktur bereits effiziente Lösungen, jedoch sind diese für eine komplexere Service-Infrastruktur nicht immer ausreichend und skalierbar genug. Sofern sich die Absicherung der Daten ausschließlich auf den Transport von von Startpunkt zum Zielpunkt beschränkt, kann nicht garantiert werden, dass sie vor missbräuchlichen Zugriffen, z.B. bei der Weiterverarbeitung einer Nachricht an einer Zwischenstation, geschützt sind. Während Sicherheitsmaßnahmen, die bereits auf der auf Nachrichtenebene einsetzen, die Integrität und Vertraulichkeit der Daten auch bei möglichen Knotenpunkten oder auch bei der Speicherung am Zielort noch garantieren [Rose04].

Da Unternehmen verstärkt auf das Konzept der Web Services setzten um verteilte Systeme in Form von serviceorientierten Architekturen zu realisieren, und dabei das leicht les- und manipulierbare XML-basierte SOAP-Protokoll für den Datentransfer zum Einsatz kommt, besteht ein Bedarf nach Lösungen zur Sicherung des Datenaustauschs auf Nachrichtenebene. Die Implementierung wirksamer Sicherheitsmaßnahmen ist jedoch zeit- und kostenaufwendig. Entwicklungsplattformen wie WebSphere Developer von IBM, WebLogic Workshop von Bea oder .NET von Microsoft unterstützen die Implementierung von Maßnahmen zur Sicherung der Web Services Kommunikation mit ihren Produkten, die sie den wachsenden Anforderungen anpassen und indem sie die von den Standardisierungsgremien verabschiedeten Spezifikationen in ihre Lösungen integrieren [Rose04].

## 1.2 Zielsetzung

Ziel der Arbeit ist ein Überblick über das generische Konzept der serviceorientierten Architektur und der Web Services als eine mögliche konkrete Implementierung einer SOA zu vermitteln, sowie die Untersuchung Web Service relevanter Sicherheitsstandards [W3C06] [OASIS06] [IETF06] und ihrer Möglichkeiten zur Sicherung der Kommunikation serviceorientierter Szenarien. Anhand eines Beispiels soll das Konzept der Web Services vorgestellt und schließlich mit einer modernen Entwicklungsumgebung veranschaulicht werden.

## 1.3 Lösungsweg

Die Diplomarbeit beginnt mit einer Problemschilderung und der daraus abgeleiteten Zielsetzung. Kapitel 2 enthält einen Überblick über das SOA Konzept, während Kapitel 3 das Web Service Modell vorstellt. In Kapitel 4 folgen eine Aufstellung allgemeiner Sicherheitsanforderungen und deren technische Umsetzungen. Die spezifische Abbildung dieser Sicherheitsanforderungen auf das Web Service Modell bildet den Inhalt von Kapitel 5. In Kapitel 6 wird das Beispielszenario vorgestellt, anhand dessen die bisherigen Erkenntnisse veranschaulicht werden, während sich Kapitel 7 sich mit der Realisierung dieses Szenarios durch die IBM WebSphere Developer Entwicklungsplattform befasst. Abschließend folgen eine Zusammenfassung und ein Ausblick.

# Kapitel 2:
# SOA - Überblick

Dieses Kapitel enthält einen allgemeinen Überblick über das Konzept der serviceorientierten Architektur. Nach einer Definition und den Hauptmotiven für die Umsetzung einer SOA folgt eine Übersicht über die Schlüsselelemente einer SOA und die Vorstellung des „find-bind-execute“ Paradigmas. Anschließend wird der Begriff des Service definiert. Die Vorstellung der Vor- und Nachteile einer SOA und ihrer Entwicklung bilden den Abschluss dieses Kapitels.

## 2.1 Definition: SOA

Die serviceorientierte Architektur ist ein technologieneutrales Konzept dessen reale Umsetzung durch verschiedenste Ansätze motiviert sein kann. Aufgrund der vielen möglichen Herangehensweisen existieren auch entsprechend viele Definitionen, von denen folgende die Grundidee hinter SOA deutlich vermittelt [DCX06]:

*„SOA ist ein Konzept einer Softwarearchitektur, die eine oder mehrere Geschäftsfunktionen als Service repräsentiert. Die Beschreibung des Interface eines Service ist plattformunabhängig. Die Implementierung der Services ist wiederverwendbar, gekapselt und lose gekoppelt. Die Interaktionen des Service sind durch eine standardisierte Infrastruktur realisiert.“*

## 2.2 Motivation

Der Begriff der serviceorientierten Architektur beherrscht gegenwärtig die Diskussion zur Gestaltung unternehmensweiter IT-Landschaften. Ein zentraler Punkt der Diskussion ist die Reduzierung der Komplexität von IT-Systemen und die Unterstützung bei der Agilität und der Wiederverwendung von Unternehmensprozessen. Der schnelle Wandel in der IT-Branche erzwingt nicht nur das ständige Anpassen und Skalieren von bestehenden Lösungen, sondern auch die Integration von neuen Lösungen in die bereits eingesetzten Systeme und Infrastrukturen. Betriebswirtschaftliche Faktoren, Unternehmensfusionen sowie die Globalisierung und die Ad-

aption der Geschäftsprozesse durch das Einbinden der Kunden und Lieferanten beschleunigen diesen Wandel. Das Aufeinandertreffen geschäftsprozessorientierter und informationstechnologischer Interessen ist unvermeidbar. Durch nicht vorraussehbare Entwicklungen sollen IT-Systeme anhand des Konzepts der SOA miteinander verbunden werden, um schneller auf neue Bedingungen reagieren zu können [MCG06].

Für das Gros der IT-Welt war es in der Vergangenheit ausreichend, sich auf individuelle Geschäftsfunktionen und -prozesse zu konzentrieren und diese in isolierten Systemen zu implementieren. Heutzutage besteht die Herausforderung aber bei der Integration dieser Systeme in die weltumspannenden Netzwerke der Unternehmen. Es sind vor allem größere Unternehmen wie die DaimlerChrysler AG, Siemens oder T-Systems die eine Vielzahl verschiedenster Softwaresysteme und -lösungen einsetzen. Oft sind diese Applikationen sind aber nur dann effizient, wenn sie mit anderen Applikationen gekoppelt werden, und daraus neue Systeme mit erweiterten und komplexeren Funktionalitäten entstehen. Die dazu erforderlichen Programmstücke sind aber möglicherweise in anderen Sicherheitsbereichen des Netzwerkes oder auch außerhalb der Unternehmensgrenzen gelagert, und verlangen daher eine Eingliederung in das Gesamtsystem. An diese Stelle tritt die Idee der serviceorientierten Architektur, die eine einheitliche und schnelle Integration lose gekoppelter und stets wiederverwendbarer Dienste vorsieht.

Die Trägheit vieler IT-Systeme ist bedingt durch die Darstellung einer Applikation als autonomes System, welches genau einen Geschäftsprozess modelliert und implementiert, d.h. die Besonderheiten eines Prozesses sind in der Applikation fest integriert. Jede Modifikation dessen erfordert somit auch eine Modifikation des entsprechenden Programmcodes in der Applikation. Diese Methodik hemmt die Agilität von Geschäftsprozessen erheblich. Eine SOA definiert eine Reihe von Prinzipien und Praktiken um die Funktionalitäten einer Applikation flexibel widerverwendbar und lose gekoppelt zu entwerfen [Rose04].

Anhand eines Beispiels soll die Idee einer SOA veranschaulicht werden. Angenommen es handelt sich um einen Bestellvorgang, dessen einzelne Schritte als separate Services modelliert werden. Um Kosten zu reduzieren sollen einige dieser Schritte in der Prozesskette durch Services bearbeitet werden die mit anderen Applikationen geteilt werden, wie bspw. die Abrechnung eines Bestellvorganges. Der Warenbestand soll außerdem direkt beim Zulieferer verwaltet werden, und die Auslieferung des Produkts an Unternehmen wie DHL oder UPS ausgelagert werden. Mit einem serviceorientierten Ansatz kann dieser Bestellprozess implementiert und automatisiert werden indem die verbleibenden internen Services (z.B. die Abfrage die anhand von Adressen die Lieferanten und die dazugehörigen Sachnummern ermittelt) mit den korrespondierenden geteilten und ausgelagerten Services, wie in Abb 2.1 ersichtlich, zusammengesetzt werden.

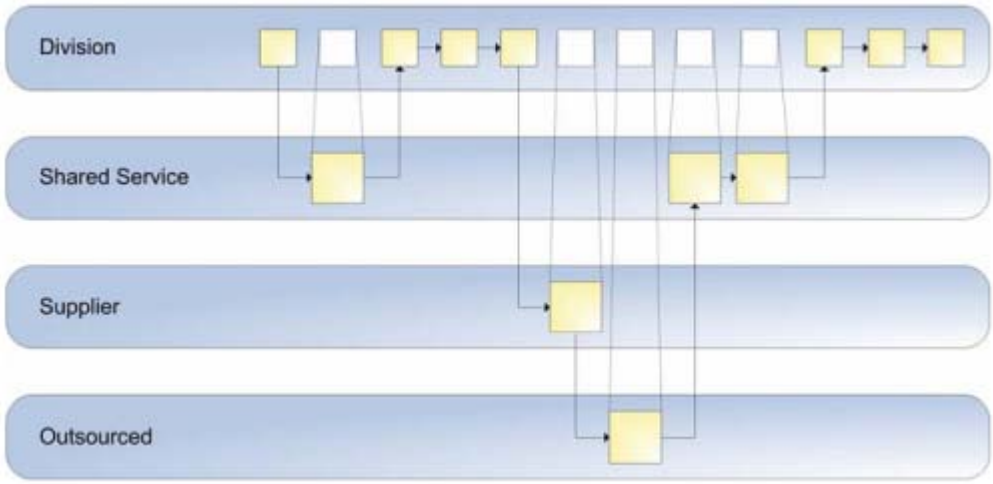



**Abb. 2.1:** Entwicklung eines Geschäftsprozesses

Der dargestellte Prozess eines Bestellvorgangs verdeutlicht außerdem, dass bei einem serviceorientierten Ansatz eine Verschiebung im Geschäftsprozess potentiell mit geringeren Kosten implementiert werden kann.

Heute haben die meisten IT-Hersteller SOA als Grundlage der nächsten Generation von Unternehmenssoftware akzeptiert. Einer Umfrage von CBDI aus dem Jahr 2003 über die Motive eine SOA im Unternehmen umzusetzen ergab, dass ungefähr 70% der Beteiligten mit der Anpassung an SOA einverstanden sind [CDBI03]. Einfachere Integration, Agilität in der Anpassung an neue Gegebenheiten und strategische Richtungswechsel sind laut Umfrage die Hauptantriebspunkte (Abb. 2.2).


**Abb. 2.2:** Motive für eine Adaption zu SOA[1]

1. Quelle: [CDBI03]

## 2.3 Schlüsselelemente

In einer serviceorientierten Architektur werden Applikationen und Geschäftsprozesse aus Services gebildet, die wiederum aus Services mit geringeren Funktionalitäten zusammengesetzt sein können. Die Schlüsselelemente einer Softwarelösung müssen serviceorientiert modelliert werden. Sie repräsentieren bestimmte Charakteristika, die wie folgt zusammengefasst werden [Bruce04]:

**1. Universelle, auf Standards basierte Konnektivität.** Bedeutet, dass die Schnittstellen der Services in einer SOA auf XML Nachrichten basieren. Für externe Services ist es daher notwendig, dass diese Schnittstellen auf Standards wie bspw. der Web Services Description Language (WSDL) basieren.

**2. Lose Kopplung.** Die technische Implementierung, der Standort und idealerweise auch das Zugriffs- und Transportprotokoll eines Service sind für den Konsumenten transparent, d.h. dass er benötigt diese nicht.

**3. Dynamisches Binden.** Ein Mechanismus der es Services erlaubt, andere Services während der Laufzeit zu identifizieren und aufzufinden, so dass Services in Echtzeit ausgewählt und ausgetauscht werden können. Diese Idee kommt bei aktuellen SOA Implementierung bisher jedoch kaum zum Einsatz.

Jede SOA benötigt einen universellen Mechanismus um alle Services zu verknüpfen die in der Gesamtlösung eines Unternehmens miteinfliessen, jedoch ohne die Sicherheit, Zuverlässigkeit oder Leistungsbandbreite zu gefährden. Diese Schicht der Infrastruktur eines Unternehmens wird heute unter dem Begriff Enterprise Service Bus (ESB) [IBM05a] geführt, auf den in dieser Arbeit jedoch nicht näher eingegangen wird da es sich dabei um ein herstellerspezfisches, proprietäres Produkt handelt.

## 2.4 „find-bind-execute“

Damit eine serviceorientierte Architektur reibungslos funktioniert, müssen die Services sowohl auf technischer als auch auf organisatorischer Ebene verwaltet werden. Auf organisatorischer Ebene ist es erforderlich, dass die Services einen eindeutigen Besitzer haben der die Erreichbarkeit, die Zuverlässigkeit, die Versionen und den Lebenszyklus verwaltet, und der, falls erforderlich, einen Abrechnungsmechanismus für die Benutzer bereitstellt. Auf der technischen Ebene müssen Applikationen und Services in der Lage sein, Services zu finden die von potentiellem Nutzen für sie sind, genau wissen was ein bestimmter Service leistet, und auch wie man ihn erreicht. Verzeichnisdienste erfüllen diese Anforderungen indem sie einen Katalog mit Serviceanbietern zur Verfügung stellen, mitsamt der Möglichkeit für den Konsumenten auf die Services zuzugreifen. In einer SOA sendet ein Kon-

sument eine Anfrage an den Verzeichnisdienst. Dieser antwortet daraufhin mit Informationen über den Service und dessen Adresse. Der Konsument verbindet sich schließlich mit dem gewünschten Service indem er dessen Adresse (endpoint) vom Serviceanbieter erhält [CEB04].

Die Benutzung solcher Verzeichnisdienste lässt den Standort des Service für den Konsumenten transparent erscheinen, und ermöglicht ihm darüberhinaus den Serviceanbieter auf Abruf auszusuchen oder auszuwechseln, was bspw. bei Business-to-Business (B2B) Applikationen nützlich wäre. Abb. 2.3 zeigt das Verhältnis zwischen Serviceanbieter, Servicekonsument und Verzeichnisdienst.

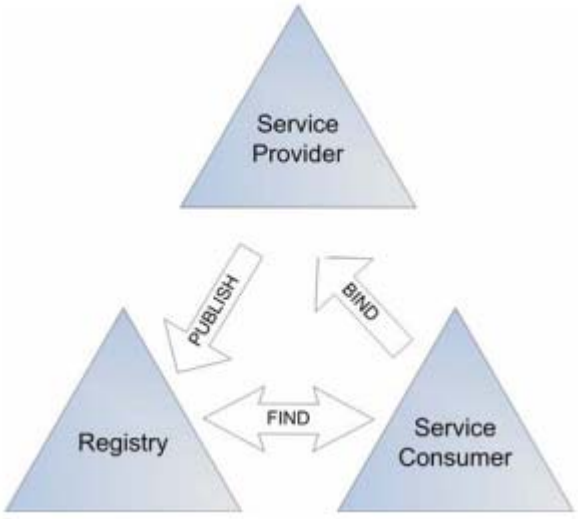



**Abb. 2.3:** Interaktion mit dem Verzeichnisdienst

Ein bekanntes Beispiel für einen solchen Verzeichnisdienst ist Universal Description Discovery and Integration (UDDI). UDDI ist ein XML-basierter Verzeichnisdienst der es Unternehmen weltweit ermöglicht, Services zu veröffentlichen und zu definieren wie diese interagieren. UDDI wurde so entworfen, dass das Verzeichnis durch SOAP Nachrichten abgefragt werden kann, und ein Zugriff auf WSDL Dokumente bereitgestellt wird. Diese beschreiben die notwendigen Protokollanbindungen und Nachrichtenformate um mit den Services zu interagieren. UDDI kann auch in der internen Infrastruktur eines Unternehmens eingesetzt werden um ein Verzeichnis bereitzustellen, welches das Auffinden und die Anbindung an die aufgelisteten Services ermöglicht. Für eine SOA innerhalb eines Unternehmens ist es sinnvoll, mindestens einen Katalog zu führen in dem alle Services aufgeführt und beschrieben sind. WSDL und UDDI sind Hauptbestandteile des Web Service Standards und werden im Kapitel 3.4 ausführlicher vorgestellt.

## 2.5 Definition: Service

So wie es viele, leicht differierende Definitionen von SOA gibt, so gibt es auch viele unterschiedliche Definitionen des Begriffs „Service“. Im Kontext von SOA und dieser Diplomarbeit ist folgende Definition treffend [Schn05]:

> *„Ein Service ist ein modularer Teil einer Software (Service Anbieter) mit einer eindeutig definierten Schnittstelle, der durch ein anderes modulares Teil einer Software (Service Konsument) aktiviert werden kann.“*

Nach IBM sollten noch einige andere Aspekte für die Definition des Service in Erwägung gezogen werden [IBM05a]:

- **Services** kapseln eine wiederverwendbare Geschäftsfunktion
- **Services** sind definiert durch eine explizite, von der Implementierung unabhängige Schnittstelle
- **Services** werden durch Kommunikationsprotokolle aufgerufen die bezüglich des Standortes und der Interoperabilität transparent sind

Folgender Abschnitt vermittelt einen genaueren Überblick über die Wiederverwendung von Services, welche Granularitätsebenen unterschieden werden können, welche Informationen die implementierungsunabhängigen Schnittstellen bereitstellen sollten, und über die Rolle der Kommunikationsprotokolle.

### 2.5.1 Wiederverwendung

Viele Beschreibungen von SOA fokussieren oft auf Geschäftsfunktionen und beziehen sich dabei auf die Verwendung grobgranularer Services. Diese Vorgehensweise ist sicher adäquat für Services die Geschäftsprozesse umsetzen und tatsächlich sehr grobgranular sind. In den meisten SOAs aber es gibt noch einige weitere sinnvolle Abstufungen der Granularitäten, die bei den sehr feingranularen, technischen Funktionen beginnen und bei den grobgranularen Geschäftsprozess-Services enden:

- Technisch funktionale Services (z.B. Benutzerauthentifizierung)
- Geschäftsfunktionale Services (z.B. einen Preis abfragen)
- Geschäftstransaktionsservices (z.B. eine Abrechnung erstellen)
- Geschäftsprozess-Services (z.B. eine Bestellung durchführen)

Zwischen jeder Abstufung der Granularität ist eine Choreographie bzw. Anpassung notwendig, um diese in einer SOA zu integrieren. Wiederverwendung kann auf jeder Granularitätsebene stattfinden, was in Abb. 2.4 illustriert wird.

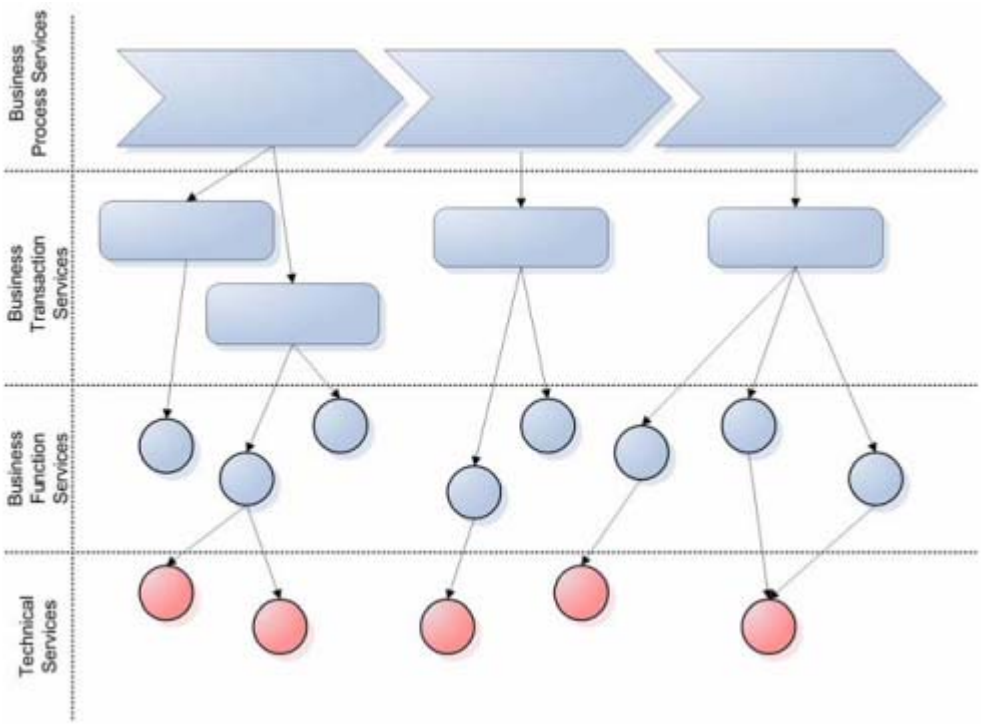



**Abb. 2.4:** Granularitätsebenen und Wiederverwendung in einer SOA

Die Art der Wiederverwendung bei einer SOA unterscheidet sich von bisherigen Ansätzen wie bspw. der komponentenbasierten Entwicklung. Die Wiederverwendung eines Java Application Programming Interface (API) etwa, bei dem während der Entwicklung entschieden wird ein bestimmtes „package“ erneut in den Quellcode einzufügen, unterscheidet sich von der Intention, die SOA bei der Wiederverwendung von Services anstrebt, und zwar bei der:

- **Laufzeit**
  Jeder Service wird an einem einzigen Ort bereitgestellt und von der Ferne aufgerufen. Der Vorteil dieses Ansatzes ist, dass Änderungen am Service nur an einer einzigen Stelle vorgenommen werden müssen.

- **Einrichtung**
  Jeder Service wird nur einmal errichtet, aber lokal, für jedes System das ihn benutzt, erneut bereitgestellt. Der Vorteil ist eine gesteigerte Flexibilität beim Erreichen von Leistungszielen oder bei der Anpassung des Service.

### 2.5.2 Granularität

In Bezug auf die Wiederverwendung von Services bedeuten die Abstufungen in der Granularität den Grad der Abstraktion. Das Granularitätskonzept einer SOA hat jedoch noch mehrere Bedeutungen [IBM05a]:

• **Abstraktionsgrad von Services**
Ist der Service ein high-level Geschäftsprozess, ein low-level Subprozess oder eine niedrige technische Funktion?

• **Granularität der Serviceoperationen**
Wie viele Operationen hat der Service? Welche Faktoren bestimmen welche Operationen in einem Service zusammengefasst werden?

• **Granularität der Serviceparameter**
Wie werden die Ein- und Ausgabedaten eines Services spezifiziert? Werden wenige, groß strukurierte Daten oder viele einfache Datentypen bevorzugt?

### 2.5.3 Transparente Schnittstellen

Die Verwendung expliziter Schnittstellen, welche die Funktionen eines Service definieren und kapseln, ist besonders wichtig für dessen Wiederverwendung. Die Schnittstelle sollte nur jene Aspekte der Verarbeitung kapseln, die in der Interaktion zwischen Servicekonsument und Serviceanbieter benötigt werden. Sie sollte also nur die für die Interaktion notwendigen Verhaltensweisen spezifizieren, jedoch nichts über die Implementierung von Konsument oder Anbieter preis geben. Durch eine Spezifikation auf diese Art und Weise ist es möglich, dass jene Komponenten der Systeme, die nicht der Interaktion zugehören (z.B. die Plattform auf der sie basieren), beliebig austauschbar sind ohne das andere System davon betroffen wird. Die Schnittstellen sollten folgendes spezifizieren [CEB04]:

- Die bereitgestellte Funktionalität (Was macht der Service?)
- Die erwarteten Ein- und Ausgabeparameter
- Vor- und Nachbedingungen (Zustand vor und nach dem Serviceaufruf)
- Fehlerbehandlung (Beschreibung der Fehler)
- Service Level Agreements (Spezifikationen über Leistung, Fehlertoleranz...)

### 2.5.4 Kommunikationsprotokolle

Es gibt eine Vielzahl von Protokollen die zum Verbinden von Applikationen verwendet werden können wie z.B. HTTP, HTTPS, JMS, CORBA oder SMTP. In grossen Unternehmen existieren oft heterogene IT-Landschaften die alle möglichen Protokolle verwenden, und es somit zu einer Herausforderung machen, die Applikationen zu verbinden die nicht über dieselben Protokolle verfügen. Es gibt verschiedene Wege diese heterogenen Applikationen zu integrieren. Ein Service sollte nur ein einziges mal durch seine implementierungs- und protokollunabhängige Schnittstelle definiert werden. Daraufhin kann er mit diversen Zugriffsprotokollen implementiert werden um die Wiederverwendung und Integration mit anderen Applikationen zu erhöhen. Eine andere Möglichkeit ist das Verpacken einer Applikation mit einer Web Service Schnittstelle, so dass sie auf standardisiertem Wege wie mit dem SOAP Protokoll benutzt werden kann [CEB04].

## 2.6 Vor- und Nachteile

Die wichtigsten Vorteile einer SOA sind die Motive die bereits in Abschnitt 2.2 beschrieben wurden. Die Wiederverwendung und die Komposition von Services ermöglichen eine gesteigerte Anpassungsfähigkeit an veränderliche Umstände und eine schnellere und günstigere Bereitstellung der Softwaresysteme.

Auf den zweiten Blick gibt es noch weitere Vorteile einer SOA. Die architekturelle Partitionierung der holisitischen Softwaresysteme zu Services mit gekapselter Funktionalität ermöglicht eine vielfältige Lebensdauer für jeden Service. Die Konnektivität verschiedener Technologien und Plattformen ermöglichen eine Synergie verschiedener Technologien und eine optimale Verortung technischer Fähigkeiten, da die spätere Benutzung eines Service unabhängig von dessen Implementierung ist. Und bevor ein Prozess aus verschiedenen Services zusammengesetzt werden kan,n muss er zunächst eindeutig identifiziert werden, was zu einer verbesserten Prozessvisibilität und Systemwartung führt.

Das inkrementelle Bereitstellen von Services im gesamten System erlaubt eine graduelle Migration neuer Technologien oder Produkte, und einer Kostenteilung der Infrastruktur und architekturellen Veränderungen über sukzessive Projekte und reduzierten Wartungskosten für die gesamte IT.

| Reuse of Services | Architectural Partitioning | Incremental Deployment |
|---|---|---|
| Greater adaptability | Diverse lifecycle "speeds" | Gradual migration |
| Faster time to deployment | Synergy of different technologies | Cost "spreading" across projects |
| Lower development cost | Optimal tech skills allocation | Reduced maintenance cost |
| | Processes visibility | |
| | Greater maintainability | |
| | Easier outsourcing/off shoring | |

**Tab. 1.1**: Vorteile von SOA[1]

SOA hat viele Vorteile (Tab. 1.1), aber auch einige Nachteile (Tab. 1.2). Um einen Service so zu entwickeln dass er tatsächlich wiederverwendbar ist, bedarf es einer formaleren Methodik und einer längeren Entwicklungszeit um die richtige Granularität und die genauen Funktionalitäten auszuwählen. SOA bedeutet außerdem eine grundlegende Veränderung innerhalb des Unternehmens, die sowohl organisatorische Veränderungen erfordert, also auch die Denkweise der IT Ingenieure umkrempelt. Eine Infrastruktur wie der ESB, die eine SOA ermöglicht, muss erst aufgebaut werden, und erfordert außerdem ein individuelles Design das nach den Bedürfnissen des jeweiligen Unternehmens zugeschnitten ist. Zusammenfassend führt die Errichtung einer SOA zu hohen Kosten im Vorfeld die womöglich nur schwer zu rechtfertigen sind. Softwaresysteme aus Services zusammenstellen bedeutet eine größere verteilte Infrastruktur zu haben, die das Testen, Debugging, Logging, Transaktionsmanagment und die Sicherheit komplexer und teuer macht.

Jede SOA bedarf einer neuen Verwaltung, wenn neue Herausforderungen wie die Zuständigkeit und Abrechnungsfähigkeit eines Service anstehen, oder die Kostenverteilung auf Servicekonsumenten und die Priorisierung von Serviceanfragen und die Konfliktlösung bei der Abwägung ähnlicher aber nicht identischer Vorraussetzungen für den Aufruf eines Service. Effektive Verarbeitungs- und Verwaltungsregeln müssen durchgesetzt werden, da der Erfolg einer SOA von all diesen Faktoren abhängt [Pezz06].

1. Quelle: [Pezz06]

| Higher Upfront Costs | More Distributed Infrastructure | Tighter Management/Governance |
|---|---|---|
| Cultural change | Extensive use of middleware | Ownership/accountability |
| Infrastructure (SOA backplane) | Transaction management | Cost allocation |
| More formal methodology | Debugging/troubleshooting | Prioritization / conflict resolution |
| Longer design time for services | End-to-end management | |
| Testing (unit/end-to-end) | More granular security | |
| | Metering/logging | |

**Tab. 1.2**: Nachteile von SOA[1]

## 2.7 Entwicklung

SOA ist kein vollständig neues Konzept. SOAs wurden schon seit den frühen 90ern umgesetzt, erreichen seitdem aber nur schrittweise den Status einer unvermeidbaren Entwicklung in der IT-Welt. SOA war ursprünglich im Bereich der peer-to-peer (P2P) Netzwerkprotokolle verbreitet, bevor es dann in der zweiten Hälfte der 90er Jahre anfing durch CORBA (Common Ojekt Request Broker Architecture), DCOM (Distributed Component Object Model) und die heranreifende MOM (Message Oriented Middleware) aufzublühen. In dieser Zeit war SOA vor allem bei Banken und Telekommunikationsunternehmen verbreitet [Pezz06].

SOA wurde jedoch erst Anfang 2000 zu einem beliebten Konzept, durch das Heranreifen von großflächig eingesetzten Entwicklungsplattformen wie J2EE (Java 2 Enterprise Edition) und .NET von Microsoft und die zunehmende Adaption von BPM (Business Process Management), und schließlich auch durch die Entwicklung der Web Services, der nächsten Weiterentwicklung einer technischen Konkretisierung des SOA Konzepts [Pezz06].

1. Quelle: [Pezz06]

# Kapitel 3:
# Web Services - Überblick

Dieses Kapitel enthält einen Überblick über das Konzept der Web Services. Nach der Definition des Begriffs folgt ein Abschnitt über die Motivation, Web Services zu implementieren, und eine Auflistung der Web Service typischen Eigenschaften. Danach werden die Grundkomponenten von Web Services vorgestellt, gefolgt von einem Rückblick auf Web Service ähnliche Konzepte. Den Abschluss des Kapitels bilden eine Vorstellung der Standardisierungsgremien und eine Übersicht über die gängigsten Web Service Standards.

## 3.1 Definition: Web Service

Für den Begriff Web Service existieren zahlreiche Definitionen [Jeck04]. Eine frühe Definition von IBM und Unisys lautete wie folgt [CWM01]:

*„Web Services are a new, standards-based approach to build integrated applications that run across an intranet, extranet, or the Internet. The approach represents a major evolution in how systems connect and interact with each other.“*

Die Definition des World Wide Web Consortium (W3C) lautet wie folgt [W3C06]:

*„Ein Web Service ist eine Software-Anwendung, die anhand eines Uniform Resource Identifier (URI) eindeutig identifizierbar ist, und deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle.“*

## 3.2 Motivation

Mit der schnellen Verbreitung des Internet wurde deutlich, dass die Infrastruktur die sich dadurch entwickelte, nicht nur dafür benutzt werden konnte um Informationen abzufragen die in Browsern dargestellt wurden, den „human-to-application“ Szenarien (H2A), sondern auch das Bedürfnis nach „application-to-application“ (A2A) Kommunikation wuchs. Es bestand zunächst die Hoffnung, dass die existierenden Protokolle und Technologien für diesen Zweck ausreichend wären, jedoch wurde schnell klar, dass dies nicht der Fall war. Das HTTP Protokoll stellt keine komplexen Operationen zu Verfügung und macht aufgrunddessen die Umsetzung von A2A Szenarien unmöglich.

Kurz vor der Jahrtausendwende veröffentlichte Microsoft das XML-basierte SOAP Protokoll, das für A2A Szenarios verwendet werden konnte. Es war zwar nur ein Protokoll unter vielen, doch aufgrund der Tatsache dass IBM anfing SOAP ab 2000 zu unterstützen, führte dies womöglich zu der breiten Akzeptanz von SOAP durch die Industrie. Zu dieser Zeit war SOAP nur ein Protokoll um komplexe A2A Szenarien umzusetzen, jedoch gewann es sehr schnell an Popularität, und es wuchsen die Bedürfnisse nach einer besseren Beschreibung und Auffindung der Services die durch SOAP implementiert wurden. Der Begriff Web Service wurde einige Monate später geprägt, als IBM, Microsoft und Ariba die Web Services Description Language (WSDL) veröffentlichten. Danach folgte die Spezifikation für UDDI, welche die Basiskomponenten der Standards und Protokolle von Web Services vervollständigte. In den darauffolgenden Jahren wurden viele Vorschläge gemacht wie die Technologie soweit verbessert werden kann, dass sie nicht nur für einfache Serviceaufrufe benutzbar ist, sondern auch in anspruchsvolleren Umgebungen funktioniert. Web Services Security (WSS) ist eine der wichtigsten Ansammlung von Standards, da sie Sicherheitslösungen adressiert, die für viele Unternehmen besonders wichtig sind [IBM06a].

In Kapitel 2 wurde bereits das Konzept der serviceorientierten Architektur erörtert, das ähnlich dem objektorientierten Programmieren, ein abstraktes Konzept ist, welches auf verschiedenste Art und Weise umgesetzt werden kann. Aufgrund ihres hohen Abstraktionsgrades ermöglichen diese Konzepte eine oft einfachere Sicht auf sonst sehr komplexe Systeme und Zusammenhänge. Häufig ist dadurch eine Bewältigung der gesteigerten Komplexität heutiger IT-Systeme möglich, und damit auch die Realisierung flexibler, robuster, sicherer und wiederverwendbarer Architekturen. Darüber hinaus gestatten SOA eine unternehmensübergreifende Integration dieser IT-Systeme, was sich in den letzten Jahren als essentielle Anforderung vieler Unternehmen herausgestellt hat [Dost05].

Da die Kosten für proprietäre Integrationslösungen immer weiter in die Höhe schießen, steigt gleichzeitig der Druck entsprechende Konzepte einzuführen die unabhängig von Programmiersprache und der verwendeten Plattform sind. Der derzeit vielversprechendste Ansatz in diese Richtung ist das Konzept der Web Services, da es auf XML basiert, und neben den theoretischen Grundlagen bereits viele fortgeschrittene Spezifikationen existieren, und zu den wesentlichen Komponenten Implementierungen vorhanden sind. Durch das Erfüllen dieser Grundvorraussetzung kann das abstrakte Konzept einer SOA mit Hilfe des Web Service Modells sowohl theoretisch veranschaulicht als auch praktisch realisiert werden [Dost05].

## 3.3 Eigenschaften

Nach IBM teilen sich Web Services folgende Eigenschaften [IBM06a]:

• **Web Services sind in sich geschlossen.**
Auf der Seite des Web Service Konsumenten ist außer XML und HTTP Unterstützung keine zusätzliche Software erforderlich. Auf Seiten des Servers ist ein HTTP und SOAP Server ausreichend.

• **Web Services sind selbstbeschreibend.**
Die Definition des Nachrichtenformats befindet sich innerhalb der Nachricht. Es sind keine zusätzlichen externen Behälter für Metadaten oder Codegenerierungstools erforderlich.

• **Web Services können über das Internet veröffentlicht, gefunden und aufgerufen werden.**
Diese Technologie benutzt leichtgewichtige Internetstandards wie HTTP zusammen mit SOAP, WSDL und UDDI.

• **Web Services sind modular.**
Web Services können durch work-flow Techniken oder durch das Aufrufen anderer Web Services zusammengesetzt werden um komplexe Geschäftsfunktionen umzusetzen.

• **Web Services sind sprachunabhängig und interoperabel.**
Client und Server können in verschiedenen Umgebungen und Sprachen implementiert werden. Um bestehende Applikationen als Web Services tauglich zu machen muss kein Code verändert werden.

• **Web Services basieren auf Standards.**
XML und HTTP bilden die technischen Grundlagen von Web Services. Ein großer Teil der Web Service Technologie wird durch open-source Projekte realisiert, was die Herstellerunabhängigkeit und Interoperabilität fördert.

• **Web Services sind lose gekoppelt.**
Bei Web Services genügt eine geringere Koordination für die gegenseitige Verbindung, und ermöglicht somit eine flexiblere Rekonfigurierung für die Integration von Services.

• **Web Services sind dynamisch.**
Mit UDDI und WSDL kann das Beschreiben und Auffinden von Web Services automatisiert werden.

• **Web Services ermöglichen einen programmatischen Zugriff.**
Der Ansatz stellt keine grafische Benutzeroberfläche bereit, sondern operiert auf Codeebene. Servicekonsumenten müssen die Schnittstelle des Service kennen, aber benötigen keine Details über dessen Implementierung.

• **Web Services können existierende Applikationen ummanteln.**
Bereits bestehende, alleinstehende Applikationen können einfach in die serviceorientierte Architektur integriert werden indem sie eine Web Service Schnittstelle implementieren.

• **Web Services bauen auf bewährten Technologien.**
Es gibt viele Gemeinsamkeiten, aber auch ein paar fundamentale Unterschiede zu den anderen Konzepten von verteilten Systemen, z.B. dass das Transportprotokoll textbasiert ist und nicht binär.

## 3.4 Die Basis von Web Services

Es gibt zahlreiche praxisreife Web Service Spezifikationen, und viele sich noch in Arbeit befindlicher Entwürfe, die alle darauf abzielen weitere standardisierte, und von Implementierung und Plattform unabhängige Funktionalitäten für Web Services bereitzustellen. Die Basis, die Web Services überhaupt möglich macht, besteht jedoch aus folgenden vier Komponenten:

• **Service Transport** ist für den Transport der Nachrichten zwischen Applikationen über ein Netzwerk verantwortlich und umfasst Protokolle wie HTTP, SMTP, FTP und das neuere Blocks Extensible Exchange Protocol (BEEP).

• **XML Messaging** unterstützt Plattforum- und implementierungsunabhängige Interoperabilität, indem die Nachrichten im verbreiteten XML Format kodiert werden. Die Protokolle sind XML-RPC, REST und das bekannteste, SOAP.

• **Service Description** beschreibt die öffentliche Schnittstelle eines Web Service, meistens mittels WSDL. Die Operationen und Nachrichten des Service werden zunächst abstrakt beschrieben und dann an ein bestimmtes Netzwerkprotokoll und Nachrichtenformat gebunden.

• **Service Discovery** zentralisiert Web Services in einem gemeinschaftlichen Verzeichnisdienst wie UDDI. Web Services können dort ihren Standort und ihre Beschreibung öffentlich machen und aufgrunddessen während der Laufzeit in Anspruch genommen werden.

Die folgenden Unterkapitel vermitteln einen Überblick über die Basiskomponenten von Web Services.

### 3.4.1 XML

XML als weit verbreitete, textbasierte Datenbeschreibungssprache bildet die Basis, auf der sich das gesamte Konzept der interoperablen, plattformneutralen und wiederverwendbaren Services stützt. Die offensichtlichen Vorteile von XML, die Unabhängigkeit der Datenrepräsentation von Applikation, Betriebssystem, Protokoll, Programmiersprache und Vokabular werfen jedoch auch für Web Services viele Sicherheitsfragen auf [Rose04].

XML Schema ist eine der Möglichkeiten, Regeln für eine bestimmtes XML Dokument zu beschreiben. XML kann unabhängig von XML Schema verwendet werden, jedoch werden bei Web Services die meisten XML Dokumente anhand eines dazugehörigen Schemas verwaltet. Schemas beschreiben die Datentypen und spezifizieren eine mögliche Ordnung der Elemente. XML Schemas sind wichtig für die automatische Validierung von XML Dokumenten, und spielen bei XML-Security eine wichtige Rolle, da dort Genauigkeit und Konsistenz entscheidend sind, und die Schemata sehr enge Vorgaben für die Struktur eines Dokuments spezifizieren. SOAP und WSDL werden durch XML Schema beschrieben, sowie auch die grundlegenden Sicherheitsstandards XML-Encryption, XML-Signature und SAML. Aufgrund der einfachen Lesbarkeit eines XML Dokuments ist die Sicherstellung der Datenintegrität eines der wichtigsten Ziele bei der Absicherung von Web Services [Rose04].

### 3.4.2 SOAP

SOAP ist das Protokoll für den Nachrichtenaustausch bei Web Services. SOAP kümmert sich um den Transport von Daten zwischen den Services oder dem Client, der einen Service anfordert. SOAP transportiert XML über standardisierte Transportprotokolle wie z.B. HTTP, und ist selbst auch durch XML definiert. Die Transportebene ist bei aktuellen Implementierungen inzwischen soweit abstrahiert worden, das sich diese beinahe beliebig austauschen lässt. Am Anfang war SOAP noch ein Akronym für „simple object access protocol“, und manchmal deutete man es auch als „service oriented architecture protocol“, doch inzwischen hat man von dieser nicht ganz korrekten Beschreibungen Abstand genommen, und belässt es bei der Abkürzung SOAP [Rose04].

SOAP besteht aus einem Envelope für die XML Nachrichten, ein uniformer Container für XML Daten. Dieser Envelope ist aufgeteilt in 2 Teilbereiche (Abb.3.1):

- **Header -** dort werden Metainformationen über die Nachricht gespeichert. Diese können Informationen über das Routing der Nachricht, über Sicherheitsmaßnahmen und/oder über die Zugehörigkeit zu einer Transaktion umfassen.

- **Body** - enthält die Nutzdaten, also die eigentliche XML Nachricht in Form von reinen Daten oder eines vollständigen XML Dokuments, oder Beschreibung der Methoden und Parameter eines RPCs (Remote Procedure Call).

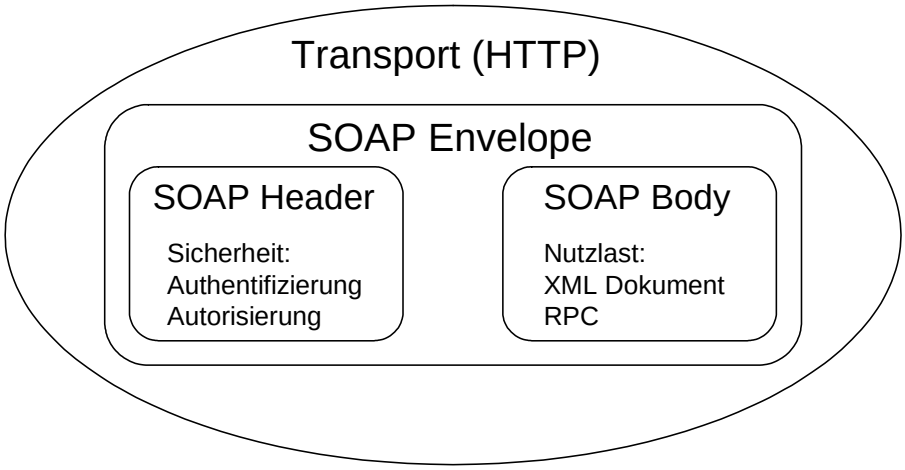



**Abb. 3.1:** Das SOAP Konzept

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <p:pizzaservice xmlns:o="http://example.org/pizzaservice">
    <p:priority>1</p:priority>
    <p:expires>2007-01-31T12:00:00-12:05:00</p:expires>
  </soapenv:Header>
  <soapenv:Body>
    <o:order xmnls:o="http://example.org/order">
      <o:pizza>salami</o:pizza>
    </o:order>
  </soapenv:Body>
</soapenv:Envelope>
```

**Codebsp. 3.1:** SOAP Envelope Struktur

Ein SOAP Envelope enthält entweder gar keinen, einen (Bsp. 3.1) oder mehrere Header, und genau einen Body, der unterschiedlicher Art sein kann und den Kommunikationsstil der SOAP Nachricht vorgibt:

- **Document** - Der Eingabeparameter ist ein XML Dokument (eine Bestellung, ein Vertrag, etc.). Ein flexibler Stil der mehr Interoperabilität gewährleistet.

- **RPC** - der synchrone Aufruf einer Operation die ein Ergebnis zurückliefert. Er besteht aus den Parametern und der Methoden die aufgerufen werden.

SOAP ist nachrichtenorientiert und schreibt nicht vor, dass eine versendete Nachricht auch eine Antwort empfangen muss. Die Beschreibung des SOAP-RPC-Protokolls ist nur ein Beispiel wie ein synchroner Aufruf nachgebildet werden kann. Dazu definiert die SOAP-Spezifikation die drei Nachrichtentypen „receive“, „response“ und „fault“. Jedoch ist es jedem selbst überlassen ein eigenes RPC-Protokoll zu definieren.

Der Nachteil an SOAP ist, dass das Protokoll durch seine XML-Basis einen erhöhten „overhead“ produziert, da SOAP die Menge der zu transportierenden Daten erheblich aufbläht. Dies beansprucht sowohl die Bandbreite des Netzwerks, als auch das Generieren und Auswerten der Nachrichten, das im Vergleich zu anderen RPC-Protokollen wesentlich mehr Rechenzeit erfordert.

### 3.4.3 WSDL

WSDL steht für Web Services Description Language und ist eine XML-basierte Sprache zur Beschreibung der Operationen die ein Web Service bereit stellt. WSDL beschreibt die Struktur des Inputs und des Outputs eines Web Service, und die zu erwartende Nutzlast seiner Nachrichten. WSDL beschreibt wie Services interagieren und sich gegenseitig in Anspruch nehmen können, wo der Service zu finden ist und was dieser Service zu leisten imstande ist. Ein WSDL Dokument besteht aus drei verschiedenen Sektionen:

- **„what“** - spezifiziert die Input- und Output Nachrichten

- **„how“** - beschreibt, wie die Nachrichten in den SOAP Envelope verpackt und transportiert, und welche Informationen in den SOAP Header eingefügt werden

- **„where“** - beschreibt die spezifische Implementierung des Web Service und die Möglichkeiten wie man dessen Endpunkt findet

Services werden in WSDL durch sechs Elemente definiert:

- **types** - Definition der Datentypen, die zum Austausch der Nachrichten benutzt werden

- **messages** - abstrakte Definitionen der übertragenen Daten, bestehen aus mehreren logischen Teilen, von denen jedes mit einer Definition innerhalb eines Datentypsystems verknüpft ist

- **portTypes** - definiert vier verschiedene Nachrichtentypen:

  - one-way (Server bekommt Input-Message vom Client)
  - request-response (Server bekommt eine Input-Message vom Client und sendet eine Output-Message zurück)
  - solicit-response (Server sendet Message und erwartet Antwort vom Client)
  - notification (Server sendet Output-Message)

- **bindings -** bestimmen das konkrete Protokoll und das Datenformat für die Nachrichten, die durch einen bestimmten Port-Typ gegeben sind

- **ports** - spezifiziert eine Adresse für eine Bindung, also eine Kommunikationsschnittstelle (üblicherweise ein URI)

- **services** - fassen eine Menge von Ports zusammen

### 3.4.4 UDDI

UDDI steht für Universal Description, Discovery and Integration und vervollständigt das Paket der Basiskomponenten von Web Services. UDDI ist ein Verzeichnisdienst für Informationen über Unternehmen, und deren Daten und Services. UDDI besitzt eine Schnittstelle für den Zugriff über SOAP Nachrichten, und gliedert seine Informationen in drei verschiedene Teilbereiche:

- **„White Pages“**
- Namensregister, sortiert nach Namen
- Auflistung der Anbieter mit allen Detailangaben
- Kontaktinformationen (Telefon, Telefax,...)

- **„Yellow Pages“**
- Branchenverzeichnis
- Spezifische Suche gemäß verschiedener Taxonomien (Ort, Dienstart,...)
- Verweist auf White Pages
- Klassifiziert die Services anhand internationaler Standards wie UNSPSC

- **„Green Pages“**
- Informationen über das Geschäftsmodell des Unternehmens
- Technische Details zu den angebotenen Web Services
- Auskunft über Geschäftsprozesse

Abb. 3.2 zeigt die Interaktion zwischen einem Servicekonsument, einem Serviceanbieter und einem Servicebroker wie z.B. ein UDDI Verzeichnisdienst.

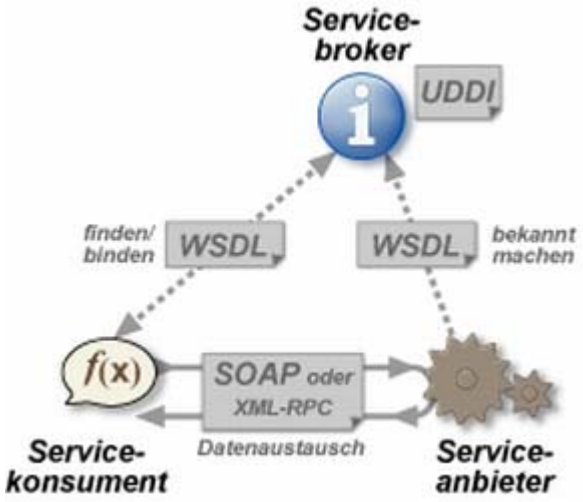



**Abb. 3.2:** Web Service Interaktion mit UDDI

UDDI ist besonders in grossen Unternehmen mit vielen wiederverwendbaren Services sinnvoll, da es das Problem löst, wie die Services angeboten werden können und aufzufinden sind. Eine SOAP Schnittstelle macht den einfachen Zugriff auf gefundene Services möglich. UDDI ist für die Realisierung einer serviceorientierten Architektur von besonderer Relevanz, da es die Anforderungen „register“, „find“ und „bind“ erfüllt. Der Zusammenhang zu Web Services stellt sich dann wie folgt dar: Der Web Service Anbieter veröffentlicht einen Vertrag in Form eines WSDL Dokuments. Dieser Vertrag wird dann mittels eines Service Brokers via UDDI registriert. Ein Web Service Konsument fragt nun den UDDI registry broker ab um den erwünschten Service zu finden, der dann wiederum auf den WSDL Vertrag verweist. Über diese Wege wird der Konsument per SOAP an den Service gebunden, so das Konsument und Service miteinander kommunizieren können [Rose04].

Sicherheit ist ein kritischer Aspekt bei UDDI. Wer ist autorisiert die Web Services zu veröffentlichen, zu benutzen, und deren Beschreibungen zu aktualisieren? Ohne entsprechende Vertrauensverhältnisse ist die Nutzung von UDDI, vor allem im Internet, bisher kaum sinnvoll möglich.

### 3.4.5 WSIL

WSDL beschreibt Services auf funktioneller Ebene, während das UDDI Schema Services eher geschäftsorientiert beschreibt. Das Ziel der Web Services Inspection Language (WSIL) bzw. WS-Inspection ist es, die verschiedenen Informationsquellen für einen Web Service in einfacher Art und Weise zu sammeln und nutzbar zu machen. WSIL ist eine ebenfalls auf XML basierende Spezifikation, welche die Zusammenführung der Referenzen auf die verschiedenen Typen von Beschreibungsdokumenten für Web Services unterstützt, und ein wohldefiniertes Muster für den Gebrauch dieser Instanzen bereitstellt, mit denen Seiten auf ihr Serviceangebot hin untersucht werden können. Die daraus gewonnenen Inspektionsdokumente werden dann beim Anbieten des Service bereitgestellt, oder können auch bspw. in HTML referenziert werden [IBM01].

Der Vorteil von WSIL gegenüber UDDI liegt darin, dass ein Serviceanbieter seine Web Service Angebote selbst bewerben kann, da WSIL das zentrale Modell der UDDI, also das Bekanntmachen eines Service im Verzeichnisdienst, dezentralisiert. Obwohl in dieser Vorgehensweise das größere Bedürfnis seitens der Unternehmen bestünde, da sich Geschäftspartner heutzutage weniger über UDDI finden, sondern eher durch bereits existierende Geschäftsbeziehungen, wird WSIL heutzutage kaum eingesetzt [IMB06a].

Abbildung 3.2 veranschaulicht das Zusammenspiel der Web Service Basiskomponenten in einer SOA.

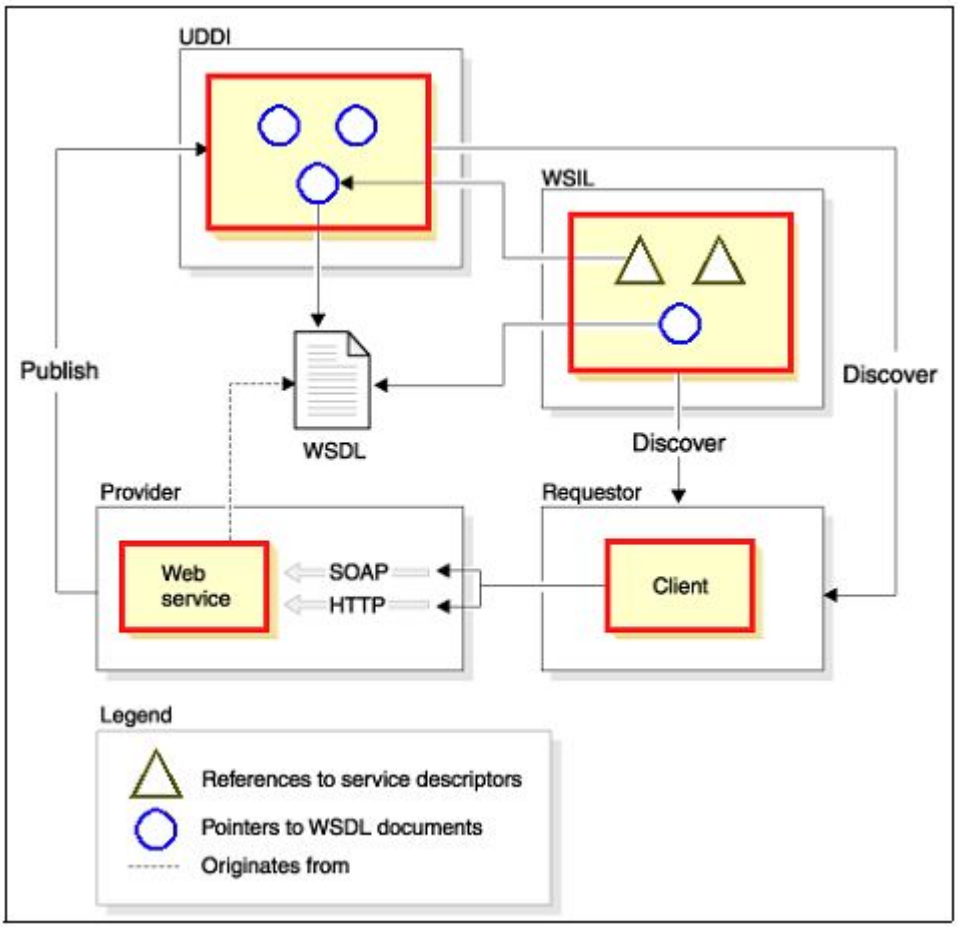



**Abb. 3.3:** Interaktion der Web Service Basiskomponenten[1]

1. Quelle: [IMB06a]

## 3.5 Rückblick

Web Services sind nicht der erste Ansatz in der Geschichte der verteilten Systeme, der in der Lage dazu wäre, das abstrakte Prinzip einer serviceorientierten Architektur in die Praxis umzusetzen. Es gab zuvor bereits etliche Programmiermodelle die ähnliche Eigenschaften wie Web Services aufweisen konnten. Zwei relevante Modelle sollen hier kurz vorgestellt werden, da sie mitunter einen grossen Einfluss bei der Entwicklung von Web Services hatten, und vor allem im Bereich der Sicherheit eine wichtige Rolle bei der Entwicklung der Web Service Spezifikationen spielten.

### 3.5.1 DCE

DCE steht für Distributed Computing Environment, und ist ein von der Open Software Foundation (OSF) entwickeltes Konzept für die Architektur verteilter Systeme. DCE bietet eine Reihe von Diensten und Werkzeugen wie Sicherheitsdienste oder Verzeichnisdienste an. DCE basiert auf dem Client-Server-Modell und benutzt den DCE Remote-Procedure-Call (DCE-RPC) für die Kommunikation verteilter Programme untereinander. DCE machte einen grossen Schritt in Richtung Vereinheitlichung von Architekturen, die zuvor alle herstellerspezifisch waren. Das DCE Modell in Software für verschiedene Plattformen umzusetzen wurde jedoch nach kurzer Zeit aufgegeben, die zugrundeliegenden Konzepte aber haben sich durchgesetzt. Seit 1996 gehört die OSF zur Open Group, wo die DCE Implementierung unter die LGPL (Lesser General Public License) gestellt wurde [OPEN05].

Die Open Group spricht auf ihrer Webseite von DCE als einen industriellen, herstellerunabhängigen Standard von distributed computing Technologien: „DCE wird von einer grossen Anzahl weltweiter Unternehmen vielen in kritischen Geschäftsumgebungen eingesetzt. Es ist ein ausgereiftes Produkt mit drei grossen Veröffentlichungen, und das einzige Middlewaresystem mit einem umfassenden Sicherheitsmodell. DCE stellt ein komplette Infrastruktur bereit. Es verfügt über Sicherheitsdienste um den Datenzugriff zu schützen und zu kontrollieren, Namensdienste die das Auffinden von verteilten Ressourcen erleichtern, und ein skalierbares Modell um weit verstreute User, Services und Daten zu organisieren. DCE läuft auf allen bekannten Plattformen und wurde entwickelt um verteilte Applikationen und heterogenen Hard- und Softwareumgebungen zu unterstützen. DCE ist eine Schlüsseltechnologie in den drei heutzutage wichtigsten Computerbereiche: Sicherheit, das World Wide Web und verteilte Objekte [OPEN05].“

### 3.5.2 CORBA

Die Common Object Request Broker Architecture (CORBA) ist eine objektorientierte Middleware, die plattformübergreifende Protokolle und Dienste definiert und von der OMG (Object Management Group) entwickelt wird. Ziel von CORBA ist das Vereinfachen der Erstellung verteilter Anwendungen in heterogenen Systemen [Rose04].

Mittels einer Interface Definition Language (IDL) erstellt man eine formale Spezifikation der Klassen und Objekte sowie sämtlicher Parameter und Datentypen. Die Beschreibung wird dann in ein Objektmodell der verwendeten Programmiersprache (die breiteste Unterstützung existiert für Java und C++) umgesetzt. Die zweite Möglichkeit verwendet clientseitig das Dynamic Invocation Interface (DII), um ohne IDL auch unbekannte Objekte ansprechen zu können. Dabei können die Methoden dynamisch abgefragt und aufgerufen werden. Serverseitig kann das Objekt dann mittels dem Dynamic Skeleton Interface (DSI) dynamisch auf die Methodenaufrufe reagieren. Die Kommunikation zweier Object Request Broker (ORB) untereinander erfolgte in CORBA 1.0 mittels eines herstellerspezifischen Protokolls. Damit auch ORBs verschiedener Hersteller aufgerufen werden können, wurde mit CORBA 2.0 das General Inter-ORB Protokoll (GIOP) festgelegt, das die Kommunikation für verschiedene Transportprotokolle definiert. Am weitesten verbreitet ist GIOP über TCP/IP, das Internet Inter-ORB Protokoll (IIOP) [OMG07].

Neben dem Protokoll definiert CORBA noch einige Dienste bzw. Services die eine definierte IDL Schnittstelle besitzen. Der wichtigste davon ist der Naming Service, eine Art Verzeichnisdienst der es den Serverobjekten ermöglicht mit festgelegten Namen angesprochen zu werden. Ein weiterer wichtiger Dienst ist z.B. der Security Service, der ungefähr ein Viertel der gesamten CORBA Servicespezifikationen einnimmt, und Schlüsselaspekte wie Authentifizierung, Autorisierung, Integrität, Vertraulichkeit und Unleugbarkeit abdeckt. Ein Grossteil der sicherheitsrelevanten Konzepte von Web Services basieren auf diesen Sicherheitsüberlegungen und -lösungen von CORBA, und man findet die gewonnen Erkenntnisse in allen Komponenten des WS-Security Pakets wieder [OMG07].

Nach den Bemühungen von Microsoft und anderen, eine objektorientierte CORBA-artige Struktur in ihrem Betriebssystem nachzubilden (z.B. DOM, DCOM....) sollte die Entwicklung von Web Services CORBA endgültig ablösen. Dies war zuerst aus mehreren Gründen offensichtlich, galt CORBA doch als zu langsam, zu komplex, unzureichend skalierbar und oftmals nicht kompatibel genug wenn es um die Implementierungen der verschiedenen Standards ging. CORBA wird seit 2004 jedoch wieder vermehrt eingesetzt, was vor allem daran liegt, dass für die netzwerk- und prozessorintensive CORBA-Struktur mittlerweile genug Rechenleistung zur Verfügung steht, und ein ORB nicht zustandslos vorgeht, wie es bei Web Services der Fall ist wenn diese das HTTP Protokoll zur Datenübertragung verwenden.

## 3.6 Standardisierungsgremien

Mit der wachsenden Beliebtheit und Ausweitung des Internets haben sich diverse Gremien herausgebildet und etabliert, die meist auf Initiative von namenhaften Unternehmen gegründet und finanziert werden. Da Web Services offene Industriestandards sind, werden sie auch von verschiedenen Organisationen und Unternehmen verabschiedet. Die Ergebnisse dieser Arbeit werden oft als Standards bezeichnet, die im Unterschied zu einer Norm jedoch nicht bindend sind sondern lediglich eine Empfehlung darstellen. Das W3 Konsortium spricht deshalb auch von „recommendations", da das W3C keine zwischenstaatliche Organisation ist und aufgrunddessen auch keine Berechtigung hat offizielle Standards zu verabschieden. Nicht immer ist die Standardisierungsarbeit dieser Gremien auch von Erfolg gekrönt. Erst kürzlich fasste das Computermagazin „c't" diverse Artikel zusammen, die von einer Krise des W3C aufgrund praxisfremder und jahrelang verschleppter Standards berichten [CT07].

Die wichtigsten Gremien in Bezug auf Web Services, Web Service Security und Web Service Interoperability werden im folgenden kurz vorgestellt:

- **IETF**
  Die Internet Engineering Task Force ist eine offene, internationale Freiwilligenvereinigung von Netzwerktechnikern, Herstellern, Netzbetreibern, Forschern und Anwendern die sich vor allem mit den Internetprotokollstandards befasst [IETF06].

- **OASIS**
  Die Organization for the Advancement of Structured Information Standards ist eine internationales Konsortium, das sich mit der Entwicklung von Web Service Standards beschäftigt. An OASIS sind u.a. Firmen wie Sun, IBM, Nokia und SAP beteiligt [OASIS06].

- **W3C**
  Das World Wide Web Consortium ist ein Gremium, das interoperable Spezifikationen, Richtlinien, Software und Tools für das Internet entwickelt und an Standardisierungen von Internet-Technologien arbeitet [W3C06].

- **WS-I**
  Die Web Services Interoperability Organization ist ein Versuch der Industrie, Web Service Interoperabilität zwischen den verschiedenen Plattformen, Applikationen und Programmiersprachen zu fördern. Mitwirkende sind u.a. Sun, IBM, HP, SAP, Microsoft und Intel [WSI06].

## 3.7 Web Service Standards

Es gibt inzwischen unzaehlige Web Service Standards, und es ist nicht immer auf Anhieb verständlich wie diese Standards gruppiert sind und wie ihr Verhältnis zueinander ist. Es existiert leider kein einfacher Web Service protocol stack, der eine simple Kategorisierung der Standards erlauben würde, da die zugrundeliegenden Konzepte des Nachrichtenaustauschs für mehr als nur den Nachrichtentransport und Serviceaufrufe benutzt werden können. Viele dieser Standards decken außerdem mehr als nur einen Aspekt der Web Service Technologie ab. Die Entwicklung der Web Service Standards geht schnell voran, und es werden ständig neue Spezifikationen erarbeitet [IBM06a].

Tabelle 3.1 listet die gängigsten Web Service Standards auf, die zur besseren Übersicht in lose zusammenhängende Kategorien eingeteilt und mit Informationen zum aktuellen Status versehen wurden.

| Standard | V | Status | Entwickler |
|---|---|---|---|
| **Core** | | | |
| • SOAP | 1.2 | Recommendation 06/2003 | W3C |
| • WSDL | 2.0 | Recommendation 03/2006 | W3C |
| • UDDI | 3.0 | Draft 10/2004 | IBM, MS, SAP, u.a. |
| • XML | 1.1 | Recommendation 09/2006 | W3C |
| **Description and Discovery** | | | |
| • WS-Inspection (WSIL) | 1.0 | Draft 11/2001 | IBM, MS |
| • WS-Discovery | | | MS, BEA, Intel, Canon |
| • WS-MetadataExchange | 1.1 | Draft 08/2006 | IBM, MS, Sun, SAP, u.a. |
| • WS-Policy | 1.2 | Draft 03/06 | IBM, MS, BEA, SAP, u.a. |
| **Messaging** | | | |
| • ASAP | 1.0 | Draft 05/2005 | OASIS |
| • SOAP with Attachments | | Note 12/2000 | MS, HP |
| • SOAP MOTM | | Recommendation 01/2005 | IBM, MS, Bea, Canon |
| • WS-Adressing | | Recommendation 09/2005 | IBM, MS, BEA, SAP, u.a. |
| • WS-Notification | 1.3 | Standard 10/2006 | OASIS |
| • WS-Eventing | | Draft 08/2004 | IBM, MS, BEA, SUN, u.a. |
| • WS-Enumeration | | Submission 03/2006 | MS, BEA, Sonic, u.a. |
| • WS-MessageDelivery | 1.0 | Submission 04/2004 | Oracle, Arjuna, Nokia, u.a. |
| • WS Reliable Messaging | 1.1 | Standard 11/2004 | OASIS |
| • WS-Resources | 1.2 | Standard 04/2006 | OASIS |
| • WS-Transfer | | Submission 09/2006 | MS, BEA, Sonic, u.a. |
| **Management** | | | |
| • WSDM | 1.0 | Draft 02/2006 | OASIS |
| • WS-Manageability | | Submission 09/2003 | IBM, u.a. |
| • SPML | 1.0 | Standard 10/2003 | OASIS |
| • WS-Provisioning | | Submission 10/2004 | IBM |

| Standard | V | Status | Entwickler |
|---|---|---|---|
| **Business Processes** | | | |
| • WS-BPEL | 2.0 | Draft 05/2006 | IBM, MS, BEA, SAP, u.a. |
| • WS-CDL | 1.0 | Recommendation 11/2005 | W3C, Oracle, Novell, u.a. |
| • WS-CAF | 1.0 | Draft 10/2005 | OASIS |
| **Transactions** | | | |
| • WS-Coordination | 1.1 | Draft 03/2006 | IBM, MS, BEA |
| • WS-Transaction | 1.0 | Consideration 08/2005 | IBM, MS, BEA |
| • WS-AromicTransaction | 1.0 | Consideration 08/2005 | IBM, MS, BEA |
| • WS-BusinessActivity | 1.0 | Consideration 08/2005 | IBM, MS, BEA |
| **Security** | | | |
| • XML-Encryption | | Recommendation 12/2002 | W3C |
| • XML-Signature | | Recommendation 12/2002 | W3C, IETF |
| • WS-Security | 1.1 | Standard 02/2006 | OASIS |
| • WS-SecureConversation | | Draft 02/2005 | IBM, MS, BEA, RSA, u.a.. |
| • WS-SecurityPolicy | 1.1 | Draft 07/2005 | IBM, MS, RSA, VeriSign |
| • WS-Trust | | Draft 02/2005 | IBM, MS, BEA, RSA, u.a. |
| • WS-Federation | 1.1 | Draft 12/2006 | IBM, MS, BEA, RSA, u.a. |
| • SAML | 2.0 | Standard 03/2005 | OASIS |

**Tab. 3.1:** Web Service Standards

Die Auflistung erhebt keinen Anspruch auf Vollständigkeit. Der Kosmos der Web Service Standards entwickelt sich schnell, und es werden ständig neue Spezifikationen verabschiedet (auch wenn letztendlich nicht immer alle in der Praxis zum Einsatz kommen). Neben den übergreifenden Industriestandards die bereits existieren, und jenen die momentan definiert werden, gibt es noch weitere Standards die jedoch nur für bestimmte Industriezweige relevant sind, und auf die hier auch nicht weiter eingegangen wird.

# Kapitel 4:
# Sicherheit - Grundlagen

Dieses Kapitel erläutert die für das Web Services relevanten Sicherheitsgrundlagen. Zunächst werden allgemeine Sicherheitsanforderungen wie Authentifizierung, Autorisierung etc. spezifiziert, und dann auf konkrete Sicherheitskonzepte wie Verschlüsselung, digitale Signaturen, SSL/TLS etc. eingegangen.

## 4.1 Sicherheitsanforderungen

Sicherheit definiert sich durch eine Vielzahl verschiedener Anforderungen, von denen die Wichtigsten kurz vorgestellt werden sollen. Die Anforderungen werden zunächst neutral formuliert, bevor sie dann in den folgenden Unterkapiteln anhand der gängigen Sicherheitskonzepte technisch spezifiziert werden, bevor in Kapitel 5 eine Abbildung dieser Anforderungen auf das Web Service Modell stattfindet.

### 4.1.1 Authentifizierung

Authentifizierung ist der Prozess, Informationen über eine Identität abprüfen zu lassen, von einer Instanz, die bereits Informationen über diese Identität gespeichert hat. Authentifizierung wird klassischerweise in drei verschiedene Typen eingeteilt:

- **„etwas das man weiß“:** PIN, Passwort, gemeinsames Geheimnis...
- **„etwas das man hat“:** Schlüssel, Karte, Token...
- **„etwas das man ist“:** Fingerabdruck, Retinascan...

Desweiteren wird zwischen „single-factor“ Authentifizierung, „two-factor“ Authentifizierung, und in selteneren Fällen auch noch „three-factor“ Authentifizierung unterschieden. Bei der „single-factor“ Authentifizierung ist nur einer der drei Typen erforderlich. Sie ist zwar am einfachsten, aber auch die unsicherste Methode, da ein bspw. ein Passwort erraten werden kann.

Bei der „two-factor“ Authentifizierung, die auch starke Authentifizierung genannt wird, sind zwei der drei Typen erforderlich. Sie ist wesentlich sicherer und gilt auch als Standard beim Zugriff auf wertvollere Informationen. „etwas das man hat“ oder „etwas das man ist“ wird dabei mit „etwas was man weiß“ kombiniert [Rose04].

### 4.1.2 Autorisierung

Autorisierung ist der Prozess der Zugriffskontrolle bei einem bereits Authentifizierten. Der Empfänger der Serviceanfrage verteilt den authentifizierten Identitäten die Erlaubnis auf bestimmte Ressourcen zugreifen zu können. Autorisierung erfordert in der Regel zunächst eine erfolgreiche Authentifizierung. Sofern ein Service verschiedene Zugriffsebenen bereitstellt, je nach dem wer ihn in Anspruch nimmt, ist die Authentifizierung erforderlich, damit der Service anhand der Identität des Aufrufenden bestimmen kann inwieweit welche Services für wen zugänglich sind. Im Gegensatz zu anderen Sicherheitsanforderungen ist Autorisierung zwar einfach zu verstehen, die technische Umsetzung jedoch ist sehr komplex.

Eine Möglichkeit Autorisierung zu implementieren besteht darin, ein Set von Berechtigungsnachweisen, „credentials“, zu erstellen, die von der Identität dann mitgeführt werden, und, wann immer nötig, demjenigen präsentiert werden, dessen Dienste die Identität in Anspruch nehmen möchte. Anhand dieser credentials wird dann eine Zugriffskontrolle vorgenommen. Alternativ könnten bestimmte Zugriffsrechte an bestimmte Inhalte gebunden werden, und die Rechte werden dann wiederum auf die Identität und deren Berechtigungen für diesen Inhalt abgebildet. Beim HTML-basierten Web erfolgt die Autorisierung typischerweise grobgranular, d.h. der Zugriff auf eine Ressource wird entweder vollständig erlaubt oder ganz verwehrt. Bei WS-Security dagegen ist eine feingranulare Zugriffskontrolle auf Nachrichten bzw. deren Inhalte möglich [Rose04].

### 4.1.3 Integrität

Integrität ist eine Zusicherung, dass niemand eine Nachricht verändert hat seit sie erstellt wurde. Integrität stellt dem Sender einer Nachricht sicher, dass der Empfänger exakt dieselben Daten erhält. Integrität im kryptograpischen Sinne wird durch die Verwendung von digitalen Signaturen erreicht. Nachrichten, die Integrität erfordern, müssen explizit oder implizit die Identität und credentials des Senders beinhalten. Integrität nachzuweisen bedeutet zu beweisen, dass kein Bit der Nachricht verändert wurde. Dabei ist erforderlich, dass etwas bestimmtes mit der Nachricht mitgeschickt wird, um einen Missbrauch durch einen potentiellen Mittelsmann zu verhindern. Das wiederum verlangt danach, dass die Daten mit einem Schlüssel verschlüsselt werden, den nur der Sender haben kann [Rose04].

### 4.1.4 Vertraulichkeit

Vertraulichkeit bedeutet die Geheimhaltung einer Nachricht. Dieser Prozess erfordert Verschlüsselung, welche die Nachricht soweit unlesbar macht für Dritte, dass nur noch autorisierte Identitäten den Text entschlüsseln können um die Daten zu lesen. Dazu ist erforderlich, dass zwischen der sendenden und der empfangenden Partei ein Geheimnis in Form eines Schlüssels ausgetauscht wird, um die Nachricht entsprechend verschlüsseln bzw. entschlüsseln zu können.

Typischerweise werden die Algorithmen, die heutzutage zur Verschlüsselung verwendet werden veröffentlicht, da man sich auf die Stärke der Schlüssel verlässt. Das Problem jedoch ist, dass der Schlüssel über sichere Wege ausgetauscht werden muss, da er sonst abgefangen und missbraucht werden kann. Eine Lösung ist die Public Key Kryptographie, welche Mechanismen für den sicheren Schlüsseltausch zur Verfügung stellt [Rose04].

### 4.1.5 Unleugbarkeit

Unleugbarkeit bedeutet nachweisen zu können, dass eine Identität A einer Identität B Daten gesendet hat. Es kann damit bewiesen werden, dass die Transaktion durch Identität A eingeleitet wurde, und keine der beiden Beteiligten wiederlegen oder verneinen kann, dass sie auch tatsächlich stattgefunden hat. Sofern die Transaktion dennoch angefochten wird, muss der Vertrag, der von beiden Beteiligten unterzeichnet wurde, vorgezeigt werden. Jede Seite muss den Vertrag unterzeichnet gesehen haben, und ihre Identitäten müssen zum Zeitpunkt der Signierung für gewöhnlich durch einen anwesenden Notar (der hier die Funktion eines Zeugen übernimmt) bestätigt worden sein. In der anonymisierten Welt des Internets jedoch ist es schwierig bis unmöglich, diese Vorraussetzungen zu schaffen.

Unleugbarkeit basiert auf der Technologie der Public Key Infrastruktur. Man beweist, dass eine Identität A Daten an eine Identität B geschickt hat, da Identität A den public key von B benutzt hat um seine Nachricht damit zu verschlüsseln. Es ist daher nur Identität B möglich, mit dem dazugehörigen private key die Daten wieder zu entschlüsseln. Das ist die einfachere Variante, und für eine strengere Form der Unleugbarkeit sind separate, notarielle Datum- und Zeitstempel erforderlich um zu belegen wann die Transaktion stattgefunden hat, sowie einer unabhängigen Verifikation der teilnehmenden Identitäten. Man unterscheidet daher zwischen der echten und kryptographischer Unleugbarkeit [Rose04]:

- **Kryptographische Unleugbarkeit -** bedeutet, das eine Identität nicht behaupten kann, sie habe eine digitale Signatur nicht erstellt. Durch Ausstellen eine public key Zertifikats durch eine Certification Authority (CA) wird nämlich der public key dieser Identität an die Person gebunden. Diese Definition sagt jedoch

nichts darüber aus, inwieweit ein Mensch die Erstellung der digitalen Signatur veranlasst hat.

• **„Echte“ Unleugbarkeit** - bedeutet, einem Dritten die Intention oder das Verhalten einer anderen Person zu beweisen: Identität A beweist dem Richter, dass Identität B plant ein Dokument zu unterzeichnen. Hardware und Software sind nicht fähig diesen Beweis zu liefern. Das einzige was bewiesen ist, ist das Verhalten eines kryptographischen Schlüssels auf einem Computersystem. Zwischen diesem System und der Person befinden sich jedoch die Hard- und Softwareschichten, welche die Person anonymisieren. Sie könnten z.B. auch unter der Kontrolle eines Angreifers sein.

Das Problem mit dem Begriff der Unleugbarkeit ist auf die ursprüngliche Definition zurückzuführen, welche sich auf ein Verhalten bzw. die Geistesverfassung eines menschlichen Wesens bezieht, die Technologie aber jene der Computer, Prozesse und kryptographischen Schlüssel ist, und es keine beweisbare Verbindung zuwischen Mensch und Computer gibt. Außerdem gibt es keine Hardware bei der Standardimplementierung eines public-key Systems, um menschliches Verhalten aufzuzeichnen und als Zeuge dieser Aktionen auftreten zu lassen. Selbst wenn es diese Hardware gäbe existiert kein Präzedenzrecht, das solche Hardware vor Gericht aussagefähig wäre. Der Grund, wieso es daher keine echte Unleugbarkeit gibt, liegt an der falschen logischen Schlussfolgerung der Definition. Fakt ist, dass eine Person ein Programm starten kann um eine Nachricht anzeigen zu lassen, und wiederum ein anderes Programm starten kann um einen privaten Schlüssel zu verwenden um eine digitale Signatur dieser Nachricht anzulegen. Bei kryptographischer Unleugbarkeit könnte jeder mit dieser Nachricht, der digitalen Signatur oder dem privaten Schlüssel beweisen, dass die Signatur durch eine exakte Kopie der Nachricht erstellt wurde. Die falsche Logik ist nun, dass auch die anderen Schritte in diesem Prozess offensichtlich nicht invertierbar sind [Cull00].

### 4.1.6 Sonstige

Weitere (schwächere) Sicherheitsanforderungen sind:

• **logging -** anhand der Aufzeichnungen von Ereignissen können sicherheitsrelevante Entscheidungen getroffen werden.

• **auditing -** die Nachprüfbarkeit erlaubt Informationen über durchgeführte Transaktionen in „audit trails“ festzuhalten, die später wiederhergestellt und analysiert werden können.

## 4.2 SSL/TLS

Secure Socket Layer (SSL) bzw. Transport Layer Security (TLS) ist das am weitesten verbreitete und am häufigsten eingesetzte Protokoll zur Sicherung des Nachrichtentransfers auf Transportebene. TLS 1.0 und 1.1 sind die standardisierten Weiterentwicklungen von SSL 3.0 (die Unterschiede sind jedoch gering). SSL ist außerdem die am meistgebrauchte Implementierung einer Public Key Infrastruktur (PKI). SSL ist auch für Web Services relevant, da es einfach zu benutzen ist, es beinahe überall eingesetzt wird. Im Grunde unterstützten alle Browser SSL, sowie auch alle Web Server bzw. Web Application Container, wie z.B. BEAs Web Logic Server oder IBMs WebSphere Application Server. Alle aktuellen Web Services Container (.NET, WebLogic Workshop, IBM WebSphere Developer) sind außerdem auch Application Container, daher kann die bereits vordefinierte Transport Layer Security für Web Services einfach verwendet werden [Rose04].

SSL stellt Vertraulichkeit bei Transaktionen sicher. Es implementiert eine shared key Verschlüsselung zwischen den beiden Kommunikationsendpunkten, nachdem es den shared key per public key Verfahren transportiert hat. SSL ist nützlich bei weniger komplexen Web Service Szenarien bei denen eine einfache point-to-point Verschlüsselung und der Authentifizierungsgrad, den SSL bereitstellt, ausreicht. SSL kann jedoch auch als zusätzliche Sicherheit auf der Transportschicht dienen, unterhalb der Sicherheit auf Nachrichtenebene. SSL wird hauptsächlich mit HTTP verwendet (HTTPS).

## 4.3 Digitale Signaturen

Digitale Signaturen werden dazu verwendet um Identifikation und Integrität herzustellen. Digitale Signaturen machen sich mathematische Einwegfunktionen zu Nutze, das „hashing“, gefolgt von einer Verschlüsselung per public key Verfahren. Eine Hashfunktion wird verwendet um einen „message digest“ zu erstellen, d.h. einen stellvertretenden Fingerabdruck der Nachricht von fixer Länge. Dieser Fingerabdruck wird dann verschlüsselt. Der message digest ist eine, normalerweise 20 Byte große Repräsentation der gesamten Nachricht. Dieser Vorgang der Erzeugung eines message digest ist insofern notwendig, da die public key Verschlüsselungsverfahren sehr langsam, und in der Größe der zu verschlüsselnden Daten limitiert sind. Sicherheit wird insofern gewährleistet, als dass die mathematische Einwegfunktion keine Invertierung des message digest erlaubt, sich daraus also nicht die vollständige Nachricht wiederherstellen lässt. Hashfunktionen sind schnell, und erzeugen keine Kollisionen, d.h. sie finden niemals denselben Hashwert für zwei verschiedene Nachrichten. Diese Eigenschaft ist wichtig, da sonst ein Angreifer die Nachricht austauschen und denselben Hashwert daraus erzeugen könnte.

SHA1 (Secure Hash Algorithm 1) ist der Standard für sichere Einweg-Hashfunktionen mit geringer Kollisionsdichte. Im Jahr 2005 ist es Kryptographieexperten mit Hochleistungstrechnern zwar gelungen, den Aufwand der Kollisionsberechnung von $2^{80}$ auf $2^{63}$ zu verringern, dennoch dürfte die Gefahr des Missbrauchs in der Praxis weiterhin äußerst minimal sein, da eventuelle Angreifer eine zweite sinnvolle Variante des signierten Dokuments erzeugen müssten, die sowohl den gleichen SHA-Wert hat als auch wieder die gleiche Signatur ergibt.

Wenn die Nachricht einer Identität A mit deren private key verschlüsselt wird, und Identität B diese Nachricht mit dem public key von Identität A lesbar macht, kann Identität B beweisen, dass diese Nachricht nur von Identität A stammen kann (sofern deren private key geheimgehalten wurde). Diese Identifikation des Senders ist ein Teil der Funktion von digitalen Signaturen. Der andere Teil stützt sich auf das Ziel, die Integrität von Nachrichteninhalten verifizieren zu können. Sofern man die gesamten Klartextnachricht hashed, und dann den daraus resultierenden message digest vor jedweden Modifikationen schützt. Und wenn der Sender und der Empfänger exakt dieselbe Nachricht und denselben Hashalgorithmus zur Verfügung haben, dann kann der Empfänger die Integrität der Nachricht prüfen, ohne versuchen zu müssen die gesamte Nachricht zu entschlüsseln.

Der message digest wird geschützt, indem man ihn einfach mit dem private key des Senders verschlüsselt, und die Originalnachricht zusammen mit dem verschlüsselten message digest an den Empfänger schickt. Public Key Verschlüsselung des message digest stellt die kryptographische Unleugbarkeit sicher, da nur jene Identität mit dem private key die Verschlüsselung veranlassen und die Nachricht initialisieren konnte. Der message digest wird verschlüsselt um zu verhindern, dass kein Mittelsmann ihn modifizieren kann.

## 4.4 Kryptographie

Der erste und wichtigste Bestandteil der Kryptographie ist Verschlüsselung. Verschlüsselung ist die Basis von XML-Encryption und XML-Signature, auf die im nächsten Kapitel noch näher eingegangen wird. Die Funktion der Verschlüsselung für digitale Signaturen wurde bereits in Kapitel 4.3 erläutert.

Das Ziel von Verschlüsselung ist es Vertraulichkeit von Daten sicherzustellen, d.h. aus einer Klartextnachricht einen verschlüsselten Text zu erzeugen der unlesbar ist bzw. keinen Sinn mehr ergibt, sofern man nicht den entsprechenden Schlüssel hat um daraus wiederum den Klartext zu rekonstruieren. Shared key Verfahren benutzen sowohl für die Verschlüsselung als auch für die Entschlüsselung denselben Schlüssel, der auch oft symmetric key genannt wird. Dieses Verschlüsselungsverfahren ist relativ schnell im Gegensatz zum public key Verfahren, bei dem jeweils verschiedene, aber mathematisch voneinander abhängige Schlüssel (public key und

private key) für die Ver- und Entschlüsselung verwendet werden. Dieses Verfahren wird aufgrund der schlechteren Performance primär dazu verwendet, die Schlüsselübergabe beim shared key Verfahren oder digitale Signaturen abzusichern, bei denen nur relativ kleine Datenmengen zu verschlüsseln sind, wie bspw. einen symmetrischen Schlüssel oder einen message digest.

## 4.5 Kerberos

Die Aufgabe des Kerberossystems ist die Echtzeit-Authentifizierung in einer unsicheren Umgebung. Es ist eine Technologie die bereits entwickelt wurde als das Internet, in der Form wie man es heute kennt, noch gar nicht existiert hat. Kerberos ist ein „trusted third-party“ Protokoll zur Authentifizierung von Kommunikationsteilnehmern. Benutzer und Dienste können im Kerberossystem ein „ticket“ erwerben, ein „token“ das eine Identität repräsentiert und zur Authentifizierung herangezogen wird, als auch shared keys für eine sichere Kommunikation, die nur dem Server bekannt sind der auch die tickets verschlüsselt. Das Kerberos Modell basiert auf einem zentralisierten Authentifizierungsserver der das Schlüsselmanagment und die Administration übernimmt. Er speichert die shared keys aller Benutzer, und generiert „session keys“ sobald zwei Benutzer sicher miteinander kommunizieren möchten. Außerdem authentifiziert das System die Identitäten von Benutzern, die Zugriffe auf gesicherte Netzwerkdienste anfordern - diese benötigen dafür ein gültiges Passwort und ein ticket [MIT07].

Microsoft hat das Kerberossystem in das Sicherheitsmodell von Windows integriert. Kerberos ist zwar nur in geschlossenen Systemen sinnvoll, in denen jeder Teilnehmer einen direkten Zugriff auf den zentralen Key Distribution Center (KDC) hat, es kann aber auch vorkommen, dass bspw. Windowssysteme via Web Services externe Services nutzen möchten. Daher sollten diese externen Systeme ihre Authentifizierungsmechanismen auf jene von Kerberos abbilden können, damit diese überhaupt miteinander interagieren können. Für diesen Fall gibt es eine WS-SecurityKerberos Spezifikation [IBM03].

## 4.6 PKI

Eine Public Key Infrastruktur (PKI) ist die Grundvorraussetzung für den Gebrauch von digitalen Signaturen und public key Verschlüsselungsmethoden. Mit entsprechenden Technologien lassen sich durch PKI die Sicherheitskonzepte der Integrität, Unleugbarkeit, Vertraulichkeit und Authentifizierung für Web Services herstellen.

Public Key Systeme arbeiten mit Schlüsselpaaren: dem private key, der strikt geheimgehalten wird, während der public key frei verteilt wird, bzw. dem anderen Teilnehmer zur Verfügung gestellt wird, um eine sichere Kommunikation zu gewährleisten. Bei bidirektionaler, vertraulicher Kommunikation ist von beiden Teilnehmern das Schlüsselpaar erforderlich. Die public keys werden dann gegenseitig ausgetauscht, so dass beide Teilnehmer die Möglichkeit haben, mit dem jeweiligen public key des Empfängers die Nachricht zu verschlüssen. Da die Schlüssel mathematisch voneinander abhängen ist es also nur demjenigen möglich, die Nachricht zu entschlüsseln, der auch den passenden private key dazu besitzt. Verschlüsselt eine Identität A eine Nachricht mit ihrem private key, und hat der Empfänger B Zugriff auf den dazugehörigen public key, kann dadurch bewiesen werden, dass die Nachricht nur von Identität A kommen konnte, d.h. damit lässt sich die Identität des Senders herstellen - das Prinzip von digitalen Signaturen [Rose04].

Die Verifikation von digitalen Signaturen setzt voraus, dass der Empfänger den public key des Senders erhält. Dieser Schritt ist besonders wichtig um ein Vertrauensverhältnis zwischen dem Sender und dem Empfänger herzustellen. Der Empfänger muss sich auf das Vertrauen, das im public key mit inbegriffen ist, verlassen, dass er auch wirklich von diesem Sender kommt, und dieser den dazugehörigen private key entsprechend unter Verschluss hält. An dieser Stelle kommt nun die Public Key Infrastruktur zum Einsatz, die sich um Zertifikate und Zertifizierungsstellen kümmert um genau diese Vorraussetzungen und Anforderungen zu gewährleisten. Der public key eines unbekannten Senders alleine reicht jedoch nicht aus um eine Vertrauensbasis zwischen ihm und dem Empfänger herzustellen, denn es fehlen noch Informationen über seine Identität die mit dem public key verknüpft sind. Außerdem sollte man wissen ob ein Dritter, dem man vertrauen kann, dessen Identität bereits verifiziert hat [Rose04].

Ein digitales Zertifikat ist eine Datenstruktur, die Informationen über eine Identität und deren public key beinhaltet. Es wird außerdem durch eine CA signiert. X.509 ist der derzeit gängigste Standard für digitale Zertifikate. Durch das Signieren des Zertifikats bürgt die CA für die entsprechende Identität im Zertifikat. Dadurch können nun die Nachrichtenempfänger dem public key, der in dem Zertifikat enthalten ist und den sie für die Verifizierung der digitalen Signatur benötigen, vertrauen. Identität B muss sich dabei aber sicher sein, dass der Schlüssel auch wirklich zu Identität A gehört. Dies stellt er fest indem er die Identität der CA, die das Zertifikat signiert hat, überprüft, und durch verifizieren der Identität und der Integrität des Zertifikats, und zwar durch die von der CA angehängten Signatur. X.509 Zertifikate beinhalten außerdem ein Validitätsdatum um vor kompromittierenden (oder abgelaufenen bzw. ungültigen) Schlüsseln zu schützen. Abb. 4.1 zeigt ein Beispielzertifikat unter Windows.

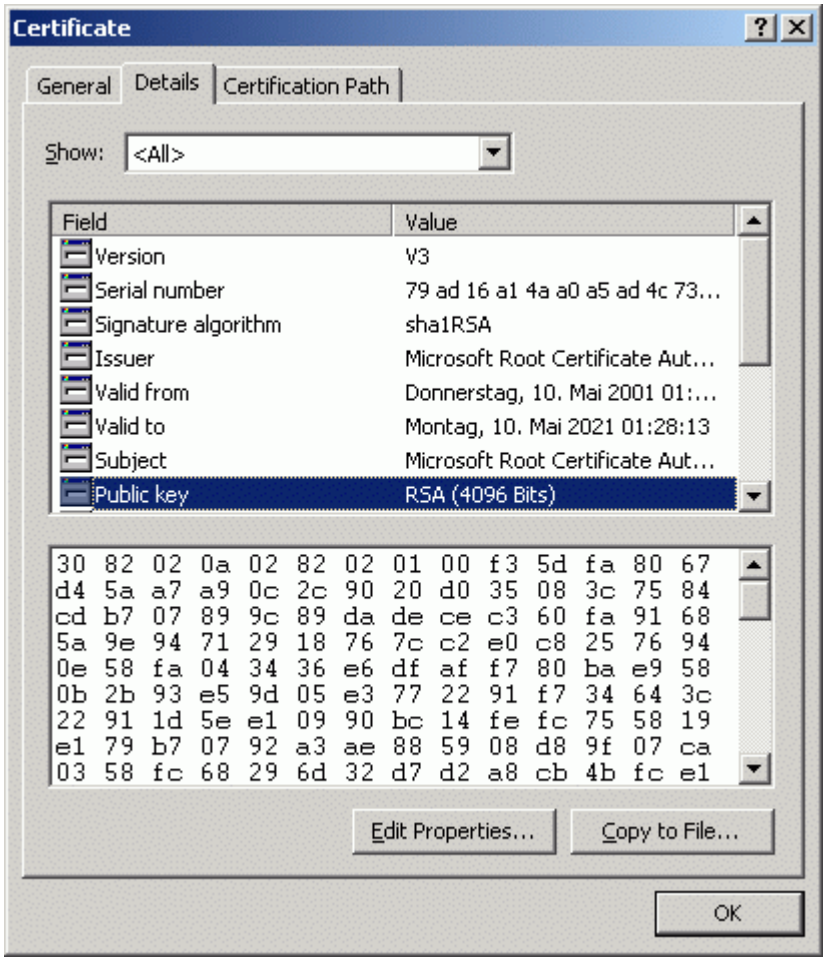



**Abb. 4.1:** X.509 Zertifikat

# Kapitel 5:
# Sicherheit für Web Services

Ein Problem des neuen Web Services Modells ist, dass es auch neue Sicherheitsanforderungen adressiert. Es reicht meist nicht aus, die Applikationen hinter Firewalls zu verstecken, da die Sicherheit bereits auf Ebene der Applikationen stattfinden muss, da nicht immer gewährleistet werden, dass ausschließlich über sichere Netzwerke kommuniziert werden, sondern möglicherweise auch über unsichere Intranets oder das öffentlich zugängliche Internet tansportiert werden und inspiziert oder sogar manipuliert werden können.

Im Laufe eines Geschäftsprozesses ist es also durchaus möglich, dass verschiedene Zwischenstationen mit verschiedenen Zugriffsrechten die Daten lesen. Sobald aber nur ein kleiner Teil dieser Kette kompromittiert wird, würde das ganze Vertrauensmodell in sich zusammenfallen. Um derartige Anforderungen umsetzen zu können, haben nahmenhafte und innovative Firmen die Entwicklung der WS-Security Standards vorangetrieben. Der Vorteil ist, dass die meisten Technologien die dort zum Einsatz kommen, bereits erprobt sind und sich schon über einen längeren Zeitraum bewährt haben, wie bspw. das Konzept der digitalen Signaturen oder die kryptographische Verschlüsselung [Rose04].

Es ist dennoch eine große und auch kostspielige Herausforderung, die zahlreichen verschiedenen Sicherheitstechnologien sinnvoll zusammenzuführen. Konzerne wie bspw. die Daimler Chrysler AG, Siemens oder T-Systems definieren IT-Strategien um ihre Sicherheitsanforderungen zu gewährleisten. Abgeleitet von der Technologie werden Blueprints erstellt, die Produkte am Markt plazieren. Neue Technologien wie Web Services werden in den Blueprints jedoch nur dann aufgenommen werden können, wenn sie die internen Sicherheitsstandards erfüllen.

## 5.1 WS-Security

Unter WS-Security versteht man eine Reihe von Spezifikationen, die sich um die Sicherung der Kommunikation zwischen Web Services kümmern. Basis von WS-Security sind die Standards XML-Signature, XML-Encryption und SAML. WS-Security beschreibt mit Hilfe dieser Spezifikationen, welche Maßnahmen erforderlich sind um SOAP Nachrichten sicher zu machen. Ziel von WS-Security ist es nicht, neue Sicherheitskonzepte zu entwerfen, sondern ein einheitliches Format dafür bereitzustellen wie Sicherheitsmaßnahmen in SOAP Nachrichten untergebracht werden können. Ein SOAP Security Header kann folgende Elementen enthalten:

- **XML Signature** und/oder
- **XML Encryption** und/oder
- **Security Token(s)**

WS-Security ist also eine Sicherheitserweiterung für SOAP, die gängige und erprobte Sicherheitskonzepte wie Verschlüsselung und digitale Signaturen, und Security Tokens (die z.B. X.509 Zertifikate, Kerberos Tickets oder SAML Assertions repräsentieren können) im SOAP Nachrichtenkontext implementiert, und unter der Bezeichnung „WS-Security: SOAP Message Security“ geführt wird [WSS06].

### 5.1.1 XML-Signature

XML-Signature oder auch XML Digital Signature ist der vom W3C vorgeschlagene Standard für digitales signieren von XML Dokumenten, Teilstücken eines Dokuments oder eines externen Objekts. Man kann damit fast alles signieren, sofern es möglich ist mit einer URL darauf zu verweisen. Eine XML Signatur ist selbst ein XML Teilstück mit einem entsprechenden, dazugehörigen XML Schema das spezifiziert wie es strukturiert ist. Innerhalb der Signatur befinden sich Referenzen (URIs) auf die zu signierenden Inhalte. Die Verwendung von XML-Signature kann überaus flexibel erfolgen: XML Dokumente können mehrere Signaturen beinhalten, außerdem ist es möglich auch nur Teilstücke eines Dokuments zu signieren, sowie auch externe Ressourcen (z.B. ohne XML Inhalte). Es gibt 3 verschiedene Typen von XML Signaturen [Rose04]:

- **„enveloping“ signature**
  hüllt die zu signierenden Elemente ein

- **„enveloped“ signature**
  taucht innerhalb des zu signierenden Elements auf

- **„detached“ signature**
  zeigt auf ein XML Element oder eine Binärdatei außerhalb der Elementhierarchie der Signatur, z.B. auf ein externes XML Dokument oder eine andere mit einer URI referenzierbare Ressource

Eine XML Signatur, bzw. deren Referenz auf das zu signierende Element, kann zur selben Zeit sowohl enveloping, enveloped und detached sein. Das <Signature> Element kann also auch mehrere <Reference> Elemente enthalten [W3C02a].

### 5.1.2 XML-Encryption

Analog zu XML-Signature ist XML-Encryption der W3C Standard für verschlüsseln von XML Dokumenten, Teilstücken eines Dokuments oder eines externen Objekts. Die Technologien sind überlappend, und teilen sich Konzepte und Strukturen.

Mit XML-Encryption ist es wie bei XML-Signature ebenso möglich, das gesamte XML Dokument oder auch nur Teilstücke davon zu verschlüsseln. Eine XML-Encryption Struktur kann daher auch mehrfach in einem Dokument vorkommen, denselben Schlüssel verwenden oder auch mit verschiedenen Schlüsseln. Dies erlaubt eine hohe Flexibilität, so dass bspw. verschiedene Teile eines XML Dokuments nur von einer bestimmten Personengruppe gelesen werden können, während sie für Andere versteckt sind. Diese Verschleierung von Teilstücken eines Dokuments durch Verschlüsselung hat jedoch Auswirkungen auf andere XML Standards. WS-I versucht dieses Problem zu lösen, indem es bestimmte Vorraussetzungen und Richtlinien für die Verwendung des Verschlüsselungselements vorgibt [W3C02b].

Trotz aller Gemeinsamkeiten mit XML-Signature, hat XML-Encryption den Verwendungsweck, vertrauliche Daten vor Unbefugten zu schützen die keinen Zugriff auf den private key haben. Die beiden Technologien sind jedoch komplementär, da es bei der Verwendung von XML und Web Services häufig vorkommt, dass man eine Nachricht sowohl verschlüsseln als auch signieren muss bevor man sie versendet. Der Unterschied ist jedoch, das XML-Signature auf das zu signierende Element verweist, während ein XML-Encryption Element die Daten die es verschlüsselt direkt beinhaltet [Rose04].

### 5.1.3 XML-Security

XML-Signature und XML-Encryption werden unter dem Begriff XML-Security zusammengefasst. Bei Zusammenwirken beider Technologien ist vor allem von Bedeutung, in welcher Reihenfolge die einzelnen Schritte ausgeführt werden. Auf der Ebene der Nachrichtensicherheit ist es grundsätzlich zu empfehlen, zuerst die Daten zu verschlüsseln um sie anschließend zu signieren, während wenn die Informationen durch Personen gelesen werden können, es Sinn macht die Daten zuerst zu signieren, und sie danach verschlüsseln. Die Idee hinter diesem Prinzip ist, dass man sehen sollte was tatsächlich signiert wurde, also das signierte repräsentativ dafür ist, was die signierende Person gesehen hat - nur Annahmen darüber zu machen könnte u.U. riskant sein. Aufgrunddessen würde das Verschlüsseln vor dem Signieren dieses Prinzip verletzen. Auf Nachrichtenebene attestiert jedoch der Server selbst die Integrität der Nachrichtenbits, und dieses Konzept wird durch Signieren der verschlüsselten Daten nicht verletzt. Eine übliche Strategie bei der Verwendung von XML-Signature in Verbindung mit XML-Encryption ist, zuerst die Daten zu verschlüsseln, und danach erst zu signieren [Rose04].

Sowohl XML-Signature als auch XML-Encryption werden aufgrund ihrer Komplexität auf kurz oder lang zu einem eingebetteten Teil der Infrastruktur, anstelle einer Technologie mit der es Entwickler tagtäglich zu tun bekommen. IBM, BEA, Microsoft und andere Hersteller von Entwicklungssoftware erleichtern den Umgang mit diesen Technologien, indem sie diese in ihre Entwicklungsumgebungen integrieren und dadurch einfachere und schnellere Implementierungen ermöglichen [Rose04].

### 5.1.4 SAML

Es gab lange Zeit keinen Standard, der geregelt hat wie Sicherheitsattribute über Unternehmensgrenzen hinweg kommuniziert werden können. Mit der Security Assertion Markup Language (SAML), deren Idee noch vor den Web Services geboren wurde, gibt es inzwischen einen Standard der genau dafür geeignet scheint, die Forderungen nach einem flexibleren, systemübergreifenden Sicherheitsmodell zu realisieren. SAML ist eine ebenfalls XML-basierte Auszeichnungssprache für die Repräsentation von Sicherheitsattributen, und ist inzwischen fester Bestandteil des WS-Security Pakets [Rose04].

Die Idee von SAML ist, dass Informationen über eine Identität von einer Domäne in eine andere übertragen werden können. Eine Vielzahl der Fragen dreht sich dabei um Identität. Wer hat Zugriff auf mein Netzwerk oder auf meine Informationen? Von wem kommt diese Anfrage? Wer bürgt dafür, dass diese Informationen korrekt sind? Wer hat mir diese vertraulichen Daten geschickt? Woher weiß ich, dass es tatsächlich der Absender ist?

Identität bezieht sich auf ein Individuum bzw. eine Entität, wie z.B. eine Maschine, die ihre Identität sich selbst oder einer Organisation gegenüber präsentiert. Eine solche Entität, die ihre Identität anderen gegenüber behauptet, nennt man auch Subjekt. Die Identität herzustellen ist eine kritische Vorbedingung bei der Bestimmung, welche Rechte diese Identität nachher überhaupt erhalten soll. Die Identität eines Subjekts wird zu Beginn durch einen unabhängigen Dritten, einer „trusted third party" eingerichtet und verifiziert, was dazu führt dass credentials erzeugt werden, die dann gegenüber einer anderen Vertrauensdomäne stellvertretend für die Identität des Subjekts stehen. Diese Identität wird dann als „portierbar" bezeichnet.

Credentials werden in Form von „assertions" (Behauptungen) dargestellt. Eine assertion kann herausgefordert und überprüft werden bevor ihr geglaubt wird. Authentifizierung ist z.B. eine Behauptung, dass ein Subjekt jenes ist, das es vorgibt zu sein, und es verifiziert werden kann, dass die vorgezeigten credentials auch tatsächlich zu diesem Subjekt gehören. Autorisierung dagegen ist eine Behauptung, dass die Identität des Subjekts dazu befugt ist bestimmte Aktionen durchzuführen. Viele bisherige Ansätze basieren auf einer zentralen CA, die Zertifikate vergibt um damit eine sichere Kommunikation zwischen den Teilnehmern zu garantieren. SAML jedoch drückt die Sicherheit in Form von Behauptungen über Subjekte aus.

SOAP ist das Bindeglied zwischen Web Services. Mit SOAP lässt sich über die Unternehmensgrenzen hinweg kommunizieren, jedoch hat SOAP keine standardisierte und interoperable Funktion um die Sicherheitseigenschaften verschiedener Entitäten mit verschiedenen Vertrauensmodellen zu kommunizieren. Da in diesem Fall die SOAP Nachrichten von einer Vertrauensdomäne in eine andere geschickt werden, müssen die entsprechenden credentials des Subjekts, das den entfernten Service anfordert, mit der Nachricht mitgeschickt werden. Auf diese Art und Weise kann die angefragte Vertrauensdomäne ihr Vertrauensmodell auf die empfangene Nachricht anwenden. SAML ist der Standard um portable Identitäten einzurichten und assertions zu beschreiben, die von diesen Identitäten angefertigt werden. SAML hat den Vorteil, da es in XML beschrieben wird und die gängigen XML Werkzeuge darauf angewendet werden können, und es nicht auf eine bestimmte Transportart festgelegt ist, da SAML die Regeln wie es transportiert wird selbst spezifiziert. Drei fundamentale, XML-basierte Mechanismen (bzw. Schemata) bilden das Grundgerüst von SAML [Rose04]:

- **assertions:** die credentials eines Subjekts
- **request/response Protokoll:** für die Verarbeitung von assertions
- **bindings:** für die Kommunikation der assertions über standardisierte Transport- und Messaging-Frameworks wie z.B. SOAP over HTTP

SAML definiert drei verschiedene Typen von assertions:

- **authentification assertions**
  spezifizieren, dass eine bestimmte Autorität das Subjekt authentifiziert hat

- **authorization assertions**
  spezifizieren, dass eine bestimmte Autorität den Zugriff eines Subjekts auf bestimmte Ressourcen freigegeben oder abgelehnt hat

- **attribute assertions**
  stellen qualifizierte Informationen über die Authentifizierungs- oder Autorisierungsassertions bereit

Assertions können nun als Security Tokens im Security Header einer SOAP Nachricht in andere Vertrauensdomänen weitergereicht werden. Damit könnten dann z.B. SSO (Single Sign On), verteilte Transaktionen oder Autorisierungsdienste realisiert werden. Security Tokens gibt es in verschiedenen Ausführungen, die im folgenden Abschnitt vorgestellt werden.

## 5.2 Security Tokens

Security Tokens sind Sicherheitsinformationen die im SOAP Security Header untergebracht werden können, und die dort typischerweise für Authentifizierungs- bzw. Autorisierungszwecke verwendet werden. Der WS-Security Standard definiert verschiedene Tokentypen, und das flexible und erweiterbare Modell von WS-Security sieht vor, dass ohne grossen Aufwand auch noch weitere Tokentypen spezifiziert werden können. Die Token fallen in drei Kategorien, die sich durch ihre wrapper[1] unterscheiden („wsse“ steht für WS-Security Extension):

- **Username Token** (username/password)
  wrapper: <wsse:UsernameToken>

- **BinarySecurity Token** (z.B. X.509 Zertifikate, Kerberos tickets)
  wrapper: <wsse:BinarySecurityToken>

- **XML Token** (z.B. SAML assertions)
  wrapper : z.B. <wsse:assertion>

1. XML-Konstrukte, die bestimmte Informationen innerhalb eines definierten Elementtyps kapseln

SOAP Message Security v1.1 spezifiziert gegenüber der Vorgängerversion mit den Tokenprofilen für Username Token und X.509 Zertifikaten außerdem noch das SAML Tokenprofil und das Kerberos Tokenprofil.

### 5.2.1 Username Token

Kein Sicherheitsmechanismus wäre komplett ohne die Username/Passwort Option, auch wenn die Sicherheitsrisiken durch bspw. zu einfache oder zu komplexe Passwörter bekannt sind. Sicherheit wird oft anhand der Kostenfrage fest gemacht, und manchmal ist dies auch die richtige Strategie, denn das Username/Passwort Prinzip lässt sich im Vergleich zu anderen Sicherheitsstrategien relativ einfach umsetzen. Um Username und Passwort sicher in einer SOAP Nachricht unterzubringen, bedarf es jedoch noch zusätzlicher Maßnahmen wie z.B. XML-Encryption oder mindestens SSL, da das Passwort im SOAP Security Header im Klartext vorliegt und sonst einfach ausgelesen werden könnte [Rose04]. Bsp. 5.1 zeigt den Ausschnitt eines Username Tokens.

```
<S:Envelope>
   <S:Header>
         ...
         <wsse:Security>
            <wsse:UsernameToken>
               <wsse:Username>username</wsse:Username>
               <wsse:Password>password</wsse:Password>
            </wsse:UsernameToken>
```

**Codebsp. 5.1:** Username Token

Alternativ zum Klartext-Passwort stellt WS-Security noch einen weitere Möglichkeit zur Verfügung, das Username Token mit „password digest“[1]. Es ist eine echte Alternative, wenn beide Kommunikationspartner das Passwort bereits als Klartext vorliegen haben. Dieser Prozess involviert einen passwort digest Algorithmus auf der Seite des Clients, der aus einem Zeitstempel, einem Nonce (zufällig generierte Anzahl von Bytes) und dem Passwort einen Hashwert bildet. Der Zeitstempel und der Nonce werden dann im Username Token mitgeschickt, damit der Server den Prozess der Digest-Generierung wiederholen kann um eine erfolgreiche Authentifizierung per Username Token zu gewährleisten [Rose04].

Selbst in besonders sicherheitssensitiven Organisationen und Unternehmen gehören Passwörter heutzutage immer noch zu den meistverwendeten Authentifizierungsmechanismen.

1. Das Passwort wird durch eine Einwegfunktion unlesbar gemacht

### 5.2.2 BinarySecurity Token

Ein BinarySecurity Token kann verschiedene Klassen von binären credentials enthalten, z.B. ein X.509 Zertifikat oder ein Kerberos ticket. Da solche Tokens im Binärformat sind, muss ein entsprechender Kodierungstyp spezifiziert werden um das Token im XML Format repräsentieren zu können (z.B. „Base64Binary“).

**• X.509 Token**

Ein X.509 Zertifikat ist ein digitaler Container für den public key Teil eines asymmetrischen (public/private) Schlüsselpaars. Eine CA, bürgt mit einer Signatur dieses Containers dafür, dass der public key dieser bestimmten Identität zugehörig ist. Ein X.509 Zertifikat ist eine frei zugängliche Information die beliebig verbreitet werden kann. Es kann trotz dieses öffentlichen Status' dennoch als Authentifizierungsmittel benutzt werden. Allein der Zugriff auf ein X.509 Zertifikat authentifiziert die Identität des Senders einer SOAP Nachricht nicht. Es wird etwas benötigt, was allgemein als „proof of possession“ bezeichnet wird. Im Fall eines X.509 Zertifikats bedeutet proof of possession eine digitale Signatur, bzw. im speziellen Fall von WS-Security eine XML-Signature. Sollte der Web Service Anbieter also ein X.509 BinarySecurity Token als Authentifizierungsmechanismus verlangen, muss der Konsument durch eine an die Nachricht angehängte digitale Signatur verifizieren, dass er Zugriff auf den korrespondierenden private key hat. Kann der Empfänger die Signatur erfolgreich verifizieren, und vertraut er der Zertifizierungsstelle welche die Identität, (die mit dem private key verknüpft ist mit dem das Zertifikat signiert wurde), verifiziert hat, dann ist ein X.509 Zertifikat ein starker Authentifizierungsmechanismus [Rose04]. Bsp. 5.2 zeigt den Ausschnitt eines X.509 Tokens.

```
<S:Envelope>
   <S:Header>
         ...
         <wsse:Security>
            <wsse:BinarySecurityToken Id="myX509Token"
            ValueType="wsse:X509v3"
            EncodingType="wsse:Base64Binary">
            NIFEPzCCA9CrAwiBGAgIQEmtJZcO …
            </wsse:BinarySecurityToken>
```

**Codebsp. 5.2:** X.509 Token

**• Kerberos Token**

Wie in Kapitel 4.5 bereits erörtert wurde, ist Kerberos ein Authentifizierungsprotokoll das mit geheimen Schlüsseln arbeitet und einen zentralisierten KDC verwendet. WS-Security ermöglicht die Verwendung von Kerberos tickets innheralb eines BinarySecurity Tokens. Es gibt zwei Ticketarten in Kerberos, die von zwei verschiedenen „ValueTypes“ innerhalb des Security Tokens repräsentiert werden können: ein Ticket Granting Ticket (TGT) und ein Service Ticket (ST). Ein TGT ist unabhängig von einem bestimmten Service und wird mehr für Single-Sign-On (SSO) Zwecke benutzt, während ein ST spezifisch an einen bestimmten Service gebunden ist [Rose04]. Bsp. 5.3 zeigt einen Ausschnitt eines Kerberos Tokens.

```
<S:Envelope>
   <S:Header>
         ...
         <wsse:Security>
            <wsse:BinarySecurityToken Id="myKerberosToken"
            ValueType="wsse:Kerberosv5TGT"
            EncodingType="wsse:Base64Binary">
            MIIEZzCCA9CgAwIBAgIQEmtJZcO…
            </wsse:BinarySecurityToken>
```

**Codebsp. 5.3:** Kerberos Token

### 5.2.3 XML Token

Charakteristisch für XML Tokens ist, dass sie nicht unter einem einheitlichen wrapper-Element zusammengefasst werden wie die BinarySecurity Tokens. Bei XML Tokens hat jeder Tokentyp sein eigenes wrapping-Element. Als Beispiel zur Veranschaulichung dieses Konzepts dient das SAML Token.

**• SAML Token**

SAML dient zur Repräsentation von Identitäten, Attributen und Authentifizierungsinformationen. Im WS-Security Standard können die SAML assertion Elemente innerhalb des Security Headers einer SOAP Nachricht empfangen werden. SAML Assertions haben jedoch ein ähnliches Problem wie X.509 Zertifikate. Der Kommunikationspartner, der sich auf die Informationen des Tokens verlässt, muss in der Lage dazu sein den Sender der Nachricht zu bestimmen, oder alternativ einer Instanz vertrauen die ein Beweis für die Identität des Senders hat, und die für diese Identität bürgt.

Aus diesem Grund muss der Empfänger anhand einer von zwei Methoden („holder-of-key“ oder „sender-vouches“) das Subjekt der assertion bestätigen [Rose04]. Bsp. 5.4 zeigt einen Ausschnitt eines SAML Tokens.

```
<S:Envelope>
   <S:Header>
         ...
         <saml:Assertion
              AssertionID="myAssertion">
         …
         </saml:Assertion>
```

**Codebsp. 5.4:** SAML Token

## 5.3 Security Layer

Um die Kommunikation zwischen zwei Parteien eines verteilten Systems zu sichern gibt es zwei Ansätze, die auf verschiedenen Ebenen der ISO/OSI-Schicht verankert sind: Transportsicherheit (point-to-point) und Nachrichtensicherheit (end-to-end).

### 5.3.1 point-to-point Security

Den Transfer zwischen zwei Kommunikationsendpunkten über ein Netzwerk bzw. das Internet zu sichern ist eine Aufgabe, die SSL schon seit geraumer Zeit zuverlässig meistert, indem sie einen sicheren Kanal zwischen zwei Servern einrichtet, über den dann die Nachrichten kommuniziert werden können. Jedoch operiert das SSL Protokoll auf der Transportschicht, d.h. dass beim Empfang einer verschlüsselten und über SSL verschickten Nachricht diese auch bereits auf der Transportschicht verarbeitet wird. Die Sicherheit der Daten ist ab diesem Zeitpunkt nicht mehr gewährleistet (Abb. 5.1).

Web Services mit SSL zu sichern ist eine einfache und effektive Methode, doch sind die Möglichkeiten und die Erweiterbarkeit von SSL stark eingeschränkt, sofern das Modell ausschließlich auf Sicherheit der Transportebene gestützt wird. SSL garantiert zwar point-to-point Sicherheit, aber in der offenen WWW-Infrastruktur wird immer häufiger Sicherheit zwischen Endknoten gefordert, da IT-Systeme immer öfter auch mit dazwischenliegenden Knoten arbeiten um z.B. monitoring, auditing, content-based routing oder Orchestrierung durchzuführen. Problematisch, da bei SSL die Nachricht im Klartext vorliegt, sobald diese den SSL Endpunkt erreicht hat. Ein weiterer Nachteil ist, dass es dabei nicht möglich ist, Teilstücke einer Nachricht zu verschlüsseln, und auch keine Möglichkeit vorgesehen ist, verschiedene Sicherheitsstufen für eingehende bzw. ausgehende Nachrichten zu definieren.


**Abb. 5.1:** point-to-point Security

SSL kommt hier an seine Grenzen, da es keine nahtlose Sicherheit über diese Zwischenschritte gewährleisten kann. Außerdem stellt es eine großes Herausforderung dar, ein Security Framework zu implementieren, welches die SSL Transportlayer-Authentifikation an die Authentifizierungs- und Autorisierungssysteme der Back-End Applikationen angleichen müsste. SSL ist dennoch eine wichtige Stütze für die Sicherheit in serviceorientierten Infrastrukturen, da es zuverlässig und unkompliziert funktioniert, sich leicht implementieren lässt, und durch die meisten Clients und Server unterstützt wird, und häufig als stabile Basis für zusätzliche Sicherheitsmaßnahmen verwendet wird [Rose04].

### 5.3.2 end-to-end Security

WS-Security hat zum Ziel SOAP Nachrichten zu sichern, egal wohin sich diese bewegen und egal wie lange deren Lebensdauer beträgt. Transportbasierte Technologien wie SSL kümmern sich ausschließlich um die Sicherheit während des Transports, ermöglichen jedoch kaum bzw. keine Kontrolle über die Sicherheit der Nachricht auf Applikationsebene, genau dort, wo den Nachrichten eine Bedeutung zukommt. Die Sicherheit wird also direkt an die zu verschickenden Daten gebunden, und nicht, wie bei der transportgebundenen Sicherheit, auf datenunabhängige Byteströme angewendet. Aus diesem Grund wird end-to-end Sicherheit auch oft als informationsbezogene Sicherheit bzw. Nachrichtensicherheit bezeichnet (Abb 5.2).

Nachrichten bleiben dabei unabhängig von der Anzahl der „hops“ (Zwischenschritte) vom Sender zum Empfänger intakt. Gemeinsame Services stellen mitunter ein Szenario bei dem multiple hops vorkommen, da die Nachrichten möglicherweise zwischen mehrfachen Serviceendpunkten transportiert werden müssen um zu ihrem Bestimmungsort zu gelangen. XML-Security stellt die notwendigen technischen Mittel zur Verfügung um Sicherheit auch über hops zu gewährleisten. Die so geschützten Nachrichten (z.B. ein XML Dokument mit einer Kreditkartennummer) sind dann auch noch nach dem Transport geschützt, wenn diese bspw. längerfristig abgespeichert werden müssen. Die Nachricht ist also selbstschützend, und die Sicherheit kann auch nur auf bestimmte Teile der Nachricht angewendet werden, z.B. um manche Passagen nur für bestimmte Nutzergruppen lesbar zu machen.


**Abb. 5.2:** end-to-end Security

Jede Nachricht kann eine eigene Sicherheitsstrategie verfolgen. Im Fall eines synchronen Web Service sind es z.B. zwei Nachrichten: eine für die Anfrage an den Service, und eine weitere für die Antwort die der Service generiert. Dabei ist es möglich, und durchaus auch legitim, dass die Anfrage zwar verschlüsselt wird, die Antwort jedoch nicht. Mit transportbasierter Sicherheit wie bei SSL, wäre diese feingranulare Abstimmung und Flexibilität nicht möglich. End-to-end Security hat im Vergleich zur point-to-point Security jedoch den grossen Nachteil, dass es schwieriger zu implementieren ist. Dies liegt vor allem auch an der Komplexität der zugrundeliegenden Technologien XML-Signature und XML-Encryption.

Dennoch kommt der Sicherheit auf Nachrichtenebene in der Praxis eine wichtige Bedeutung zu, vor allem als Zweitmittel zur Unterstützung und Verstärkung der Sicherheitsmaßnahmen die bereits auf Transportebene stattfinden, die aber nicht immer ausreichend sind um die gewünschte Sicherheit über den reinen Transport der Daten hinaus zu gewährleisten. Eine Kombination wäre aber auch dahingehend möglich, wenn man sich für eine Authentifizierung durch Benutzername/Passwort entscheidet. Diese kann bei WS-Security als Username Token im Rahmen eines Security Tokens im SOAP Header implementiert werden. Das Passwort kann innerhalb der Nachricht unverschlüsselt bleiben (ähnlich wie bei der grundlegenden HTTP Authentifizierung für Transportsicherheit). Mit SSL könnte man die Vertraulichkeit der Daten dennoch sicherstellen, da diese nun zumindest auf dem direkten Transportweg von der Quelle zum Ziel gesichert wäre [Rose04].

## 5.4 XML Schema Validierung

Bei Web Services und den meisten Geschäftssituationen werden die zu verarbeitenden XML Dokumente durch ein korrespondierendes XML Schema verwaltet (z.B. können Entwicklungswerkzeuge das XML Schema in das entsprechende WSDL Dokument automatisch integrieren). Aufgrunddessen kann die Überprüfung eines XML Schemas eine starke Sicherheitsmaßnahme darstellen, da SOAP Nachrichten, die durch eine fehlerhafte Hierarchie, defekte Tags, potentiell gefährliche Inhalte oder andere Ungereimtheiten in ihrer Struktur auffallen, bereits im Vorfeld der Weiterverarbeitung gefiltert werden, da die strengen Vorgaben des XML Schemas Genauigkeit und Konsistenz erzwingt, was vor allem bei den komplexeren XML-Security Konstrukten wichtig ist [Rose04].

## 5.5 WS-*

Die unter dem Begriff WS-* zusammengefassten Standards sind ebenfalls Teil des Web Services Security Frameworks, auf welches WS-Security aufsetzt (Abb. 5.3). Darunter fallen größtenteils jedoch noch unfertige Spezifikationen [Rose04].

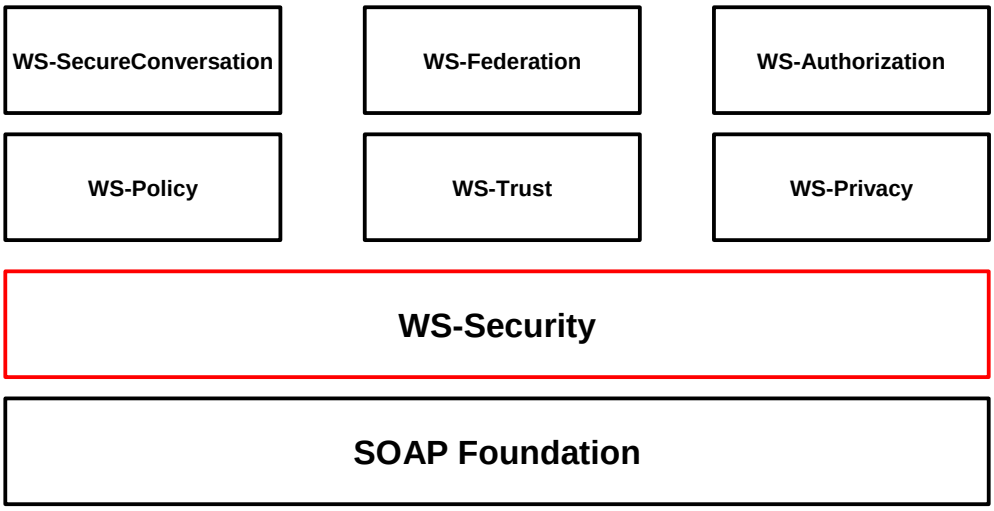



**Abb. 5.3:** Der WS-Security Stack

• **WS-SecureConversation** ermöglicht auf Nachrichtenebene das, was SSL auf Transportebene leistet - einen Sicherheitskontext herzustellen, der den sicheren Austausch von Nachrichten gewährleistet.

• **WS-Federation** unterstützt die Errichtung von Vertrauensverhältnissen bzw. einheitlichen, organisationsübergreifenden Sicherheitsrichtlinien um z.B. Kerberos- und X.509-Systeme zusammenführen zu können.

• **WS-Policy** spezifiziert die Möglichkeiten, wie Web Services ihre Sicherheitsrichtlinien beschreiben und kommunizieren können. Untergeordnete Teilmengen davon sind WS-PolicyAssertions und WS-PolicyAttachment.

• **WS-Trust** ermöglicht die Errichtung von übergeordneten Vertrauensverhältnissen für den sicheren Austausch von Nachrichten.

• **WS-Privacy** wird innerhalb von P3P (Platform for Privacy Preferences) implementiert, und dient zur Kommunikation privater Richtlinien, die der Sender einer SOAP Nachricht erfüllen muss um einen Web Service aufzurufen.

## 5.6 Weitere Sicherheitsmodelle

Im nahen Umfeld der WS-* Familie für Sicherheitsspezifikationen gibt es weitere Technologien, die für den sicheren Einsatz von Web Services entwickelt wurden. Hierunter fallen die Konzepte XKMS und XACML.

### 5.6.1 XKMS

XKMS steht für XML Key Management Specification und komplementiert die XML-Security Standards XML-Signature und XML-Encryption. Sobald Web Services diese beiden Technologien einsetzen um eine end-to-end Sicherheit ihrer Nachrichtenkommunikation zu erreichen, ist es von großem Vorteil wenn sie sich der public key Kryptographie bedienen. Diese setzt jedoch eine unterstützende Infrastruktur vorraus, welche die Verteilung, Zertifizierung und das Lebenszyklus-Management der dabei verwendeten Schlüssel reguliert.

PKI hat sich jedoch als teuer, schwierig umsetzbar und komplex in der Wartung herausgestellt. Komplex auch aus dem Grund, da bisher immer jeder Anbieter und Konsument eines Web Service eine eigene PKI aufbauen musste. An diesem Punkt kommt XKMS ins Spiel, mit der Idee die PKI selbst als vertrauenswürdigen Web Service zu implementieren. Dabei werden zwei Protokolle spezifiziert: X-KISS (XML Key Information Service Specification), das die Erstellung eines Service übernimmt, an den die Applikation die Verarbeitung der Schlüsselinformationen deligiert, und X-KRSS (XML Key Registration Service Specification), das die Registrierung und die Verwaltung der Schlüsselpaare regelt. Das Ziel von XKMS ist es also, PKI Unterstützung für XML Applikationen wie Web Services bereitzustellen, bzw. den Anforderungen von XML-Signature und XML-Encryption gerecht zu werden [Rose04]. XMKS 2.0 ist seit Juni 2005 eine W3C Recommendation.

### 5.6.2 XACML

XACML steht für eXtensible Access Control Markup Language und ist ein XML Schema welches die Darstellung und Verarbeitung von Autorisierungs- und Berechtigungsrichtlinien beschreibt. XACML repräsentiert Regeln, die spezifizieren wann, wo, wer, was und wie auf Informationen zugegriffen werden darf - diese Form der Zugriffskontrolle wird oft als Rechteverwaltung bezeichnet.

Die Fülle verschiedener Mechanismen für die Durchsetzung dieser Zugriffskontrolle ist ein wichtiger Grund für die Verwendung von XACML. Damit diese Richtlinien korrekt implementiert werden können ist es normalerweise so, dass die verschiedenen Durchsetzungsmechanismen separat verwaltet werden, was das modifizieren der Sicherheitsrichtlinen aber sehr teuer macht. Das Ziel der XACML

Spezifikation kann jedes Objekt sein, dass durch XML referenziert wird. Dies erlaubt eine sehe feingranulare Kontrolle [Rose04]. XACML 2.0 ist seit Februar 2005 ein OASIS Standard.

## 5.7 XML Firewalls

XML-Firewalls, auch Web Service Firewalls, Web Service Security Gateways oder SOAP-Proxy genannt, sind auf Applikationsebene angesiedelte Gateways für die Verarbeitung von XML bzw. eingehenden SOAP Requests. Ihr Ziel ist es eventuelle Sicherheitsrisiken schon vor dem eigentlichen Eindringen in das System abzuschwächen. Die Hauptaufgabe der Firewall besteht in der Validierung und Filterung jeder SOAP Nachricht, die von einem externen Netzwerk aus das System erreicht. Analog dazu gibt es auch die Möglichkeit eine zweite XML Firewall einzusetzen, die dann alle ausgehenden SOAP Requests in fremde Netzwerke überwacht.

Eine XML Firewall ist für viele Bereiche verantwortlich: sie prüft ob der Inhalt einer SOAP Nachricht einem gegebenen XML Schema entspricht, sie erkennt Direktiven für das Routing von SOAP Nachrichten, sie leitet erfolgreich authentifizierte, autorisierte, inspizierte und validierte Anfragen an den entsprechenden Web Services Provider weiter, sie sorgt für Sicherheit auf der Transportschicht, sie führt Username Token und X.509 Token Authentifizierung durch, sie interpretiert XML Signature und XML Encryption, sie kümmert sich um logging und auditing und schützt durch Prüfung und Nachrichteninhalte vor XML- und SOAP-spezifischen und anderen Angriffen auf der Applikationsebene.

## 5.8 WS-I Basic Security Profile

Ziel der Arbeit der WSI sind die „profiles“, Implementierungsrichtlinien die bestimmte Vorgaben beinhalten, inwiefern zueinandergehörige Web Service Spezifikationen interoperabel sind. Bereits fertiggestellt wurde das Basic Profile, das Attachments Profile und ein einfaches SOAP Binding Profile. Desweiteren stellt WS-I Beispielapplikationen bereit, welche die Richtlinien implementieren und die darin geforderte Interoperabilität demonstrieren. Werkzeuge, um die Nachrichten zwischen den Web Services auf Kompatibilität mit dem WS-I Standard zu prüfen, sind ebenfalls Teil des Pakets.

Von größerem Interesse für die vorliegende Arbeit ist das Basic Security Profile, das aber noch keinen endgültigen Status erreicht hat und bisher nur als Working Group Draft in der Version 1.1 vorliegt. Der Inhalt des Drafts ist eine umfangreiche Dokumentation über eine Ansammlung nicht-proprietärer Web Services Spezifikationen mit diversen Klarstellungen und Erweiterungen von Spezifikationen welche die Interoperabilität fördern. Das Basic Security Profile ist eine Erweiterung des Ba-

sic Profiles, ist dazu konsistent und liefert zusätzliche Funktionalitäten indem es, falls erforderlich, dieses um sicherheitsrelevante Merkmale erweitert.

Interoperabilität kann nicht immer gewährleistet werden, jedoch versuchen die WS-I profiles die häufigsten, in der Praxis auftretenden Probleme zu lösen. Die Interoperabilität, von der in diesem Profil die Rede ist, bezieht sich jedoch nur auf den Web Service Layer, und nicht auf etwaige darunterliegende Layer, wie z.B. der Transport über TCP, HTTP etc. Es wird vorrausgesetzt, dass deren Interoperabilität als verstanden und erprobt gilt. Interoperabilität zwischen neuen Sicherheitstechniken gewährleistet keine zusätzliche Sicherheit, vor allem auch da der Vorgang der Zusammenführung der Technologien und Protokolle eventuell sogar neue Sicherheitsrisiken mit sich bringt. Das Profil versucht diese jedoch explizit zu vermeiden. Finden sich jedoch mehrere Optionen für die Lösung eines Problems, entscheidet sich WS-I gezielt für die Option mit der höheren Interoperabilität, und nimmt damit eventuelle Schwächen in der Sicherheit in Kauf.

Die Vorraussetzungen, welche dem Basic Security Profile zugrunde liegen, sind von aktuellen Arbeiten der Standardisierungskommitees wie OASIS abgeleitet, deren Spezifikationen im Basic Security Profile referenziert werden. Diese Vorraussetzungen geben jeweils die Kriterien für eine Übereinstimmung mit dem Profil vor. Typischerweise beziehen sich diese auf eine existierende Spezifikation und zeigen Verfeinerungen, Erweiterungen, Interpretationen und Klarstellungen zur Verbesserung der Interoperabilität auf. Keine der Vorraussetzungen, unabhängig von ihrem Konformitätsgrad, sollte jedoch als Limitierung der Möglichkeiten bei der Abwehr von Angriffen verstanden werden, sofern die Implementierung ansonsten dem Profil konform ist. Der Spielraum des Basic Security Profile hängt ausschließlich von den Technologien ab, die es referenziert [WSI06b].

# Kapitel 6: Modellentwurf

In diesem Kapitel wird ein Beispielszenario vorgestellt, anhand dessen die theoretischen Kenntnisse veranschaulicht werden, und dessen Realisierung in Kapitel 7 veranschaulicht wird. Der Anfang dieses Kapitels besteht aus einer Beschreibung des Szenarios, der dazugehörigen Services, und der Datenklassifizierung, in deren Abhängigkeit die Sicherheitsinfrastruktur entworfen wird. Ein Lösungsansatz schließt das Kapitel ab.

## 6.1 Die Services

Als Beispiel für eine SOA die mit Web Service Standards implementiert wird, dient ein praxisnahes Modell welches die Kommunikation zwischen einem Servicekonsumenten und drei verschiedenen Web Service Anbietern veranschaulichen soll. Um die Abfolge eines komplexen Geschäftsprozesses zu veranschaulichen, werden ingesamt drei Services modelliert, die die einzelnen Schritte eines Geschäftsprozesses simulieren.

Angenommen es gibt einen Kunden der anhand einer gegebenen Lieferantenadresse die dazugehörigen Sachnummern eines Lieferanten ermitteln möchte, und über die Abfrage eines UDDI Verzeichnisdienstes einen geeigneten Web Service A findet, der diese Aufgabe erfüllt. Der Kunde kann nun anhand der Informationen, die ihm durch das in der UDDI verankerte WSDL Dokument zur Verfügung stehen, diesen Service finden und in Anspruch nehmen, indem er ihm eine SOAP Nachricht mit den geforderten Parametern (in diesem Fall die Adresse des Lieferanten) an den Service schickt. Die Applikation, die sich hinter der Schnittstelle von Service A verbirgt, verteilt die Aufgaben an zwei weitere Applikationen, (z.B. aufgrund von einer Kostenersparnis), welche die gewünschten Funktionalitäten bereits implementiert haben, und die ebenfalls per Web Service Schnittstelle zur Verfügung stellen. Die Abfrage gliedert sich daher in zwei Arbeitsschritte: Service B ermittelt die Lieferanten anhand der Adresse, und Service C ermittelt anhand der Lieferantenummer die dazugehörigen Sachnummern. Service A nennt man aufgrunddessen auch einen zusammengestellten (composed) Web Service, da er andere Services nutzt um einen Service mit erweiterten Funktionalitäten anzubieten.

Aus den Ergebnissen der Aufrufe von Service B und Service C bildet Service A schließlich das Ergebnis, welches der Kunde angefordert hat, und durch Service A erwartungsgemäß erhält. Die drei Services definieren sich durch folgende Ein- und Ausgabeparameter:

• **Service A**
- Eingabe: Adresse
- Ausgabe: Lieferantennummer, Sachnummern

• **Service B**
- Eingabe: Adresse
- Ausgabe: Lieferantennummer

• **Service C**
- Eingabe: Lieferantennummer
- Ausgabe: Sachnummern

Die Kommunikation zwischen dem Klienten und dem Web Service, und die Kommunikation zwischen den Web Services selbst, basiert auf einem RPC Request-Response Mechanismus, dessen Abfolge in Abb. 6.1 grafisch dargestellt ist:

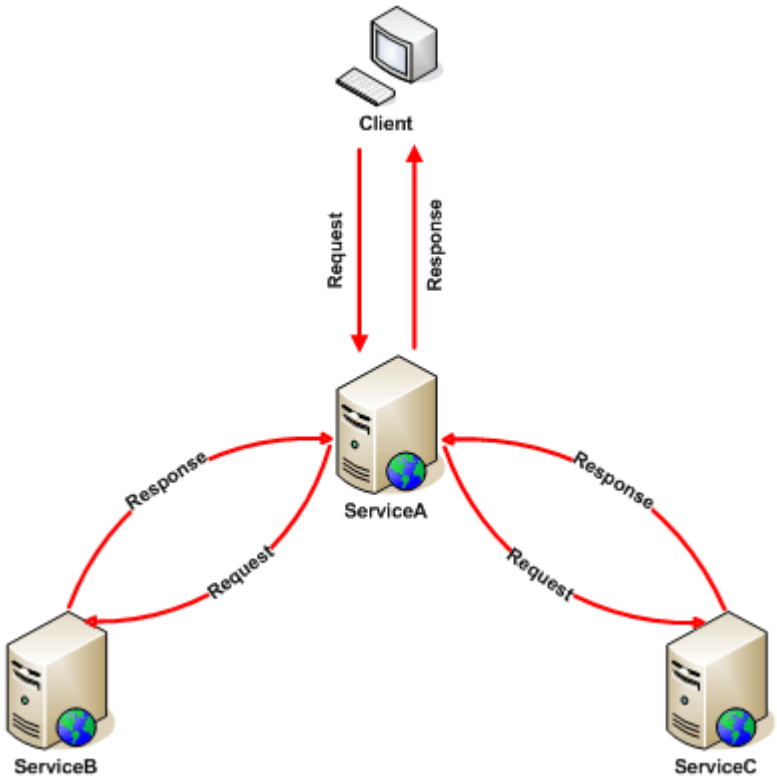



**Abb. 6.1:** Web Services Kommunikationsszenario

## 6.2 Datenklassifizierung

Die zentrale Frage bei der Klassifizierung von Daten bezüglich ihrer Bedeutung für das Unternehmen ist: Was sind die Konsequenzen wenn die Daten manipuliert werden oder in falsche Hände gelangen? Die Datenklassifizierung erfolgt meist in einem abgestuften Schema, welches die Risiken bei einer eventuellem Missbrauch und Manipulation spezifiziert, bzw. die verschiedenen Sicherheitsabstufungen eines Unternehmens repräsentiert. Ein solche Aufteilung könnte in „normal“, „hoch“, „sehr hoch“ oder. „öffentlich“, „intern“ und „vertraulich“ erfolgen [SAP07].

### 6.2.1 öffentlich

- Verstöße gegen Gesetze, Verträge oder andere Regelwerke würden keine schwerwiegende Konsequenzen und lediglich geringe Strafen nach sich ziehen
- Ein Datenmissbrauch hätte minimale soziale bzw. finanzielle Auswirkungen auf die betroffenen Personen
- Die Auswirkungen eines sicherheitsrelevanten Vorfalls auf Ihre Arbeit werden von den betroffenen Personen als gering eingeschätzt
- Bei einem Sicherheitsproblem ist nur eine geringfügige rufschädigende Wirkung für das Unternehmen zu erwarten
- Die finanziellen Verluste für das Unternehmen im Fall eines Sicherheitsverstoßes wären gering

### 6.2.2 intern

- Verstöße gegen Gesetze, Verträge oder andere Regelwerke würden schwerwiegende Konsequenzen und hohe Strafen nach sich ziehen
- Ein Datenmissbrauch hätte schwerwiegende Folgen für den Schutz von Personendaten
- Die Auswirkungen eines sicherheitsrelevanten Vorfalls auf Ihre Arbeit werden von einigen betroffenen Personen als unzumutbar eingeschätzt
- Bei einem Sicherheitsproblem ist eine rufschädigende Wirkung für das Unternehmen zu erwarten
- Die finanziellen Verluste für das Unternehmen im Fall eines Sicherheitsverstoßes wären hoch, aber nicht existenzbedrohend

### 6.2.3 vertraulich

• Verstöße gegen Gesetze, Verträge oder andere Regelwerke würden existenzbedrohende Konsequenzen nach sich ziehen

• Datenmissbrauch hätte extrem schwerwiegende Folgen für den Schutz von Personendaten

• Ein Sicherheitsvorfall würde den sozialen oder finanziellen Ruin für die betroffenen Personen bedeuten

• Die Auswirkungen eines sicherheitsrelevanten Vorfalls auf ihre Arbeit werden von allen betroffenen Personen als unzumutbar eingeschätzt

• Bei einem Sicherheitsproblem ist eine gravierende, möglicherweise existenzbedrohende Rufschädigung für das Unternehmen zu erwarten

• Die finanziellen Verluste für das Unternehmen im Fall eines Sicherheitsverstoßes wären existenzbedrohend

## 6.3 Sicherheitsmaßnahmen

Auf Basis der Datenklassifizierung werden Entscheidungen für die Sicherheitsmaßnahmen einer Sicherheitsinfrastruktur getroffen, und im folgenden Unterkapitel in Abhängigkeit des Security Layers spezifiziert.

### 6.3.1 Sicherheit auf Transportebene

• **Vertraulichkeit**
Da SOAP das textbasierte Kodierungsschema XML benutzt kann eine Nachricht beim Abfangen leicht entschlüsselt werden. Außerdem transportieren bestimmte Authentifizierungsmethoden (z.B: Username Token) ihre Daten im Klartext, daher ist eine Sicherung dieser Mechanismen auf Transportebene durch SSL/TLS notwendig sobald eine SOAP Nachricht über ein unsicheres Netzwerk geschickt wird.

• **Integrität**
Integrität auf der Transportschicht wird durch SSL/TLS ermöglicht, was die Löschung, Veränderung, Verzögerung oder das wiederholte Senden einer Nachricht verhindert.

**• Client Authentifizierung**
Die Authentifizierung des Clients auf der Transportschicht muss durch das SSL „handshake“ Protokoll vollzogen werden, das durch den SSL3.0/TLS1.0 Standard spezifiziert wird. Der Status eines jeden SSL Client Zertifikats das während eines SSL „handshakes“ empfangen wird muss verifiziert werden um sicherzustellen dass das Zertifikat immer noch Gültigkeit besitzt und von einer vertrauenswürden Autorität herausgegeben wurde.

**• Server Authentifizierung**
Der Server, der eine SSL/TLS geschützte HTTP Verbindung (HTTPS) akzeptiert, muss sich selbst authentifizieren, indem er sein SSL Server Zertifikat an den Klienten schickt, welches dann anhand des handshake Protokolls bearbeitet wird, das durch den SSL3.0/TLS1.0 Standard spezifiziert wird.

### 6.3.2 Sicherheit auf Nachrichtenebene

**• Authentifizierung mit Username Token**
Mit der Authentifizierung durch ein Username Token kann der Urheber einer SOAP Nachricht durch Benutzername und Passwort identifiziert werden. Diese Form der Authentifizierung sollte in Zusammenspiel mit SSL/TLS ablaufen um die Vertraulichkeit der credentials sicherzustellen die dabei transferiert werden.

**• Authentifizierung durch X.509 Token**
Mit der Authentifizierung durch ein X.509 Token kann der Urheber einer SOAP Nachricht anhand des Namens eines signierten X.509 Zertifikats identifiziert werden. Das signierte Zertifikat sollte innerhalb eines BinarySecurity Token Elements im SOAP Security Header verschickt werden. Um die beanspruchte Identität zu validieren, muss die SOAP Nachricht mit einem „proof of possession“ für den dazugehörigen private key versehen werden - der SOAP Body muss also mit dem dazugehörigen private key signiert werden. Außerdem müssen die signierten Daten einen Zeitstempel enthalten (der vom Empfänger geprüft wird), und das BinarySecurity Token Element selbst. Der Status des signierten Zertifikats muss verifiziert werden um sicherzustellen dass das Zertifikat immer noch Gültigkeit besitzt und von einer vertrauenswürden Autorität herausgegeben wurde.

**• Vertraulichkeit**
XML-Encryption ist ein Sicherheitsmechanismus für persistente, end-to-end Vertraulichkeit, die auf individuelle SOAP Nachrichten innerhalb eines Nachrichtenstroms oder sogar auf einzelne Elemente der Nachricht angewendet werden kann. Es wird hauptsächlich dann verwendet, wenn die SOAP Nachrichten mehrere Zwischenknoten passieren müssen oder wenn die Vertraulichkeit auch

bei der Verarbeitung und Speicherung in einer Applikation gewährleistet werden soll. Desweiteren wird XML-Encryption verwendet, falls bestimmte Elemente einer Nachricht verschlüsselt, und andere im Klartext vorliegen sollen.

• **Integrität**
XML-Signature ist ein Sicherheitsmechanismus für persistente, end-to-end Integrität die auf digitalen Signaturen beruht, und die auf individuelle SOAP Nachrichten innerhalb eines Nachrichtenstroms oder sogar auf einzelne Elemente der Nachricht angewendet werden kann. Es wird hauptsächlich dann verwendet, wenn die Integrität der gesamten SOAP Nachricht oder einzelner Elemente auch über die Übertragung hinaus gewährleistet sein soll. Desweiteren wird XML-Signature bei der Authentifizierung der Herkunft einer Nachricht verwendet, oder wenn bestimmte Tokens oder Elemente fest an den SOAP Body oder andere Teile der Nachricht gebunden werden müssen. Wenn eine digitale Signatur validiert wird, muss der Status der signierten X.509 Zertifikats verifiziert werden um sicherzustellen dass das Zertifikat immer noch Gültigkeit besitzt und von einer vertrauenswürden Autorität herausgegeben wurde.

### 6.3.3 Weitere Maßnahmen

Im Idealfall sollten mindestens folgende Ereignisse erfasst (logging) werden damit sie nachvollziehbar sind und man eventuelle Konsequenzen daraus ziehen kann:

- Authentifizierungsversuche und -ergebnisse
- Autorisierungsergebnisse
- Fehlersituationen während Erhalt, Analyse oder Verarbeitung einer Nachricht

Nachprüfbarkeit (auditing) erlaubt Informationen über durchgeführte Web Service Transaktionen in audit trails festzuhalten, die später wiederhergestellt und analysiert werden können. Solche audit trails sollten jedoch selektiv konfigurierbar sein, um den Umfang der dadurch generierten Informationen kontrollieren zu können.

Alle SOAP Nachrichten die von externen Netzwerken empfangen oder dorthin geschickt werden sollten detaillierte Filtermethoden und Validierungsprozesse durchlaufen, welche die Nachrichten nach kritischen oder potentiell gefährlichen Inhalten durchsuchen, und verifizieren dass die Struktur der Nachricht mit der Struktur, die durch das XML Schema vorgegeben ist, konform geht.

## 6.4 Lösungsansatz

Anhand des Modellbeispiels, der Datenklassifizierung und den Sicherheitsrichtlinien werden geeignete Maßnahmen für eine Sicherheitsinfrastruktur getroffen. Das Szenario sieht drei verschiedene Zonen vor, die analog zu den Daten in öffentlich, intern und vertraulich klassifiziert werden (Abb 6.2).

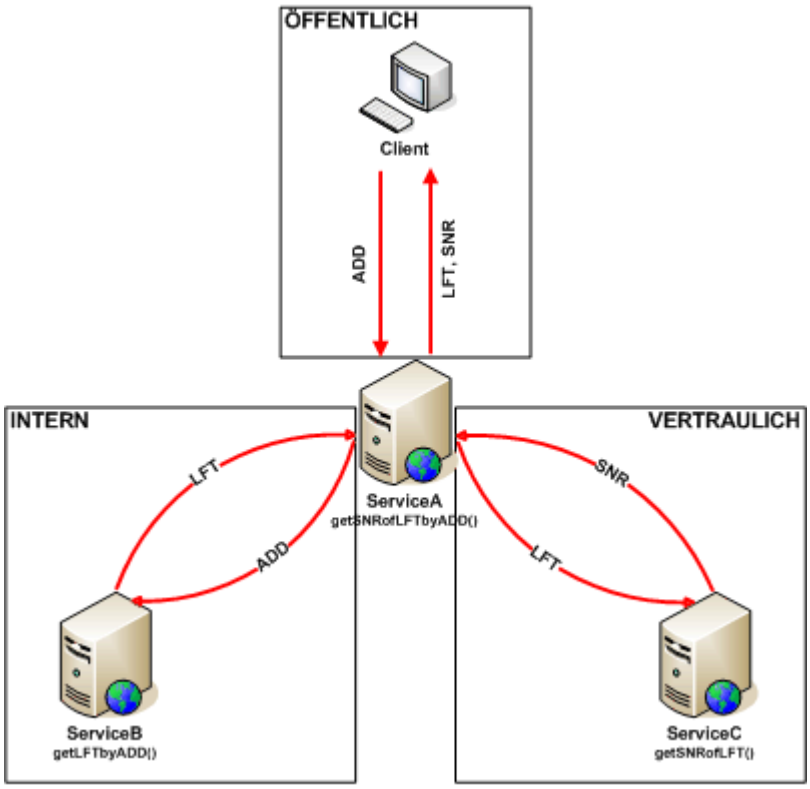



**Abb. 6.2:** Zoneneinteilung für Security

Der Client sendet einen Request an Service A. Inhalt seiner Nachfrage sind die Adressdaten (ADD) eines Lieferanten. Der Client hat keinen Zugriff auf interne Daten, und es werden keine Daten modifiziert. Aufgrunddessen wird die Kommunikation zwischen Client und Service A als öffentlich klassifiziert. Daraus leitet sich folgender Vorschlag für Sicherheitsmaßnahmen ab:

- **SSL/TLS:** für die Vertraulichkeit der Daten
- **SSL/TLS:** für die Integrität der Daten
- **XML Schema Validierung**

Service A leitet die Adressdaten (ADD) an Service B weiter, der als Antwort die Informationen des Lieferanten (LFT) an Service A zurückschickt. Hier findet ein Zugriff auf sensible Kundendaten statt. Aufgrunddessen wird die Kommunikation zwischen Service A und Service B als intern klassifiziert. Daraus leitet sich folgender Vorschlag für Sicherheitsmaßnahmen ab:

- **SSL/TLS:** Authentifizierung durch Username Token
- **SSL/TLS:** für die Vertraulichkeit der Daten
- **SSL/TLS:** für die Integrität der Daten
- **Autorisierung** auf dem Application Server
- **XML-Schema Validierung**
- **Logging:** alle Authentifizierungsversuche
- **XML-Signature (optional):** für „end-to-end“ Integrität der Daten

Service A leitet die Lieferanteninformationen (LFT) an Service C weiter, der als Antwort die Sachnummern (SNR) des Lieferanten an Service A zurückschickt. Hier findet ein Zugriff auf vertrauliche Daten statt. Aufgrunddessen wird die Kommunikation zwischen Service A und Service C als vertraulich klassifiziert. Daraus leitet sich folgender Vorschlag für Sicherheitsmaßnahmen ab:

- **SSL/TLS:** Authentifizierung durch Username Token und X.509 Zertifikat
- **SSL/TLS:** für die Vertraulichkeit der Daten
- **SSL/TLS:** für die Integrität der Daten
- **Autorisierung** auf dem Application Server
- **XML-Schema Validierung**
- **Logging:** alle Authentifizierungsversuche und SOAP Requests
- **XML-Encryption:** für „end-to-end“ Vertraulichkeit der Daten
- **XML-Signature (optional):** für „end-to-end“ Integrität der Daten

Im darauffolgenden Kapitel wird nun geprüft, inwiefern sich dieser Modellentwurf mitsamt den vorgeschlagenen Sicherheitsmaßnahmen mit einer modernen und Web Service fähigen Entwicklungsumgebung wie dem IBM WebSphere Developer realisieren lässt.

# Kapitel 7: Realisierung

Dieses Kapitel befasst sich mit der Realisierung des Beispielszenarios anhand der WebSphere Developer (WSD) Entwicklungsplattform. Eine Beschreibung der J2EE Architektur vervollständigt die Vorstellung der Web Service spezifischen Standards. Das darauffolgende Unterkapitel erläutert die schrittweise Entwicklung eines Web Service, wie diese getestet werden können und welche Client-Typen zur Verfügung stehen. Abschließend folgt eine Vorstellung der Security Architektur des WSD, und wie WS-Security Maßnahmen damit realisiert werden können.

## 7.1 IBM WebSphere Developer

Als Basis der WebSphere Softwareplattform ist der WebSphere Application Server (WAS) eine der prominentesten javabasierten Plattformen für Applikationen der E-Business Welt. Der WebSphere Application Server ist J2EE 1.4 kompatibel und bietet Unterstützung für Web Services an. Im Vergleich zur Vorgängerversion enthält Version 6.0 neue Web Service Technologien wie z.B. WS-AtomicTransaction, WS-Addressing und die Konformität zum WS-I Basic Profile in der Version 1.1. Version 6.1 unterstützt außerdem bereits den Entwurf des WS-I Security Profile 1.0, das in Kapitel 4.9 bevorgestellt wurde. Eine Auflistung der vom WAS 6.0 unterstützen Web Services relevanten Standards zeigt Tabelle 7.1.

| Technology or specification | Version or level supported |
|---|---|
| **Transports** | |
| HTTP/HTTPS | v1.0 and v1.1 |
| JMS | |
| **Messaging** | |
| SOAP specification | v1.1 |
| SOAP Attachements | |
| **Description** | |
| UDDI | v2.0 and v3.0 |
| WSDL | v1.1 |
| WSIL | v1.0 |
| **Security** | |
| WS-Security | OASIS Standard 1.0 |
| **Ineroperability** | |
| WS-I Basic Profile | 1.1.2 |
| WS-I Simple SOAP Binding Profile | 1.0.3 |
| WS-I Attachments Profile | 1.0 |
| **Other Standards** | |
| JAX-RPC | v1.0 for J2EE 1.3, v1.1 for J2EE 1.4 |
| JSR 109 | J2EE 1.3 |
| JSR 921 | J2EE 1.4 |

**Tab. 7.1:** Web Service Standards Unterstützung des WAS 6.0[1]

Das Application Server Toolkit (AST) ist eine Untermenge des IBM Rational Entwicklungsumgebungen, und stellt die Unterstützung für das Erstellen neuer Applikationen für den WAS bereit. Das AST beinhaltet Assistentenprogramme und andere Tools um Web Services zu erstellen. Mit der Version 6.1 erreicht das AST einen wichtigen Fortschritt, da es nun eine eigene J2EE Umgebung, Eclipse 3.1 und Version 1.0 der Eclipse Web Tool Platform (WTP) enthält. Eine Übersicht über die Rational Produktpalette zeigt Abb. 7.1 - die hierarische Gliederung kann so verstanden werden, dass jede vorherige IDE (Integrated Development Environment) in der nachfolgenden Obermenge der IDEs enthalten ist, die wiederum eine erweiterte Funktionalität bieten.

1. Quelle: [WSD06]

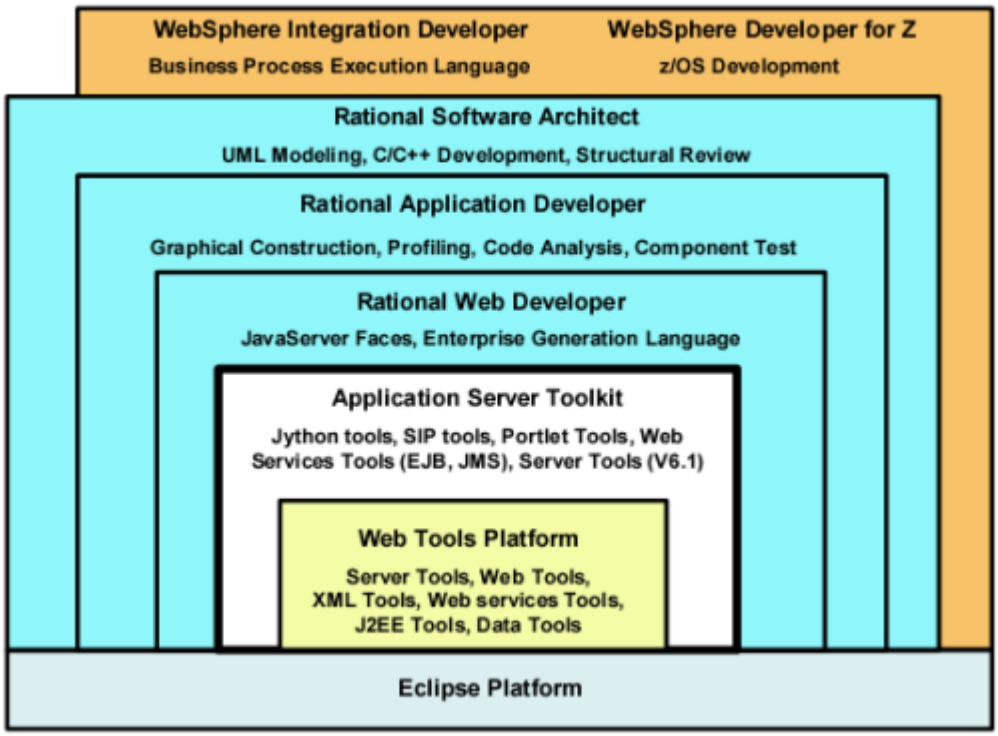



**Abb. 7.1:** IBM Rational Produktpalette[1]

Mit den Werkzeugen des AST können Serviceanbieter und Servicekonsumenten anhand von WSDL Dokumenten, Java Beans oder EJBs (Enterprise Java Beans) realisiert werden. Die Web Service Assistentenprogramme ermöglichen die automatische Generierung von Proxycode für den einfachen Zugriff auf Web Services, sowie einfache Clients um diese testen zu können. Mit den Deployment Descriptors (per XML Schema definierte Konstrukte für die Konfiguration von Web Services) kann WS-Security für die WAS Umgebung konfiguriert werden, und die vollständige WAS Testumgebung erlaubt das Testen der Services. Weitere Funktionalitäten sind das Auffinden und veröffentlichen von Web Services über UDDI, das automatische Erzeugen von Java-Codefragmenten aus WSDL Dokumenten, und die automatische Überprüfung von Überinstimmungen mit WS-I Vorgaben.

1. Quelle: [IBM06a]

## 7.2 J2EE

Die Java 2 Enterprise Edition (J2EE) 1.4 bietet durch das neue JAX-RPC 1.1 API eine umfassende Web Services Unterstützung, da sie auch Serviceendpunkte auf Basis von Servlets und Enterprise Beans unterstützt: JAX-RPC 1.1 stellt Web Services Interoperabilität auf Basis der WSDL und SOAP Protokolle zur Verfügung. Die J2EE 1.4 Plattform unterstützt außerdem die „Web Services für J2EE“ Spezifikation (JSR 921), welche die Anforderungen für die Entwicklung und Errichtung von Web Services definiert, und das JAX-RPC Programmiermodell verwendet. Zusätzlich zu den ganzen Web Service APIs bietet die J2EE 1.4 Plattform außerdem noch Unterstützung für das WS-I Basic Profile 1.0 an, und erleichtert mit den Erweiterungen der Java Servlets und Java Server Pages (JSP) Technologien das Erstellen von Web Front Ends.

Folgende Java Standards und APIs sind für Web Services relevant (JSR steht für Java Specification Request und kennzeichnet eine Anforderung für eine Änderung der Java Language Specification (JLS)):

- **JSR 101: Java APIs for XML-based RPC (JAX-RPC)**[1]
  Ermöglicht die Entwicklung von Web Applikationen und Web Services mit integrierter XML-basierter RPC Funktionalität nach SOAP Spezifikation 1.1.

- **JSR 109: Implementing Enterprise Web Services**[2]
  Definiert Web Services für die J2EE Architektur. Es ist eine Servicearchitektur, welche die J2EE Komponentenarchitektur unterstützt um ein Client-Server Programmiermodell zur Verfügung zu stellen, das hinsichtlich der Applikationsserver portabel und interoperabel ist, und eine skalierbare und sichere Umgebung gewährleisten kann.

- **JSR 31: Java Architecture for XML Data Binding (JAXB)**[3]
  Ermöglicht effizientes, standardisiertes Mapping zwischen XML und Javacode.

- **JSR 67: Java APIs for XML Messaging 1.0 (JAXM)**
  Implementiert das SOAP 1.1 Protokoll und ermöglicht Applikationen das Senden und Empfangen von dokumentbasierten XML Nachrichten.

- **JSR 93: Java API for XML Registries (JAXR)**
  Stellt eine uniforme und standardisierte Java API für den Zugriff auf verschiedene Arten von XML-basierten Verzeichnisdiensten zur Verfügung.

1. Nachfolger ist JSR 224: Java API for XML-based Web Services (JAX-WS) 2.0
2. Nachfolger ist JSR 921: Implementing Enterprise Web Services 1.1
3. Nachfolger ist JSR 222: Java Architecture for XML Data Binding (JAXB) 2.0

## 7.3 Web Service Entwicklung

Der Entwicklungsprozess eines Web Service gestaltet sich ähnlich wie die Entwicklung einer beliebigen anderen Software. Es gibt dabei im Wesentlichen vier Phasen die sich folgendermassen aufgliedern:

**1. Build-Phase**
Beinhaltet die Definition der Funktionalitäten eines Service

**2. Deploy-Phase**
Beinhaltet die Bekanntmachung der Servicedefinitionen per WSDL Dokument und des Web Service Sourcecodes

**3. Run-Phase**
Beinhaltet das Auffinden und Aufrufen des Web Service

**4. Manage-Phase**
Beinhaltet sowohl das Management und die Administration, als auch Wartung und Performancemessung des Service

Abbildung 7.1 veranschaulicht die Zusammenhänge der Entwicklungsschritte in einer grafischen Übersicht:

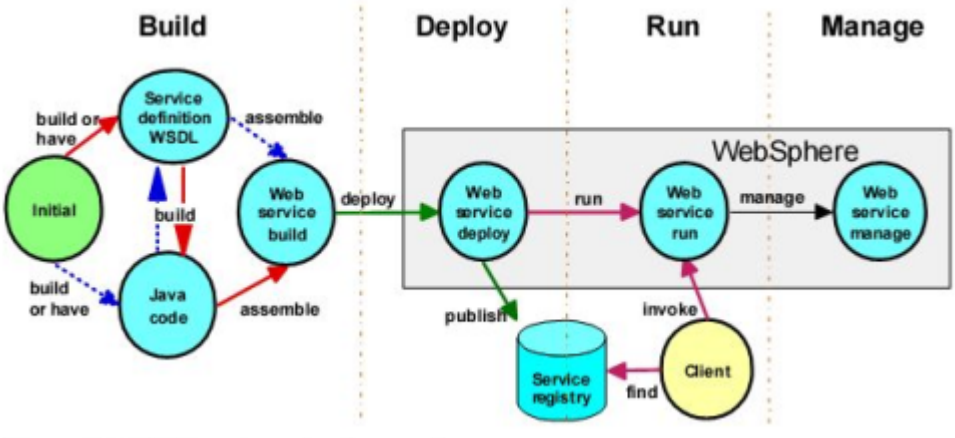



**Abb. 7.2:** Web Service Entwicklung[1]

1. Quelle: [IBM06a]

### 7.3.1 Implementierung

Es gibt drei Möglichkeiten einen Web Service zu entwickeln:

- **„top-down“**
  Anhand der Servicedefinition des WSDL-Dokuments wird mit Zuhilfenahme einer spezifischen Programmiersprache das Gerüst mit den Methoden und Parametern (bei Java z.B. als Java Skeleton Bean) für die neue Applikation erstellt.

- **„bottom-up“**
  Eine bereits existierende Anwendung in einer spezifischen Programmiersprache wird in einen Web Service transferiert. Es entstehen Service „wrappers“ welche die Funktionalitäten des Service anhand des WSDL Dokuments beschreiben und zugänglich machen.

- **multiple Services**
  Die dritte Möglichkeit besteht darin, aus einer neuen Kombination bereits existierender Services neue Funktionalitäten zu generieren.

Das Beispielszenario wurde mit der „top-down“ Methode erstellt, ein Prozess der sich in folgende Teilschritte gliedert:

1. Erstellen des Service Interface (per WSDL Dokument)
2. Generierung des Skelett-Codes (incl. Methoden und Parameter)
3. Implementierung des Web Service (Skelett-Code komplettieren)

Für das Szenario mussten ingesamt drei WSDL Dokumente definiert werden. Die Beschreibung des WSDL Dokuments von Service B dient beispielhaft für alle drei Services (und wurde zur besseren Übersichtlichkeit in Teilstücke zerlegt).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ServiceB" targetNamespace="http://samples"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:impl=http://samples
    xmlns:intf="http://samples
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

**Codebsp. 7.1:** ServiceB.wsdl - Teil 1

```
<wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace=http://samples
    xmlns="http:// www.w3.org/2001/XMLSchema" xmlns:intf="http://samples">
        <element name="getLFTbyADD">
            <complexType>
                <sequence>
                    <element name="PLZ" type="xsd:int"></element>
                    <element name="ORT" type="xsd:string"></element>
                    <element name="STR" type="xsd:string"></element>
                </sequence>
            </complexType>
        </element>
        <element name="getLFTbyADDResponse">
            <complexType>
                <sequence>
                    <element name="LFT" type="xsd:int"></element>
                </sequence>
            </complexType>
            </element>
    </schema>
</wsdl:types>
```

**Codebsp. 7.2:** ServiceB.wsdl - Teil 2

Teil 1 enthält den Namen des WSDL Dokuments und eine Reihe von Namespaces (Bsp. 7.1).[1] Teil 2 spezifiziert anhand eines XML Schemas die Definition der komplexen Datentypen die durch den Service kommuniziert werden (Bsp. 7.2).

```
<wsdl:message name="getLFTbyADDRequest">
    <wsdl:part element="intf:getLFTbyADD" name="parameters"></wsdl:part>
</wsdl:message>
<wsdl:message name="getLFTbyADDResponse">
    <wsdl:part element="intf:getLFTbyADDResponse" name="parameters"></wsdl:part>
</wsdl:message>
<wsdl:portType name="ServiceB">
    <wsdl:operation name="getLFTbyADD">
        <wsdl:input message="intf:getLFTbyADDRequest" name="getLFTbyADDRequest">
        </wsdl:input>
        <wsdl:output message="intf:getLFTbyADDResponse" name="getLFTbyADDResponse">
        </ wsdl:output>
    </wsdl:operation>
</wsdl:portType>
```

**Codebsp. 7.3:** ServiceB.wsdl - Teil 3

Der <message> Tag definiert die zu übertragenden Parameter des Service abstrakt. Der <portType> spezifiziert den „Request/Response“ Nachrichtentyp mit dem der Service seine Nachrichten kommuniziert (Bsp. 7.3).

1. XML Namespaces schützen die definierten Namen eines XML Dokuments

```
        <wsdl:binding name="ServiceBSoapBinding" type="intf:ServiceB">
        <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="getLFTbyADD">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="getLFTbyADDRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="getLFTbyADDResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="ServiceBService">
            <wsdl:port binding="intf:ServiceBSoapBinding" name="ServiceB">
                <wsdlsoap:address location="http://localhost:9080/ServiceB/"/>
            </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

**Codebsp. 7.4:** ServiceB.wsdl - Teil 4

Das <binding> Element spezifiziert das konkrete Protokoll (hier SOAP over HTTP) für den Nachrichtenaustausch. Hier findet auch die Unterscheidung zwischen den Übertragungsmethoden „document/literal“ oder „rpc/encoded“ statt. Das <service> Element wird ein <port> (Service Endpunkt) mittels eines <binding> an die Operationen des Service gebunden, und die Adresse festgelegt, unter der man diesen Endpunkt erreichen kann (Bsp. 7.4).

Anhand der Servicedefinition des WSDL Dokuments erfolgt die automatische Generierung des JavaBean Skelett-Codes durch ein Assistentenprogramm des AST (Bsp. 7.5 und 7.6).

```
public interface ServiceB extends java.rmi.Remote {
    public int getLFTbyADD(int PLZ, java.lang.String ORT, java.lang.String STR)
   throws java.rmi.RemoteException;
}
```

**Codebsp. 7.5:** ServiceB.java

```
public class ServiceBSoapBindingImpl implements samples.ServiceB{
    public int getLFTbyADD(int PLZ, java.lang.String ORT, java.lang.String STR)
    throws java.rmi.RemoteException {
        return -2;
    }
}
```

**Codebsp. 7.6:** ServiceBSoapBindingImpl.java

Neben den beiden generierten Java-Skelett-Klassen, werden außerdem noch der Deployment Descriptor für den Web Service erzeugt, sowie die Adresse des WSDL Dokuments aktualisiert. ServiceB.java ist die Schnittstellen-Klasse, während ServiceBSoapImplementation.java die Skelett-Klasse des Service repräsentiert, deren default-Methode in der Praxis mit der Funktionalität des Service ersetzt werden würde (da in dieser Arbeit jedoch die Kommunikation zwischen den Services, und nicht die Implementierung der Geschäftslogik selbst im Vordergrund steht, reicht der default-Code zur Veranschaulichung aus).

Es gibt verschiedene Möglichkeiten den Web Service zu implementieren:

- Eine neue Implementierung schreiben
- Den default-Code durch eine bereits existierende Implementierung ersetzen
- Einen weiteren Web Service aufrufen

Auf Service A trifft die dritte Möglichkeit zu, da dieser im Beispielszenario die Rolle des Web Service spielt dessen Funktionalität sich aus den Aufrufe von Service B und Service C ergibt (Bsp. 7.7). Um die Methoden von Service B und Service C für Service A zugänglich zu machen, tritt Service A als Client auf, der über die Proxy-Klassen von Service B und Service C auf deren Methoden zugreifen kann. Die Generierung der dafür notwendigen Proxy-Klassen geschieht ebenfalls automatisch durch die Assistentenprogramme.

```
public class ServiceASoapBindingImpl implements samples.ServiceA{
    public int getSNRofLFTbyADD(int PLZ, java.lang.String ORT, java.lang.String STR)
            throws java.rmi.RemoteException {
                System.out.println("Calling ServiceB to retreive LFT ...");
                new ServiceBProxy().getLFTbyADD();
                System.out.println("Calling ServiceC to retreive SNR ...");
                return new ServiceCProxy().getSNRofLFT();
        }
}
```

**Codebsp. 7.7:** ServiceASoapBindingImpl.java

Die Web Service Entwicklungswerkzeuge des AST erlauben eine automatisierte Installation der erzeugten Web Services auf der Web Application Server Testplattform. Gleichermassen bieten sie auch die Möglichkeit an, entsprechende Proxyklassen zum Testen des Web Service zu generieren. Optional besteht außerdem die Möglichkeit der Veröffentlichung der Servicedefinitionen in einem UDDI Verzeichnis, und der Generierung von WSIL Dokumenten anhand von WSDL Dokumenten.

### 7.3.2 Testen

Es gibt verschiedene Möglichkeiten die Funktionalitäten eines Web Service mit dem WSD zu testen:

- Web Services Explorer
- Test Client JSPs
- Universal Test Client

Die einfachste Methode ist mit dem WSD internen Web Services Explorer, der im Gegensatz zu den anderen Methoden keine Proxy-Klassen benötigt, sondern nur anhand des WSDL Dokuments die Funktionalitäten des Service überprüft (Abb. 7.3).

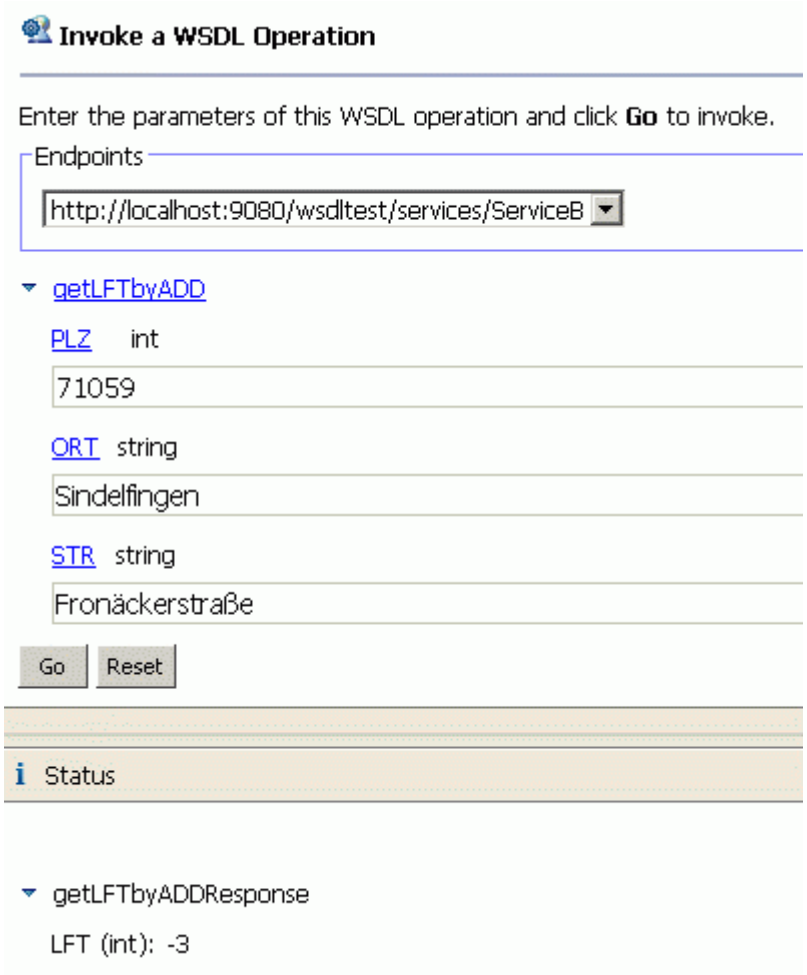



**Abb. 7.3:** Web Services Explorer

Im Quellcode sehen die mit dem Web Services Explorer getesteten Request/Response SOAP Nachrichten wie in Beispiel 7.8 und 7.9 aus.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
    xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:q0=http://samples
    xmlns:xsd="http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <q0:getLFTbyADD>
            <q0:PLZ>71059</q0:PLZ>
            <q0:ORT>Sindelfingen</q0:ORT>
            <q0:STR>Fronäckerstraße</q0:STR>
      </q0:getLFTbyADD>
    </soapenv:Body>
</soapenv:Envelope>
```

**Codebsp. 7.8:** SOAP Request Envelope

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <p113:getLFTbyADDResponse>
            <p113:LFT>-3</p113:LFT>
        </p113:getLFTbyADDResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

**Codebsp. 7.9:** SOAP Response Envelope

### 7.3.3 Clients

WSD unterstützt die Erstellung verschiedener Web Service Clients:

• **Web**
Servlets oder JSPs, oder JavaBeans die durch Servlet/JSP aufgerufen werden

• **EJB**
Session EJBs oder JavaBeans die durch eine Session EJB aufgerufen werden

• **Organisierter Java Client**
ein Java Programm das in einem Client Container läuft

• **Alleinstehender Java Client**
ein Java Programm das außerhalb eines Containers läuft

Abb. 7.4 zeigt ein Beispiel für einen Testclient auf Basis von automatisch generierten JSPs.


**Abb. 7.4:** JSP Test Client

Ein alleinstehender Java Client für Service A könnte folgerndermassen aussehen:

```
public class ServiceAJavaClient {
   public static void main(String [] args) {
       System.out.println("ServiceA Java Client");
        try{
             System.out.println("Getting ServiceA proxy ...");
             ServiceAProxy proxy = new ServiceAProxy();
             proxy.setEndpoint("http://localhost:9080/ServiceA/services/ServiceA");
             System.out.println("Using endpoint: "+proxy.getEndpoint());
             System.out.println("Invoking ServiceA ...");
             System.out.println("Request: getSNRofLFTbyADD(PLZ, ORT, STR)");
             System.out.println("Response: "
                     +proxy.getSNRofLFTbyADD(71059, "Sindelfingen", "Fronäckerstraße"));
             } catch (Exception e) {
             e.printStackTrace();
             }
             System.out.println("End");
       }
}
```

**Codebsp. 7.10:** ServiceAJavaClient.java

Abb. 7.4 zeigt den Konsolen-Output des Clients.

```
ServiceA Java Client
Getting ServiceA proxy ...
Using endpoint: http://localhost:9080/TestServiceA/services/ServiceA
Invoking ServiceA ...
Request: getSNRofLFTbyADD(PLZ, ORT, STR)
Response: -2
End
```

**Abb. 7.5:** Java Client Output

### 7.3.4 Handlers

Handler können benutzt dazu werden um SOAP Nachrichten, bevor oder nachdem sie über das Netzwerk verschickt werden, zu verarbeiten oder zu modifizieren. Ein Handler ist eine Komponente die mit einem Web Service oder einem bestimmten Web Service Port assoziiert ist, und Funktionen wie Verschlüsselung, Entschlüsselung, Logging, Auditing, Caching und Authentifizierung übernehmen kann.

Ein gewöhnlicher Verwendungszweck von Handlern ist für die Überprüfung oder Modifikation von SOAP Headern in einer SOAP Nachricht. Ein Handler muss die javax.xml.rpc.handler.Handler Schnittstelle implementieren, oder alternativ den GenericHandler erweitern. Wenn ein Handler aufgerufen wird, wird ihm die Instanz eines MessageContext weitergereicht, der dazu benutzt werden kann um mittels der SwA (Soap with Attachments) API for Java auf den SOAP Envelope zuzugreifen.

Sobald für einen Service mehrere Handler definiert werden tritt eine Handler Kette in Kraft, welche die Bearbeitungsreihenfolge der einzelnen Handler spezifiziert. Sobald ein Handler mit der Bearbeitung fertig ist, übergibt er dem nächsten Handler in der Kette das Ergebnis. Handler sind eine gute Möglichkeit um SOAP Nachrichten vor- und nachzubearbeiten, jedoch erhöht sich durch die Verwendung von Handlern auch die Antwortzeit der Kommunikationsteilnehmer, da durch Handler eine neue Verarbeitungsebene eingeführt wird. Außerdem gibt es Restriktionen bezüglich den Änderungen die ein Handler bei einer SOAP Nachricht vornehmen darf. Eine einfache Regel besagt, dass Handler hauptsächlich mit SOAP Headern benutzt werden sollen, und nicht etwa die Operationen oder die Nachrichtenstruktur des SOAP Bodys verändern dürfen.

Für das Beispielszenario wurden Handler serverseitig für das Logging der SOAP Request und Response Nachrichten generiert. Beispiel 7.11 zeigt den Output eines solchen Logging Handlers der in einer Logdatei gespeichert wird (Bsp. 7.11).

```
SOAP Request: Fri Feb 09 04:10:43 CET 2007 :
<soapenv:Envelope
    xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header/>
    <soapenv:Body>
        <p113:getLFTbyADD xmlns:p113="http://samples">
            <p113:PLZ>71059</p113:PLZ>
            <p113:ORT>Sindelfingen</p113:ORT>
            <p113:STR>Fronäckerstraße</p113:STR>
        </p113:getLFTbyADD>
    </soapenv:Body>
</soapenv:Envelope>

SOAP Response: Fri Feb 09 04:10:43 CET 2007 :
<soapenv:Envelope
    xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header TimeStamp="Fri Feb 09 04:10:43 CET 2007"/>
    <soapenv:Body>
        <p113:getLFTbyADDResponse xmlns:p113="http://samples">
            <p113:LFT>-2</p113:LFT>
        </p113:getLFTbyADDResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

**Codebsp. 7.11:** ServiceB.log

Eine weitere Möglichkeit, SOAP Nachrichten abzufangen und auszuwerten, besteht über den WSD internen TCP/IP Monitor, der den TCP/IP Verkehr auf einem bestimmten Port abhören kann (Abb. 7.5).

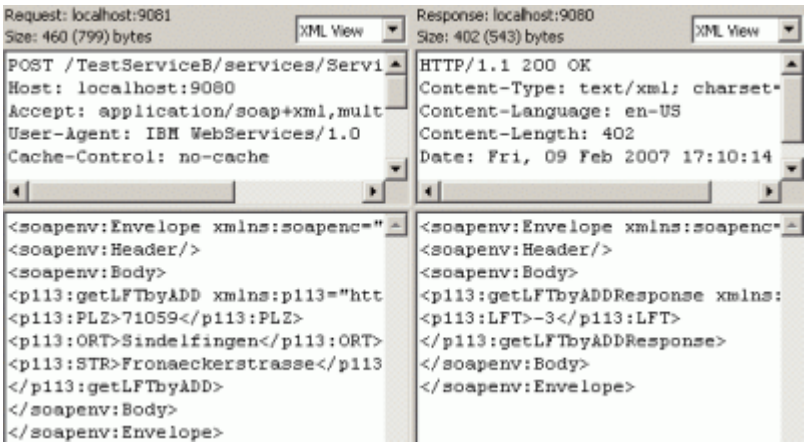



**Abb. 7.6:** TCP/IP Monitor

## 7.4 WS-Security Architektur

WAS 6.0 benutzt eine Erweiterung des JSR109 Deployment Descriptor und Binding Deployment Modells um WS-Security zu implementieren. Die WS-Security Laufzeit wird durch Handler (Kap. 7.3.4) implementiert. Die Handler sind für verschiedene Teilbereiche eines Web Service registriert.

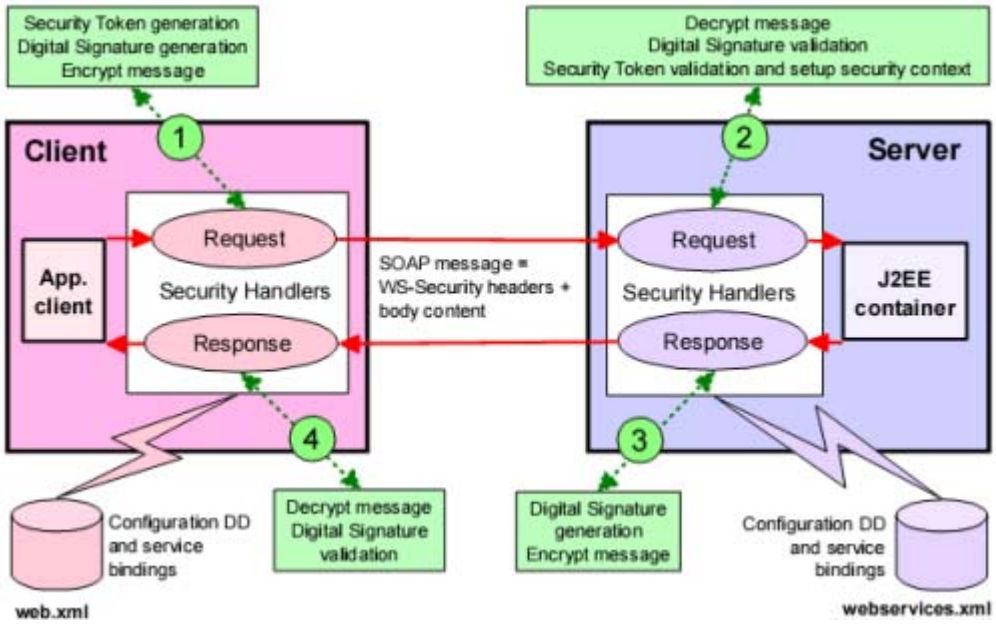



**Abb. 7.7:** Handler-Architektur für WS-Security

**1. Client:** Der „request security handler“ wird ausgeführt, um die erforderlichen Security Header in der SOAP Nachricht unterzubringen bevor diese an den Server geschickt wird.

**2. Server:** Der „request security handler“ wird aufgerufen um zu verifizieren dass die SOAP Nachricht alle Sicherheitsanforderungen erfüllt die durch die Deployment Descriptors des Servers vorgebeben werden, bevor die Nachricht schließlich an die Implementierung des Web Service weitergeleitet wird.

**3. Server:** Der „response security handler“ wird aufgerufen, um die Informationen für den SOAP Security Header zu generieren bevor die Nachricht zurück an den Client geschickt wird.

**4. Client:** Der „response security handler“ wird aufgerufen, um zu verifizieren dass die Sicherheitsinformationen der SOAP Nachricht den Anforderungen, die durch die Deployment Descriptors des Client spezifiziert werden, entsprechen, bevor die Nachricht dann zur Implementierung des Client weitergeleitet wird.

Wenn die Anforderungen der Sicherheitsvorgaben für eine SOAP Nachricht nicht erfüllt werden, wird eine SOAP „fault“ Antwort an den Requester zurückgeschickt.

Die WS-Security Vorgaben werden in der IBM Erweiterung der Web Services Deployment Descriptors und Bindings spezifiziert. Der WS-Security Deployment Descriptor und Binding basiert auf einem Web Service Port. Jeder Web Service Port kann seine eigenen WS-Security Vorgaben definieren. Die WS-Security Anforderungen können außerhalb der Geschäftslogik der Applikation definiert werden.

Es gibt jeweils zwei Paare von Security Handler Konfigurationen auf Client- und Serverseite:

- **Client: Request Generator -** definiert die Vorgaben des „request security handler“ für die abgehende SOAP Request Nachricht, wie das Generieren einer Nachricht mit WS-Security (Signatur, Verschlüsselung und das Anfügen von Security Tokens).

- **Client: Response Consumer -** definiert die Vorgaben des „response security handler“ für die ankommende SOAP Response Nachricht, wie das Sicherstellen dass die erforderlichen Integritätsteile signiert sind (und die Signatur verifiziert wird), und dass die erforderlichen Teile verschlüsselt sind (und eine Verschlüsselung durchgeführt wird), und dass die Security Tokens validiert werden.

- **Server: Request Consumer -** definiert die Vorgaben des „request security handler“ für ankommende SOAP Request Nachricht, wie das Sicherstellen dass die erforderlichen Integritätsteile signiert sind (und die Signatur verifiziert wird), und dass die erforderlichen Teile verschlüsselt sind (und eine Verschlüsselung durchgeführt wird), und dass die Security Tokens validiert werden, und dass der WebSphere Sicherheitskontext mit der zutreffenden Identität bereitgestellt wird

- **Server: Response Generator -** definiert die Vorgaben des „response security header“ für die abgehende SOAP Response Nachricht, wie das Generieren einer Nachricht mit WS-Security (Signatur, Verschlüsselung und das Anfügen von Security Tokens).

Die WS-Security Anforderungen im Request Generator müssen zum Request Consumer passen, und der Response Generator muss wiederum zum Response Consumer passen, ansonsten wird der Request oder die Response abgelehnt. WS-Security verlangt nach einer Verhandlung bzw. einem Austausch von Sicherheitsrichtlinien zwischen Client und Server. Basierend auf den ausgetauschen Sicherheitsrichtlinien sollten folgende vier Sicherheitskonfigurationen definiert werden:

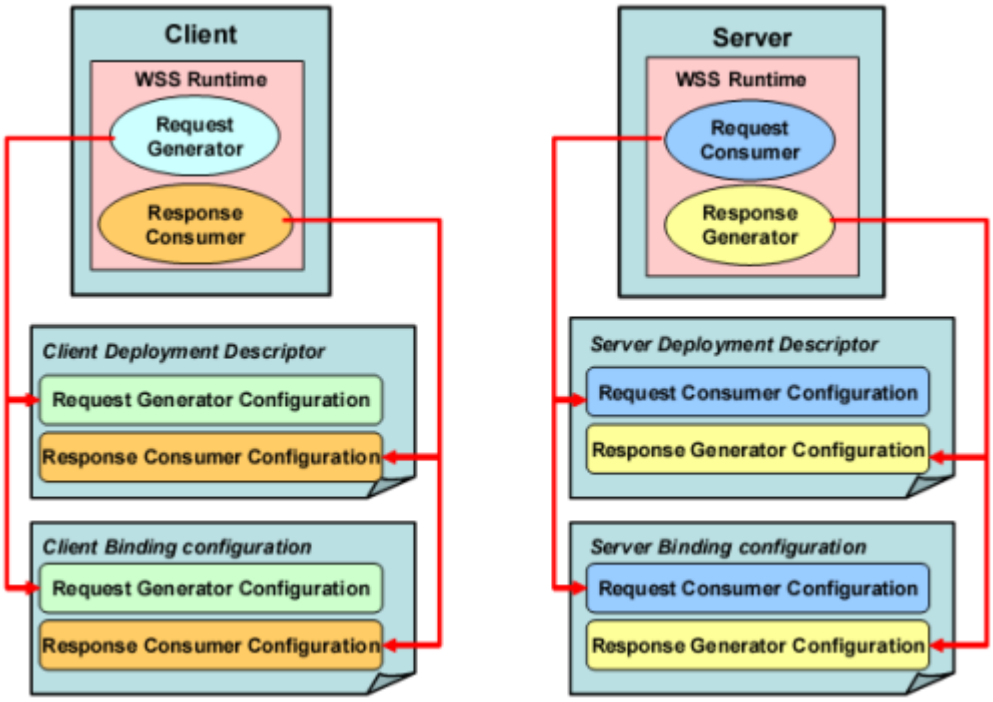



**Abb. 7.8:** Security Handler Konfiguration

Die WS-Security Richtlinien werden in den J2EE Web Services Deployment Descriptors definiert. Es gibt vier Konfigurationsdateien: die Deployment Descriptor Erweiterungen (WS Extension) für Client und Server, und die Binding Dateien (WS Binding) für Client und Server. Diese haben folgende Aufgabe:

- **WS Extension** - spezifiziert, welche Sicherheitsrichtlinien benötigt werden (z.B. das Signieren des Body, und die Verschlüsselung des Username Tokens).

- **WS Binding** - spezifiziert, wie die Sicherheitsrichtlinien, die in WS-Extension spezifiziert werden, angewendet werden (z.B. die Schlüssel für das Signieren/ Verschlüsseln, und welches Security Token in den Header eingefügt wird).

Die einzelnen Konfigurationsschritte werden mit den dafür vorgesehenen Editoren des AST vorgenommen und in folgender Reihenfolge ausgeführt:

1. **Client:** send request message (WS Extension)
2. **Client:** send request message (WS Binding)
3. **Server**: receive request message (WS Extension)
4. **Server:** receive request message (WS Binding)

5. **Server:** send response message (WS Extension)
6. **Server:** send response message (WS Binding)
7. **Client:** receive response message (WS Extension)
8. **Client:** receive response message (WS Binding)

Vor der WS-Security Konfiguration müssen noch „key stores“ (Schlüsselspeicher) vorbereitet werden, um die Nachrichten signieren und/oder verschlüsseln zu können. Für Applikationen in einer realen Produktionsumgebung sind dafür entsprechende Zertifikate und Schlüssel notwendig, in dem begrenzten Rahmen des Beispielszenarios wurde jedoch auf die Konfiguration eigener key stores verzichtet, sondern auf die vom WSD bereitgestellten key stores zurückgegriffen.

## 7.5 WSD Security

Dieses Kapitel stellt die Sicherheitsmechanismen vor, die durch WSD zur Verfügung stehen um Web Service Szenarien zu sichern.

### 7.5.1 Authentifizierung

Um Web Services mit WS-Security benutzen zu können, muss zunächst der Application Server für Security konfiguriert werden. Dazu werden anhand eines Benutzerverzeichnis’ (LDAP, Betriebssystem, oder ein selbst definiertes Verzeichnis) User angelegt, deren Username und Passwort bspw. für ein entsprechendes Username Token im SOAP Security Header zur Authentifizierung verwendet werden können. Abb. 7.6 zeigt ein solches Authentifizierungsszenario.

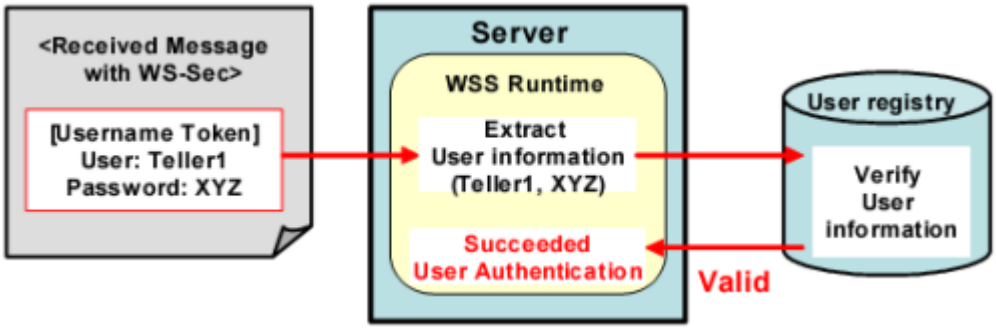



**Abb. 7.9:** Authentifizierung mit Username Token

Die Benutzerinformationen im Security-Header der SOAP Nachricht werden vom Server an das Benutzerverzeichnis weitergeleitet und dort verifiziert. Wenn die Informationen gültig sind, wird das Ergebnis zurück an den Server gegeben der die Nachricht daraufhin akzeptiert, da der Benutzer erfolgreich authentifiziert wurde. Im Username Token Profil wird der „digest“ des Passworts als Passworttyp spezifiziert, aber dieser Typ wird vom WAS 6.0 nicht unterstützt, sondern es kann nur das Klartextpasswort in ein Username Token eingefügt werden. Das bedeutet, dass der Transfer einer Nachricht mit dem Username Token über unsichere Netzwerke entweder durch Transportsicherheit wie SSL/TLS oder durch Nachrichtensicherheit wie WS-Encryption zusätzlich abgesichert werden muss um die Benutzerinformationen vor Fremdzugriffen zu schützen.

Die Assistentenprogramme des AST 6.1 (Version 6.0 verfügt noch nicht über diese Funktion) erlauben eine automatisierte Konfiguration von Authentifizierungsmechanismen mit Security Tokens. WSD stellt standardmäßig zwei BinarySecurity Tokentypen zur Verfügung (Das Kerberos und SAML Token Profil werden vom WAS 6.0 noch nicht unterstützt):

- **X.509 Token**
- **LPTA Token** (Bsp. 7.12)
  Das LPTA token ist ein WebSphere-spezifisches BinarySecurity Token das durch einen WebSphere Sicherheitsmechanismus authentifiziert wird.

```
<S:Envelope>
   <S:Header>
         ...
         <wsse:Security>
            <wsse:BinarySecurityToken
            ValueType="wsst:LTPA">
            nwHBBZwUF+m94fAuY57oQrGFyK…
         </wsse:BinarySecurityToken>
```

**Codebsp. 7.12**: LPTA Token

## 7.5.2 XML-Security

Integrität und Vertraulichkeit einer SOAP Nachricht wird durch XML-Signature und XML-Encryption hergestellt. Einer der größten Vorteile von XML-Security gegenüber SSL/TLS ist, dass die Vertraulichkeit und Integrität einer Nachricht auch für mehrere Empfänger einer Nachricht erhalten werden kann, während es sich bei SSL/TLS nur auf einen einzigen Empfänger beschränkt. Mit dem WAS lassen sich außerdem verschiedene Elemente als Ziel einer Signatur (z.B. body, timestamp, Security Token) oder einer Verschlüsselung (z.B. bodycontent, username token, si-

gnature) bestimmen. Mit diesen selektiven Auswahlmöglichkeiten lassen sich flexible und komplexe Sicherheitsanforderungen umsetzen, wie sie bspw. mit SSL/ TLS nicht möglich wären. Die Assistentenprogramme des AST 6.0 erlauben eine automatisierte Konfiguration von XML-Signature und XML-Encryption mit vordefinierten key stores (in Version 6.1 können eigene key stores spezifiziert werden).

Ein Beispiel aus dem Szenario zeigt eine SOAP Request Nachricht, die von Service A an Service B geschickt wird, und deren Body-Element (welches die Adressdaten des Lieferanten (ADD) enthält) signiert und verschlüsselt wurde. Aus Gründen der Übersichtlichkeit wurde die Nachricht in Teilstücke zerlegt, und einige Namespaces, und Informationen zur Kanonisation und Transformation entfernt.

```
<soapenv:Envelope
<soapenv:Header>
<wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
soapenv:mustUnderstand="1">
<wsse:BinarySecurityToken
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
wsu:Id="x509bst_754033076593207236">
MIIDQTCCAqqgAwIBAgICAQQwDQYJKo...
</wsse:BinarySecurityToken>
```

**Codebsp. 7.13:** SOAP mit XML-Security - Teil 1

Bsp. 7.13 zeigt einen SOAP Security Header innerhalb des SOAP Envelope. Das mittels des <wsse:BinarySecurityToken> Elements spezifizierte X.509 Security Token gehört der digitalen Signatur an, die den SOAP Body der Nachricht signiert.

```
<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier
ValueType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-x509-token-profile-1.0#X509v3SubjectKeyIdentifier">
/62wXObED7z6c1yX7QkvN1thQdY=
</wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<CipherData>
<CipherValue>
RmjDIBS6Tj4gCVZieKruNU...
</CipherValue>
</CipherData>
<ReferenceList>
<DataReference URI="#wssecurity_encryption_id_7893019825006154444"/>
</ReferenceList>
</EncryptedKey>
```

**Codebsp. 7.14:** SOAP mit XML-Security - Teil 2

Bsp. 7.14 zeigt die Verschlüsselungsinformationen innerhalb des <EncryptedKey> Elements mit denen der SOAP Body verschlüsselt wurde.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#wssecurity_signature_id_9046116397874166835">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
ZLJZH3/TVjdvicEc9XtdbkGMOww=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
S2lzBiWXPA4+l...
</ds:SignatureValue>
<ds:KeyInfo>
<wsse:SecurityTokenReference>
<wsse:Reference URI="#x509bst_754033076593207236"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
```

**Codebsp. 7.15:** SOAP mit XML-Security - Teil 3

Bsp. 7.15 zeigt innerhalb des <ds:Signature> Elements, die Informationen zur Signatur mit welcher der SOAP Body signiert wurde.

```
</wsse:Security>
</soapenv:Header>
<soapenv:Body
xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsu:Id="wssecurity_signature_id_9046116397874166835">
<EncryptedData
xmlns=http://www.w3.org/2001/04/xmlenc#
Id="wssecurity_encryption_id_7893019825006154444"
Type="http://www.w3.org/2001/04/xmlenc#Content">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
<CipherData>
<CipherValue>
vSnzs1+Wh7qj5CepRHHHZhge...
</CipherValue>
</CipherData>
</EncryptedData>
</soapenv:Body>
</soapenv:Envelope>
```

**Codebsp. 7.16:** SOAP mit XML-Security - Teil 4

Bsp. 7.16 zeigt den verschlüsselten SOAP Body innerhalb des <EncryptedData> Elements. Die Antwort des Servers auf den SOAP Request ist bezüglich der Security Erweiterungen im SOAP Header analog aufgebaut.

### 7.5.3 WSD Security Extensions

WebSphere Developer verfügt über weitere WS-Security Mechanismen, von denen einige standardmäßig nicht von anderen Herstellern unterstützt werden, sondern die spezifische Erweiterungen der WSD Funktionalitäten darstellen.

• **Identity assertion (IDAssertion) -** ein weiterer Authentifizierungsmechanismus, der jedoch zwischen drei Parteien errichtet wird: dem Client, dem Endpoint Server, und einem dazwischenliegenden vermittelnden Server, der den Endpoint Server um die Aufgabe der Authentifizierung des Clienten entlastet. Dadurch wird erreicht, dass das Passwort des Benutzers nicht bis zum Endpoint Server transportiert werden muss.

• **Pluggable token architecture -** lässt den Benutzer eigene Tokentypen spezifizieren und implementieren. Dazu werden vom WAS entsprechende Schnittstellen zur Verfügung gestellt, die von den entsprechenden Klassen des Benutzers implementiert werden können.

• **Timestamp extension** - ermöglicht das Hinzufügen eines Zeitstempels innerhalb eines Zielelements von XML-Signature oder XML-Encryption, um dieses mit einer Lebensdauer zu versehen (andere Application Server könnten eine Nachricht mit dieser Funktion aber womöglich nicht verarbeiten).

• **Nonce -** ein zufällig generierter Wert der ebenfalls in ein Zielelement von XML-Signature oder XML-Encryption eingefügt werden kann. Es reduziert die Chance auf einen Wiederholungsangriff (andere Application Server könnten eine Nachricht mit dieser Funktion aber womöglich nicht verarbeiten).

• **Certificate caching -** verbessert die Performance des Systems, indem das Ergebnis einer Authentifizierung mittels eines Zertifikats im lokalen Cache des Application Servers abgespeichert wird. Beim nächsten Request des Clienten kann die Verifizierung aus dem Cache benutzt werden, anstatt den erheblichen Aufwand zu betreiben, das Zertifikat erneut zu verifizieren.

### 7.5.4 Weitere Sicherheitsmechanismen

Um Integrität und Vertraulichkeit der Kommunikation zweier Parteien herzustellen, kann ein Sicherheitskontext zwischen den Teilnehmern eingerichtet werden. Um einen solchen Sicherheitskontext zu etablieren und benutzen zu können, müssen beide Teilnehmer vor der Kommunikation security credentials austauschen mit denen ein gegenseitiges Vertrauen aufgebaut wird. Für solche Szenarien gibt es zwei Spezifikationen, die diese Aufgaben lösen:

- **WS-Trust -** beschreibt, wie Security Tokens ausgetauscht werden können
- **WS-SecureConversation -** beschreibt, wie ein Security Context Token zwischen zwei Teilnehmern etabliert werden kann

Die Implementierung eines Security Context Token wird beim WAS 6.0 nicht bereitgestellt, es gibt jedoch eine erweiterbare Architektur, mit der es möglich ist ein Security Context Token für den WAS zu implementieren.

Desweiteren gibt es auch eine WS-SecurityKerberos Spezifikation, die beschreibt wie Web Services mit Kerberos zusammen benutzt werden können, jedoch ist diese Implementierung ebenfalls nicht Teil der Standardfunktionen des WAS 6.0.

## 7.6 Ergebnisse

Mit dem WSD lassen sich Web Services auf verschiedene Weisen realisieren. Es werden umfangreiche Client-Typen und Möglichkeiten zum Testen der Web Services angeboten, und mit den WSDL und UDDI Werkzeugen lassen sich serviceorientierte Szenarien schnell realisieren.

Mit der standardmäßigen SSL-Unterstützung des WebSphere Application Servers lässt sich die Sicherheit auf Transportebene herstellen. Die Integration der WS-Security Spezifikationen erlaubt das Konfigurieren von XML-Signature und XML-Encryption. Desweiteren lassen sich grundlegende Authentifizierungskonzepte mit Security Tokens wie Username/Passwort oder X.509 Zertifikate realisieren. Das Konzept der Deployment Descriptors erlaubt eine einfache Konfiguration der WS-Security Spezifikationen, und das Konzept der Handler ermöglicht die Implementierung weiterer Funktionalitäten, wie bspw. das Manipulieren von SOAP Headern durch Sicherheitsmaßnahmen, oder andere nützliche Funktionen wie logging, auditing oder caching. Die eingebauten WS-I Werkzeuge können die Implementierungen anhand der unterstützen WS-I Profile auf Interoperabilität überprüfen lassen. Komplexere Sicherheitsszenarien, wie sie durch SAML, XMKS oder WS-Trust und WS-SecureConversation möglich wären, lassen sich dagegen noch nicht standardmässig realisieren.

# Kapitel 8: Zusammenfassung und Ausblick

Das Ziel dieser Diplomarbeit war es, einen Überblick über die Web Service spezifischen Sicherheitsstandards zu verschaffen, und das Konzept einer SOA mittels des Web Service Modells zu veranschaulichen.

Kapitel 2 bot einen Einstieg in die Begrifflichkeiten der serviceorientierten Architektur. Es wurde das SOA Konzept, die Motivation eine SOA umzusetzen, die Vor- und Nachteile und die Entwicklung von SOA vorgestellt. Weiterhin wurde der Begriff des Service definiert, der in Kapitel 3 um die Definition des Web Service ergänzt wurde. Dieses Kapitel beschäftigte sich mit dem Überblick über das Web Service Modell. Es wurden die Schlüsselelemente spezifiziert, und die Basiskomponenten vorgestellt die ein Web Service Szenario ermöglichen. Desweiteren wurde ein Rückblick zu ähnlichen Programmierkonzepten wie CORBA und DCE vorgenommen, und eine Übersicht über die Stanardisierungsgremien und deren Web Service relevanten Standards vorgestellt. Kapitel 4 beschrieb die allgemeinen Sicherheitsanforderungen und ihre technischen Umsetzungen die in einer Sicherheitsinfrastruktur für Web Services eine Rolle spielen. In Kapitel 5 wurden die Web Service spezifischen Sicherheitsspezifikationen, die WS-Security Standards, vorgestellt, das Konzept und die Funktion der Security Tokens wurde erläutert, und eine Unterscheidung zwischen Transport- und Nachrichtensicherheit vorgenommen. Um den Überblick zu vervollständigen, wurden weitere Web Service relevante Sicherheitskonzepte aufgegriffen.

Kapitel 6 enthält die Vorstellung eines Modellentwurfs mit Web Services, eine Klassifizierung der Daten und ein davon abgeleiteter Vorschlag für eine Sicherheitsinfrastruktur. Die in den vorherigen Kapiteln spezifizierten Sicherheitsmaßnahmen wurden nun auf Transport- und Nachrichtenebene aufgegliedert und auf das Modell übertragen, wodurch sich ein Lösungssansatz für eine Sicherheitsinfrastruktur herausgebildet hat. Kapitel 7 beschrieb schließlich die praktische Untersuchung der in den vorherigen Kapiteln vorgestellten Konzepte der Web Services und der relevanten Sicherheitsstandards. Zunächst wurde die IBM Entwicklungsumgebung WebSphere Developer vorgestellt, und daraufhin folgt die Beschreibung der Java-spezifischen Web Service APIs mit denen sich plattformunabhängige Web Service Szenarios realisieren lassen. Es folgten Beispiele wie Web Services mit Hil-

fe des WSD entwickelt und getestet werden können, welche Client Typen möglich sind, und wie das Konzept der Handler funktioniert, die Funktionalitäten wie Sicherheit und Logging ermöglichen. Die Beschreibung der WSD-spezifischen Sicherheitsarchitektur und -Implementierungen schlossen das Kapitel ab.

Diese Diplomarbeit demonstriert, dass man mit Entwicklungsumgebungen wie dem WebSphere Developer von IBM in der Lage ist, eine serviceorientierte Architektur mit Web Services zu realisieren, und entsprechende Sicherheitsmaßnahmen darauf abzubilden. Es ist jedoch wichtig, dass komplexe Sicherheitstechnologien wie XML-Signature und XML-Encrpytion auch in Zukunft Teil der Produkte werden, da eine grundlegende Implementierung dieser Konzepte zeit- und kostenaufwendig ist. Ähnliches gilt für komplexe Standards wie SAML oder XKMS, deren praktischer Einsatz wesentlich von der Komplexität ihrer Implementierung abhängt.

Wie auch bei allen vorherigen Erfolgskonzepten wie dem Client/Server Computing oder der objektorientierten Programmierung gilt auch für das Konzept der serviceorientierten Architektur bzw. des Web Service Modells, dass eine längerfristige Durchsetzung auch von der Standardisierungsarbeit abhängt, die eine weiträume Verbreitung und Akzeptanz oft erst möglich macht. Die Unterstützung vieler namenhafter Firmen für Web Services lässt zwar vermuten, dass man sich auf einem richtigen Weg befindet, jedoch muss sich erst noch herausstellen inwieweit sich dieses Konzept auch tatsächlich für die Implementierung einer serviceorientierten Architektur durchsetzen wird. Eine aktuelle Studie der Butler Group besagt, dass bereits viele Unternehmen an SOA gescheitert sind, da die Konzentration zu sehr auf der Technik liegt, und weniger auf der Frage inwiefern die Geschäftsprozesse überhaupt gesteuert werden sollen [BG07].

Organisationen wie OASIS oder das W3C tragen zu einer Standardisierung der technischen Basis bei, und mit ihren Spezifikationen lassen sich bereits Sicherheitsmechanismen implementieren die in Zusammenarbeit mit gängigen Sicherheitstechnologien eine Infrastruktur für den sicheren Datenaustausch zwischen Web Services ermöglichen. Jedoch bedarf es aufgrund der großen Menge an Spezifikationen und teilweise überlappender und mangelhafter Standards auch einer Organisationen wie WS-I, deren Arbeit für die Interoperabilität zwischen den Systemen, Plattformen und Spezifikationen Grundsteine für den Erfolg dieser Konzepte legt.

# Literaturverzeichnis


[Bruce04] Bruce Silver Associates: *Enterprise Service Bus Technology for Real-World Solutions*

[BG07] Butler Group: *Planning and Implementing SOA*

[CEB04] Corporate Execution Board: *Service-oriented Architecture: Theory and Praxis*

[Cull00] A. McCullagh, W.Caelli: *Non-Repudiation in the Digital Environment*. (WWW Dokument) http://www.firstmonday.dk/issues/issue5_8/mccullagh/#m3

[CWM01] OMG Group: *CWM Web Services Specification*

[CDBI03] *Web Services Usage Survey* (WWW document) http://www.cbdiforum.com/bronze/webserv_usage/webserv_usage.pdf

[CT07] c't Magazin: *Webstandards: Die Krise des W3C*

[DCX05] C. Costadin, M. Ziegler, T. Eckardt: *Web Service Reference Architecture* (DaimlerChrysler internes Dokument)

[DCX06] *DaimlerChrysler*

[Dost05] W. Dostal, M. Jeckle, I. Melzer: *Service-orientierte Architekturen mit Web Services*

[Hims05] M. Himsolt, M. Jeckle, I. Melzer, B. Zengler: *Web Services Standards Overview*

[IETF06] *The Internet Engineering Task Force* (WWW Dokument) http://www.ietf.org/

[IBM05a] Redbook Patterns: *SOA with an Enterprise Service Bus in Web Sphere Application Server V6* (WWW Dokument) http://www.redbooks.ibm.com/redbooks/pdfs/sg246494.pdf

[IBM05b] Redbook: *WebSphere Application Server V6 Planning and Design WebSphere Handbook Series* (WWW Dokument) http://www.redbooks.ibm.com/redbooks/pdfs/sg246446.pdf

[IBM05c] Redbook: *WebSphere Version 6 Web Services Handbook Develop ment and Deployment* (WWW Dokument) http://www.redbooks.ibm.com/redbooks/pdfs/sg246461.pdf

[IBM06a] Redbook: *Web Services Handbook for WebSphere Application Server Version 6.1* (WWW Dokument) http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf

[IBM06b] Redbook: *IBM WebSphere Application Server V6.1 Security Handbook* (WWW Dokument) http://www.redbooks.ibm.com/redbooks/pdfs/sg246316.pdf

[IBM01] IBM: *WSIL Specification* (WWW Dokument) ftp://www6.software.ibm.com/software/developer/library/ws-wsilspec.pdf

[IBM03] IBM: *WS-Security Kerberos* (WWW Dokument) http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-seckerb/WS-Security-Kerberos.pdf

[Jeck04] M. Jeckle: *Web Services* (WWW Dokument) http://www.jeckle.de/webServices/index.html

[MCG06] M. Herrmann, M. A. Aslam: *Mercedes Car Group (MCG) Enterprise Architektur - Ein Ansatz zur semantischen Modellierung der Services in einer SOA*

[MIT07] MIT: *Kerberos* (WWW Dokument) http://web.mit.edu/kerberos/www/

[OMG07] Object Management Group: *OMG Security*. (WWW Dokument) http://www.omg.org/technology/documents/formal/omg_security.htm

[OPEN05] The Open Group: *DCE Portal*. (WWW Dokument) http://www.opengroup.org/dce/

[OASIS06] *OASIS Standards* (WWW Dokument)
http://www.oasis-open.org/specs/index.php

[Pezz06] M. Pezzini: *An SOA Maturity Model: Where Do You Stand and Where Are You Going?*

[Rose04] J. Rosenberg, D. Remy: *Securing Web Services with WS-Security*

[SAP07] SAP AG: *Sicher im Netz*
https://www.sicher-im-netz.de/

[Schn05] C. Schnupfhagn, G. Sopper: *SOA und SOO – das A und O der Serviceorientierung*

[W3C06] *World Wide Web Consortium* (WWW Dokument)
http://www.w3.org/

[WSI06a] *Web Services Interoperability Organization* (WWW Dokument)
http://www.ws-i.org/

[WSI06b] WSI: *Basic Security Profile* (WWW Dokument)
http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

[WSS06] OASIS: WSS - Web Services Security (WWW Dokument)
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[W3C02a] W3C: *XML-Signature Specification* (WWW Dokument)
http://www.w3.org/TR/xmldsig-core/

[W3C02b] W3C: *XML-Encryption Specification* (WWW Dokument)
http://www.w3.org/TR/xmlenc-core/

[WSD06] IBM: *WebSphere Developer Help Contents*

# Abkürzungen

| | |
|---|---|
| A2A | Application-to-Application |
| API | Application Programming Interface |
| ASAP | Asynchronous Services Access Protocol |
| AST | Application Server Toolkit |
| B2B | Business-to-Business |
| BEEP | Blocks Extensible Exchange Protocol |
| BPEL | Business Process Execution Language |
| BPM | Business Process Management |
| CA | Certificate Authority |
| CWM | Common Warehouse Metamodel |
| WS-CAF | Web Services Composite Application Framework |
| WS-CDL | Web Services Choreography Description Language |
| CORBA | Common Object Request Broker Architecture |
| DCE | Distributed Computing Environment |
| DII | Dynamic Invocation Interface |
| DCOM | Distributed Component Object Model |
| DOM | Document Object Model |
| DSI | Dynamic Skeleton Interface |
| EJB | Enterprise Java Beans |
| ESB | Enterprise Service Bus |
| FTP | File Transfer Protocol |
| GIOP | General Inter-Orb Protocol |
| H2A | Human-to-Application |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| IDE | Integrated Development Environment |
| IDL | Interface Definition Language |
| IETF | Internet Engineering Task Force |
| IIOP | Internet Inter-Orb Protocol |
| IP | Internet Protocol |
| IT | Information Technology |
| J2EE | Java 2 Enterprise Edition |
| JAXB | Java Architecture for XML Binding |
| JMS | Java Messaging Service |
| JSP | Java Server Pages |

| | |
|---|---|
| JLS | Java Language Specification |
| JSR | Java Specification Request |
| KDC | Key Distribution Center |
| LDAP | Lightweight Directory Access Protocol |
| LGPL | Lesser General Public License |
| MOM | Message Oriented Middleware |
| OMG | Object Management Group |
| OSF | Open Software Foundation |
| P2P | Peer-to-Peer |
| P3P | Platform for Privacy Preferences |
| PKI | Public Key Infrastructure |
| RAD | Rational Application Developer |
| REST | Representational State Transfer |
| RPC | Remote Procedure Call |
| SAML | Security Assertion Markup Language |
| SHA | Secure Hash Algorithm |
| SMTP | Simple Mail Transfer Protocol |
| SOA | Service-oriented Architecture |
| SPML | Service Provisioning Markup Language |
| SSL | Secure Socket Layer |
| SSO | Single Sign On |
| ST | Service Ticket |
| TCP | Transmission Control Protocol |
| TGT | Ticket Granting Ticket |
| TLS | Transport Layer Security |
| UDDI | Universal Description Discovery and Integration |
| UNSPSC | United Nations Standard Products and Services Code |
| UPS | United Parcel Service |
| URI | Uniform Resource Identifier |
| WAS | WebSphere Application Server |
| W3C | World Wide Web Consortium |
| WSDL | Web Service Description Language |
| WSDM | Web Services Distributed Management |
| WSD | WebSphere Developer |
| WS-I | Web Services Interoperability Organization |
| WSIL | Web Services Inspection Language |
| WSS | Web Services Security |
| WSSE | Web Services Security Extension |
| WTP | Web Tool Platform |
| XACML | Extensible Access Control Markup Language |
| X-KISS | XML Key Information Service Specification |
| X-KRSS | XML Key Registration Service Specification |
| XKMS | Extensible Key Management System |
| XML | Extensible Markup Language |

# Abbildungen

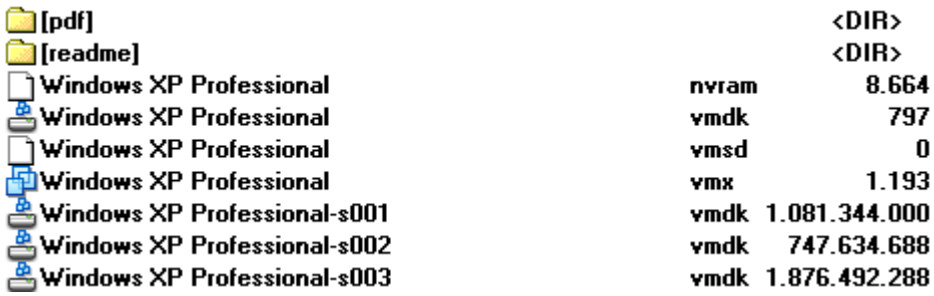
# Tabellen

# Codebeispiele

# Inhalt der DVDs

Die beigefügten DVDs weisen folgende Verzeichnisstruktur auf:

## DVD1:




- **[pdf]:** Beinhaltet die Diplomarbeit als PDF Datei
- **[readme]:** Anleitung zum Starten des VMware Image

Die Imagedateien erstrecken sich über beide DVDs, und enthalten ein Windows XP mit der Installation des IBM WebSphere Developer incl. Application Server Testumgebung und dem workspace des Beispielszenarios.

## DVD2: