

---

## 10 Metadaten und Interoperabilität von Diensten

### 10.1 Überblick

In den letzten Kapiteln haben wir erfahren, wie ein Dienst aufgebaut ist, wie dieser durch die WSDL repräsentiert wird und wie man den Dienst nutzt. In diesen Beschreibungen fehlte bisher der Aspekt der Dienstigenschaften (oder auch der Richtlinien des Dienstes) vollständig. So haben wir zum Beispiel nicht beschrieben, ob der Dienst einen sicheren Transfer von Nachrichten benötigt oder ob zum Beispiel eine Optimierung von Binärdaten während der Übertragung stattfinden soll.

In diesem Kapitel werden wir jetzt erfahren, wie man Dienstrichtlinien beschreiben kann. Dazu werden wir jedoch noch nicht speziell auf einzelne Richtlinien eingehen, sondern vielmehr die zugrunde liegenden Frameworks, Technologien und Spezifikationen kennenlernen. Im weiteren Teil des Kapitels werden wir an unserem Beispiel des ATM-Dienstes erfahren, wie man diese Metadaten in die Dienstbeschreibung einbaut und nutzt. Im Verlauf des Buches, zum Beispiel beim Thema Sicherheit (siehe Kapitel 13), sind diese Grundlagen notwendig, die jeweiligen speziellen Richtlinien werden ausführlich beschrieben.

### 10.2 Richtlinien an Dienste formulieren: WS-Policy

#### 10.2.1 Ziele von WS-Policy

Beginnen wir mit einer viel beachteten Pressemitteilung vom 4. September 2007: »Das W3C gibt heute einen entscheidenden Web-Standard für die Funktionserweiterung von Web Services und serviceorientierten Architekturen (SOA) heraus. ...« [48] Und damit ist die Dringlichkeit eines solchen Standards schon fast erklärt, aber jetzt zurück zu unseren Diensten.

Betrachten wir den in den vergangenen Kapiteln beschriebenen Dienst genauer, so können wir die folgenden Dinge festhalten:

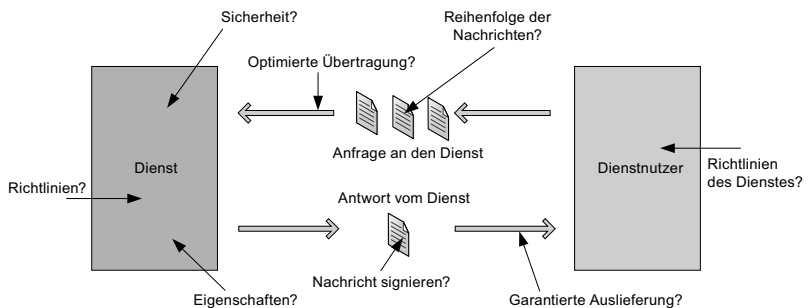
- Die Schnittstelle des Dienstes ist definiert.
- Das Format der Nachricht und die Nachricht selbst ist beschrieben.
- Die Adresse des Dienstes ist definiert.
- Das Transportprotokoll ist festgelegt.

*Gedankenexperiment*

Was in der Aufzählung jedoch vollständig fehlt, ist die modulare Beschreibung von Richtlinien. Stellen Sie sich doch einmal vor, wir möchten einen Dienstenutzer für den ATM-Dienst entwickeln. In der Vergangenheit musste dazu nicht nur die Beschreibung (WSDL) bekannt sein. Wir mussten auch die technischen Eigenschaften des Systems kennen, das den Dienst bereitstellt. Um bei der Sicherheit zu bleiben, mussten wir wissen, wie die Nachricht verschlüsselt wird, welcher Algorithmus verwandt wurde, wie der Aufbau der Nachricht nach der Verschlüsselung aussieht und wie auch immer im speziellen Fall die Eigenschaften der Sicherheitsbeschreibung aussehen. Das Ganze gipfelt dann darin, dass wir unser lokales System dem gegenüberliegenden System angepasst haben, in der Hoffnung, dass es überhaupt möglich war (siehe Abbildung 10-1). Wenn jetzt der Anbieter des Dienstes etwas ändert, geht das Spiel wieder von vorn los. Sie werden sicherlich verstehen, dass die Liste hier beliebig fortzusetzen ist.

*Hoffnung auf einen Standard*

**Abb. 10-1**  
Dienst und Richtlinien



Mit WS-Policy werden die folgenden Fragen und Problemkreise adressiert:

- Beschreibung optionaler oder benötigter Richtlinien an Dienst und Dienstenutzer
- Richtlinien dynamisch aktivieren oder deaktivieren, ohne die Implementierung des Dienstes zu verändern
- Richtlinien dynamisch aktivieren oder deaktivieren, ohne den Dienst zu beenden oder neu starten zu müssen

- grundlegende Spezifikationen (WS-Policy und Policy Assertion) für ein Rahmenwerk
- spezielle Spezifikationen für Sicherheit, Transaktionen (WS-SecurePolicy etc.)
- Richtlinien lassen sich sowohl von Entwicklungswerkzeugen als auch von den Service-Bus-Implementierungen auswerten.

Zusammengefasst wird es mit der WS-Policy-Spezifikation möglich [47], einfache und komplexe Richtlinien für einen Dienst zu beschreiben und auszuwerten. Die Abhängigkeit von verschiedenen Anbietern sinkt, und die Austauschbarkeit wird erhöht. Nicht zuletzt schon das unsere Nerven und den Geldbeutel – werden doch nicht für jeden Dienst die verschiedensten, inkompatiblen Anpassungen notwendig.

## 10.2.2 WS-Policy im Detail

### Bestandteile und Beschreibung

Richtlinien für einen Dienst sind in seiner Beschreibung (WSDL) abzulegen (siehe Abbildung 10-2). Die Verwendung von WS-Policy in der Web Service Description Language wird mit folgendem Quelltextfragment angezeigt:

```
<wsp:UsingPolicy
  wsdl:Required="true"/>
```

Betrachtet man Beispiele in der Literatur oder im Web, ist oft die kürzere Form `<wsp:UsingPolicy/>` zu finden. Hierbei ist zu beachten, dass einige Service-Bus-Implementierungen existieren, die alle Richtlinien (Policies) ignorieren, wenn das zusätzliche Attribut »Required« nicht gesetzt ist. Also schreiben wir es am besten dazu, da es eigentlich auch nicht stört.

Die Beschreibung der Richtlinien wird mittels der folgenden Bestandteile durchgeführt (wir ziehen es vor, bei den englischen Begriffen zu bleiben, da wir diese auch während der Entwicklung vorfinden). Diese abstrakte Beschreibung wird auch als *Policy Model* bezeichnet.

#### ■ Policy Assertion

Eine Policy Assertion beschreibt eine Möglichkeit, eine Pflicht oder auch eine Verhaltenseigenschaft. Eine Menge dieser Richtlinien kann einer Entität (Endpunkt, Nachricht, Operation etc.) in der WSDL zugeordnet werden. Welche Richtlinien existieren, wird durch die WS-Policy-Spezifikation nicht vorgegeben und wird in speziellen Domain-spezifischen (Sicherheit, Transaktion etc.) Spezifikationen definiert.

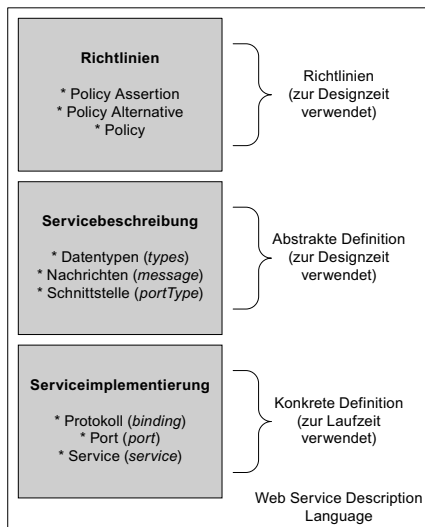
#### **Listing 10.1**

*Richtlinien einschalten*

*wsdl:Required oder nicht*

*Policy Model*

**Abb. 10-2**  
Erweiterung der WSDL  
um den Policy-Block



#### Lax Mode

Eine Richtlinie kann für sich selbst definieren, dass sie ignoriert werden darf. Eine solche Richtlinie kann ignoriert werden, wenn es um die Feststellung von Kompatibilität zwischen zwei Parteien geht und der Policy-Vergleich sich nicht im »Lax Mode« befindet. Richtlinien werden vom Autor des Dienstes definiert und können durch Ausdrücke zu Gruppen oder Alternativen zusammengefasst werden.

#### ■ Policy-Alternative

Eine Policy-Alternative ist eine Menge von Richtlinien. Diese Menge kann auch leer sein.

#### ■ Policy

Die Policy ist eine Menge von Alternativen. Diese Menge kann auch leer sein.

Da das Ganze etwas abstrakt ist (mehr dazu finden Sie unter [47]), zeigen wir jetzt erst einmal ein illustrierendes Beispiel. In der Beschreibung der Richtlinien wird im Beispiel festgelegt, dass der Dienstendpunkt vollständig kompatibel zur WS-Addressing-Spezifikation ist (`wsaw:UsingAddressing`). Im Weiteren wird spezifiziert, wie viel Zeit maximal zwischen dem Empfangen von zwei zusammengehörigen Nachrichten vergehen darf (`wsrn:InactivityTimeout`). Seien Sie nicht zu ungeduldig, wenn an dieser Stelle noch nicht alle Details verständlich sind. Wie schon gesagt, werden die spezifischen Richtlinien durch entsprechende weitere Spezifikationen definiert – und die folgen in den nächsten Kapiteln, hierbei jeweils im zugehörigen Themenkomplex. Möchten

Sie also etwas über die Policies zur Sicherheit erfahren, müssen Sie im Kapitel Sicherheit nachschlagen.

```
<wsp:Policy wsu:Id="ATMService_Policy">
  <wsp:ExactlyOne>
    <wsp>All>
      <wsaw:UsingAddressing/>
      <wsrm:RMAssertion>
        <wsrm:InactivityTimeout Milliseconds="600000"/>
      </wsrm:RMAssertion>
    </wsp>All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

**Listing 10.2**

*Ein illustrierendes  
Beispiel*

**Austausch der Richtlinien**

Wenn Sie jetzt einmal überlegen, was wir mit den Richtlinien für den Dienst beschreiben, fällt deren Bedeutung für eigentlich alle Bereiche der Entwicklung auf. Möchte man einen Dienstenutzer entwickeln, kann auf diese Definition zurückgegriffen werden. So wäre auch eine visuelle Unterstützung im Entwicklungswerkzeug möglich. Leider existiert für unser verwendetes Eclipse [8] so etwas noch nicht, für einige andere Entwicklungsumgebungen schon [23].

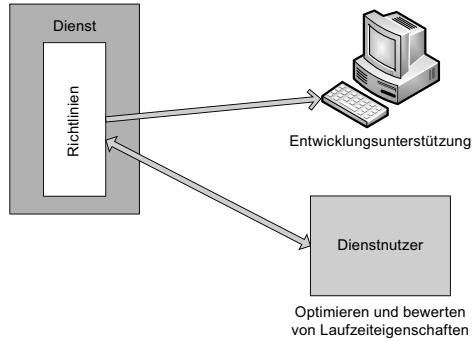
Eine weitere Möglichkeit ist der Austausch der Informationen auf Ebene der beteiligten Systeme. Wenn wir zum Beispiel im Dienst eine optimierte Übertragung von Binärdaten vereinbaren, können sowohl die Systeme vom Dienst als auch vom Dienstenutzer darauf reagieren. Im besten Fall erhalten wir somit eine optimale Übertragungsgeschwindigkeit der Daten. Die Kette von Möglichkeiten ließe sich jetzt natürlich beliebig fortsetzen. Wichtig bei alledem ist jedoch vor allem die Standardisierung hinter der WS-Policy-Spezifikation. Alle Systeme, die diese Möglichkeit der Definition von Richtlinien unterstützen, sind zueinander kompatibel.

**Domain-spezifische Richtlinien**

Nach der Erarbeitung der grundlegenden Spezifikation, WS-Policy, könnte man jetzt Domain für Domain betrachten. In jeder der einzelnen Domains erfolgt jeweils eine eigenständige Definition von Richtlinien.

Derzeit sind die folgenden wichtigen spezifischen Richtlinien und ihre Spezifikationen zu finden:

**Abb. 10-3**  
Richtlinien für  
Entwicklung und  
Nutzung



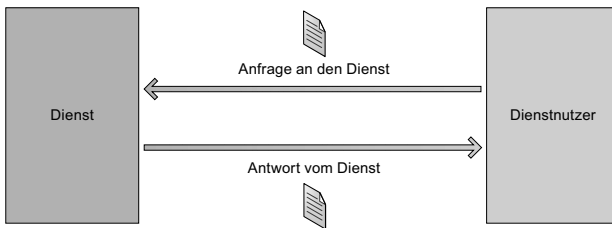
- **Web Service Security Policy Language (WS-SecurityPolicy) [53]**  
Die *Web Service Security Policy Language* beschreibt Richtlinien, die sich auf die Sicherheit der Nachricht und Nachrichtenübertragung beziehen. Dabei werden auch die weiterführenden Spezifikation *SOAP Message Security* [28], *WS-Trust* [56] und *WS-SecureConversion* [51] berücksichtigt.
- **WS-Transactions [42][43]**  
In dieser Spezifikation, oder besser gesagt in den verschiedenen Teilspezifikationen, werden Mechanismen für die transaktionale Kopplung von Diensten beschrieben. Obwohl sie für alle Transaktionsarten definiert sind, werden von vielen Web-Service-Engines lediglich *WS-AtomicTransaction*-Richtlinien plattformübergreifend unterstützt.
- **Web Service Reliable Messaging Policy (WS-RM Policy) [49]**  
Diese Spezifikation beschreibt die Domain-spezifischen Richtlinien für *WS-ReliableMessaging*. Das beinhaltet zum Beispiel die garantierte Auslieferung der Nachrichten in festgelegter Reihenfolge, die Beziehungen zu WS-Security und die Qualität der Auslieferung von Nachrichten beim wiederholten Senden (garantiert, wenigstens einmal, höchstens einmal etc.).

## 10.3 Routing der Nachrichten: WS-Addressing

### 10.3.1 Überblick

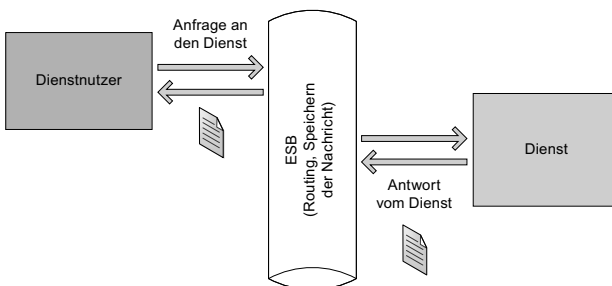
In den ersten Beispielen haben wir gelegentlich schon einmal die Nachrichten betrachtet. Bisher ist uns dabei wahrscheinlich noch nicht aufgefallen, dass es keine expliziten Informationen zum Ziel der Nachricht gab. Das ist auch nicht weiter verwunderlich, hatten wir es doch mit

einer Punkt-zu-Punkt-Verbindung zu tun. Oder, um in unserem Beispiel zu bleiben, wir (und damit auch Herr Kainer) wussten genau, wo sich der Dienst befindet. An der Kommunikation war kein Service-Bus beteiligt, und der Empfänger konnte daher auch die Antwort auf direktem Weg erhalten (Abbildung 10-4). Solche Szenarien sind durchaus für verschiedene Probleme als Lösung denkbar. Wenn Sie lediglich den Export von Daten aus Ihrer Anwendung modellieren wollen, eine sehr begrenzte Anzahl von Diensten anbieten (zum Beispiel den aktuellen Comic-Strip des Tages) etc., kann eine direkte Kommunikation, ohne weitere technische Dinge (wie einen ESB), durchaus Sinn machen.



**Abb. 10-4**  
Punkt-zu-Punkt-Verbindung

Interessanter wird es da schon, wenn wir jetzt in die weltweite Realität gehen – die des World Wide Web. Hier kann die Nachricht über die verschiedensten Kanäle laufen, muss wahrscheinlich zwischengespeichert werden, passiert Kontrollpunkte (Firewalls) oder ist mit den verschiedensten Metadaten und Policies zur Verarbeitung versehen. Allen diesen Überlegungen ist gemein, dass sich das Ziel der Nachricht nicht mehr auf direktem Weg finden lässt oder erreichbar ist (Abbildung 10-5).



**Abb. 10-5**  
Beliebiger Nachrichtenpfad

Bevor bei der Adressierung von Nachrichten eine Menge der verschiedensten Standards und Mechanismen entstehen konnte, hat das *World Wide Web Consortium* (W3C) mit *WS-Addressing* [58] eine gemeinsame, herstellerübergreifende Lösung geschaffen.

### 10.3.2 Spezifikation

Mit der entsprechenden Ausrüstung an modernen Werkzeugen, die auch hier im Buch vorgestellt werden, ist die Kenntnis von WS-Addressing empfehlenswert, jedoch ist ein tiefer Einstieg in die Spezifikation nicht immer notwendig. Das soll aber nicht bedeuten, dass wir sofort zum nächsten Kapitel übergehen können. Manchmal ist es schon günstig, den Kopf einer Nachricht interpretieren zu können.

Die WS-Addressing-Spezifikation besteht im Wesentlichen aus drei Teilen:

#### ■ Core [59]

In dieser Teilspezifikation wird abstrakt ein Satz von Eigenschaften und XML-Infoset-Elementen definiert, die eine Adressierung von Nachrichten zwischen Sender und Empfänger ermöglichen. Dabei werden zwei austauschbare Konstruktionen beschrieben, die normalerweise in Nachrichtensystemen und in Transportprotokollen Verwendung finden.

Das erste Konstrukt bezeichnet die Endpunkt-Referenz und unterstützt die dynamische Erzeugung von Dienstadressen, die Beschreibung von Dienstinstanzen in einer statusbehafteten Kommunikation und den flexiblen Austausch von Informationen zum Endpunkt. Im Kopf einer SOAP-Nachricht ist jede Endpunkt-Referenz durch einen eigenen Eintrag vertreten. Die Endpunkt-Referenz ist wie folgt spezifiziert:

**Listing 10.3**  
Endpunkt-Referenz

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceParameters>xs:any*
    </wsa:ReferenceParameters> ?
  <wsa:Metadata>xs:any*</wsa:Metadata>?
</wsa:EndpointReference>
```

Das zweite Konstrukt sind die *Message Address Properties*. Diese werden direkt in den Kopfteil der Nachricht geschrieben und sind wie folgt spezifiziert:

**Listing 10.4**  
Message Address  
Properties

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

- **SOAP Binding [61]**

Dieses Dokument beschreibt, wie die abstrakten Eigenschaften aus dem Core-Dokument mit dem SOAP-(1.1, 1.2)-Protokoll abzubilden sind.

- **Metadata [60]**

Dieses Dokument beschreibt, wie die abstrakten Eigenschaften aus dem Core-Dokument mit WSDL abzubilden sind. Auch wird hier beschrieben, wie WSDL-Metadaten in der Endpunkt-Referenz abzulegen sind und wie WS-Policy innerhalb der WSDL für Adressangaben zu nutzen ist. Das folgende Listing zeigt einen Ausschnitt aus einer WSDL-Datei, in der mittels Policies angezeigt ist, dass WS-Addressing zu verwenden ist.

```
<wsp:Policy>
  <wsam:Addressing>
    <wsp:Policy/>
  </wsam:Addressing>
</wsp:Policy>
```

**Listing 10.5**  
WS-Addressing  
erzwingen

### 10.3.3 Adressangaben in unserem Beispiel

Ausgerüstet mit allem Wissen zur Adressierung von Nachrichten, können wir jetzt unseren ATM-Dienst genauer betrachten. Während der Anforderung von Theaterkarten ist eine erste Nachricht (SOAP 1.2) mit Adressangaben zu sehen (siehe Listing 10.6). Deutlich zu erkennen ist der Namensraum (<http://www.w3.org/2005/08/addressing>) sowie die Pflichtangabe Action. Zusätzlich befinden sich weitere optionale Informationen im Kopfteil, wie auch die ReplyTo-Angabe – der Rückweg zu unserem Dienstenutzer.

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">
      http://localhost:8080/ATM/ATM
    </To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">
      http://www.fi.de/atm/ATMServicePort/bookTicketRequest
    </Action>
    <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
      <Address>
        http://www.w3.org/2005/08/addressing/anonymous
      </Address>
    </ReplyTo>
```

**Listing 10.6**  
Nachricht (Anfrage)

```

    <MessageID
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:01d5b786-4ab3-4292-b79f-c49ebf169dcc
    </MessageID>
  </S:Header>

  <S:Body>
    <ns2:ATMOrder
      xmlns:ns2="http://www.fi.de/atm/types">
      . . .
    </ns2:ATMOrder>
  </S:Body>
</S:Envelope>

```

Nicht weiter verwunderlich ist, dass aus der ReplyTo-Information der gesendeten Nachricht die To-Information zur Antwortnachricht wird. Natürlich darf auch die Aktion nicht fehlen. Das folgende Listing zeigt einen Ausschnitt aus der Antwort, die Herr Kainer als Ergebnis seiner Anfrage erhält.

**Listing 10.7**  
Nachricht (Antwort)

```

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">
      http://www.w3.org/2005/08/addressing/anonymous
    </To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">
      http://www.fi.de/atm/ATMServicePort/bookTicketResponse
    </Action>
    <MessageID
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:128fc015-3b04-4782-a317-6ed3b26980dc
    </MessageID>
    <RelatesTo
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:01d5b786-4ab3-4292-b79f-c49ebf169dcc
    </RelatesTo>
  </S:Header>

  <S:Body>
    <ns2:ATMOrderConfirmation
      xmlns:ns2="http://www.fi.de/atm/types">
      . . .
    </ns2:ATMOrderConfirmation>
  </S:Body>
</S:Envelope>

```

## 10.4 Bootstrapping, Metadaten und Transfer

Die Bootstrapping-Phase ist durch den Austausch von Informationen zum Dienst während der Entwicklung gekennzeichnet. Gewöhnlich sind damit die Beschreibung (WSDL) sowie die Eigenschaften (Policies) gemeint – also kurz gesagt alle Informationen, die wir benötigen, um einen Dienstenutzer zu entwickeln. Bisher haben wir dazu die Syntax `<service_url>/<service>?wsdl` genutzt. Diese Möglichkeit hat sich zwar in den letzten Jahren zu einem Industriestandard entwickelt (wird also von der überwältigenden Mehrheit der Systeme unterstützt), ist jedoch genau genommen eine vereinfachte Syntax.

Die Standardisierung des Bootstrapping wird durch die *WS-MetadataExchange*-Spezifikation beschrieben. Da hiermit die auszuführenden Aktionen beschrieben werden, ist noch die Spezifikation des dahinter liegenden Transports notwendig – WS-Transfer.

Bitte seien Sie nicht allzu überrascht, wenn die beiden Spezifikationen (und deren Implementierung) in den Dokumenten und Produkten nicht beschrieben werden. Normalerweise arbeiten diese vollständig im Hintergrund und werden eher selten »an der Oberfläche« zu sehen sein.

»unsichtbare  
Spezifikationen«

### 10.4.1 Austausch von Metadaten: WS-Transfer

*WS-Transfer* [55] beschreibt einen Mechanismus, um eine XML-basierte Repräsentation einer Entität (zum Beispiel des Dienstes) über die Web-Service-Infrastruktur auszutauschen. Betrachtet man die damit beschriebenen Nachrichten genauer, ist eine Analogie zum HTTP-Protokoll erkennbar. Mithilfe der WS-Transfer-Spezifikation wird die Abhängigkeit zu Standards außerhalb unserer Web-Service-Architektur minimiert und gleichzeitig das vorhandene SOAP-basierte Protokoll weiter genutzt.

Die folgenden Ressource- und Ressource-Factory-Operationen sind definiert:

- **Get (Resource-Operation)**  
Die Get-Operation holt eine zum Zeitpunkt des Aufrufes gültige Repräsentation der Ressource.
- **Put (Ressource-Operation)**  
Die Put-Operation erneuert die Ressource. Das geschieht durch das Übermitteln einer neuen Repräsentation.
- **Delete (Ressource-Operation)**  
Mittels dieser Operation wird die Ressource komplett gelöscht.

- **Create (Ressource-Factory-Operation)**

Die Create-Operation wird genutzt, um eine neue Ressource anzulegen und eine initiale Repräsentation bereitzustellen.

Ein gutes Beispiel für einen Get-Transfer ist die Ermittlung der Metadaten (WSDL und Policies) eines Dienstes. Das folgende Beispiel zeigt die Anforderung derselben:

**Listing 10.8**

Anforderung der Metadaten

```
<s11:Envelope
  xmlns:s11='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:wsa10='http://www.w3.org/2005/08/addressing'>
  <s11:Header>
    <wsa10:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa10:Action>
    <wsa10:To>
      http://localhost:8080/ATM/metadata
    </wsa10:To>
    . . .
  </s11:Header>
  <s11:Body />
</s11:Envelope>
```

## 10.4.2 Austausch der Metadaten: WS-MetadataExchange

Web-Service-Metadaten beschreiben, was ein Dienstenutzer wissen muss, um mit einem Dienst zu interagieren. *WS-MetadataExchange* [46] definiert, wie diese Metadaten zwischen zwei Parteien ausgetauscht werden. Die übermittelten Informationen sind im Detail:

- *Policies*: Eigenschaften, Fähigkeiten und Charakteristik eines Dienstes
- *WSDL*: Operation, Schnittstelle und Endpunkt eines Dienstes (siehe Kapitel 7)
- *SOAP*: Struktur und Aufbau der Nachrichten und Datentypen (siehe Kapitel 6)

Die *WS-MetadataExchange*-Spezifikation beschreibt also, wie ein Dienst als Ressource für seine Eigenschaften zu betrachten ist. Dazu wird die *WS-Transfer*-Spezifikation als Grundlage genutzt. Innerhalb von *WS-MetadataExchange* wird aber lediglich das Ermitteln der Dienst-Metadaten beschrieben. Die Spezifikation definiert somit kein Protokoll, um andere Informationen des Dienstes zu ermitteln (wie zum Beispiel Attribute, Status und Eigenschaften).

Wenn wir jetzt für unseren Herrn Kainer einen Dienstenutzer erstellen möchten, könnte man eine Anfrage mittels Metadata-Exchange an die bekannte Adresse des Dienstes stellen. Beachten Sie, dass keine Angabe der Form `?wsdl` mehr benötigt wird.

```
<s11:Envelope
  xmlns:s11='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:wsa10='http://www.w3.org/2005/08/addressing'>
  <s11:Header>
    <wsa10:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa10:Action>
    <wsa10:To>
      http://localhost:8080/ATM/metadata</wsa10:To>
    . . .
  </s11:Header>
  <s11:Body />
</s11:Envelope>
```

**Listing 10.9**

Anforderung der  
Metadaten: Anfrage

Als Antwort erhalten wir die komplette Beschreibung des Dienstes und seiner Eigenschaften:

```
<s11:Envelope
  xmlns:s11='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:wsa10='http://www.w3.org/2005/08/addressing'
  xmlns:mex='http://schemas.xmlsoap.org/ws/2004/09/mex'
  xmlns:wsp='http://schemas.xmlsoap.org/ws/2004/09/policy'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'>
  <s11:Header>
    <wsa10:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa10:Action>
    . . .
  </s11:Header>
  <s11:Body>
  <mex:Metadata>
    <mex:MetadataSection Dialect='http://schemas.xmlsoap.org/wsdl/'>
      <wsdl:definitions
        xmlns:soap12='http://schemas.xmlsoap.org/wsdl/soap12/'
        xmlns:tns='http://www.fi.de/atm'
        xmlns:atm-types='http://www.fi.de/atm/types'
        xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
        xmlns:xsd='http://www.w3.org/2001/XMLSchema'
        xmlns:wsp='http://schemas.xmlsoap.org/ws/2004/09/policy'
        xmlns:wsu='http://docs.oasis-open.org/wss/2004/01
          /oasis-200401-wss-wssecurity-utility-1.0.xsd'
        xmlns:wsam='http://www.w3.org/2007/05/addressing/metadata'>
```

**Listing 10.10**

Anforderung der  
Metadaten: Antwort

```
name="ATMService"  
targetNamespace="http://www.fi.de/atm">  
  
<wsp:UsingPolicy />  
. . .
```

Sehr schön ist das übertragene WSDL zu sehen. Das Besondere daran ist, dass wir nicht mehr wissen müssen, wo die WSDL eigentlich zu finden ist – im Gegensatz zu der bisher angewandten Variante `<service?wsdl`. Ein Web-Service-Stack hat somit wesentlich mehr Freiheiten, zum Beispiel zur Berücksichtigung von dazwischen liegenden Repositories. Sowohl *Metro* auf der Java-Seite als auch .NET aufseiten von Microsoft unterstützen den Austausch der Metadaten gemäß *WS-Transfer* und *WS-MetadataExchange*.

## 10.5 Web Service Interoperability Technologies (WSIT) und Metro

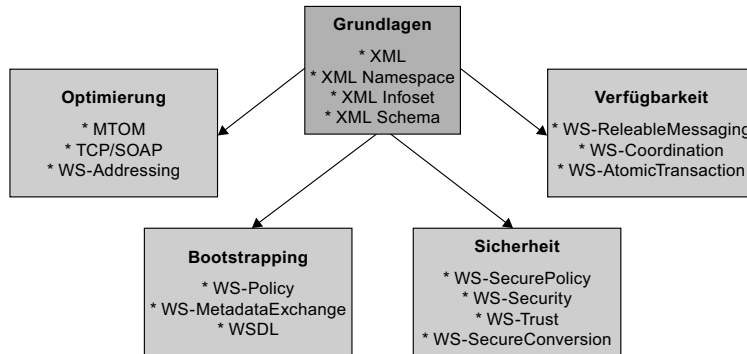
Die *Web Service Interoperability Technologies*, kurz WSIT, sind derjenige Teil von Metro, der sich aktiv mit der Interoperabilität von Diensten zwischen der Java- und der .NET-Welt beschäftigt. Früher getrennt (manch einer erinnert sich bestimmt noch an das Projekt *Tango*), ist es heute Bestandteil des *Metro-Web-Service-Stack* (ab Version 1.3). Um diese Interoperabilität auf höchstem Niveau zu gewährleisten, arbeiten Sun und Microsoft sehr eng zusammen (siehe [66]).

### 10.5.1 Bestandteile

Ausgehend von den in der Java Runtime Edition bereits vorhandenen APIs, erweitert WSIT diese um neue Features für die Arbeit mit Diensten. Die Abbildung 10-6 zeigt einen Überblick über die durch das WSIT-Projekt implementierten, unterstützten und getesteten Standards (angelehnt an Quelle [21]; Stand Februar 2009).

#### Bootstrapping and Configuration

*Bootstrapping and Configuration* bezeichnet die Phase in der Entwicklung (in unserer Definition), die sich mit dem Erzeugen eines Dienstnutzers beschäftigt. Dazu wird die URL des Dienstes genutzt, um die WSDL des Dienstes zu holen. Woher die Dienst-URL kommt, ist nicht Bestandteil der Spezifikation. Diese könnte zum Beispiel über ein Repository bezogen werden.



**Abb. 10-6**  
Bestandteile des  
WSIT-Projekts

Der Prozess des Bootstrappings und der Konfiguration besteht aus den folgenden Schritten:

1. Voraussetzung: Die URL für den Dienst ist vorhanden.
2. Mithilfe der URL und des `wsimport`-Werkzeugs (oder unseres Ant-Tasks) wird eine *WS-MetadataExchange*-Anfrage zum Dienst gesendet. Als Antwort bekommen wir die Beschreibung und die Eigenschaften (Policies) des Dienstes (Sicherheit, Verfügbarkeit, Transaktionen etc.) zurück.
3. Mithilfe der WSDL ist es jetzt möglich, einen Dienstenutzer aufzubauen.
4. Ergebnis: Der Dienstenutzer kann auf den Dienst zugreifen und diesen nutzen.

Für weiterführende Informationen siehe Abschnitte 10.4.1 und 10.3 sowie Kapitel 6 und 7.

### Message Optimization Technology

Dienste tauschen Nachrichten unter sich oder mit dem Dienstenutzer aus. Dabei reichen die Nachrichten von einfachen Anfragen bis hin zu großen binär kodierten Dokumenten. Eine einfache Anfrage ist zum Beispiel das Buchen von Theaterkarten. Die Nachrichten sind einfach zu kodieren, und es ist sicherlich wenig Anstrengung erforderlich, um die Übertragung zu optimieren. Ganz anders schaut es da schon bei der Anforderung eines Kreditwunsches aus. Herr Kainer, als moderner Mensch unserer Zeit, nutzt den ATM und das Portal der Sparkasse häufiger. Infolgedessen wird er auch einen Kredit per Portal beantragen. Dieses Dokument kann sehr groß sein und wird schlimmstenfalls durch einen binären Standard repräsentiert. Das Dokument, eingefügt in die Nachricht, führt unweigerlich zu einer sehr großen SOAP-Nachricht. In dieser müssten zudem die binären Daten noch als Text kodiert sein.

1-KB-Hint

Schlussendlich ist die Performance der Anwendung und des Netzwerks negativ betroffen. Im schlimmsten Fall geht das so weit, dass unsere Anwendung nicht mehr nutzbar wird (es sei denn, Ihre Zielgruppe sind sehr geduldige Zeitgenossen). Als Lösung bietet es sich an, solche Nachrichten immer in der optimalen Form zu kodieren. Das kann zum Beispiel bedeuten, große binäre Dateien getrennt von der eigentlichen Nachricht zu übertragen. WSIT bietet hierzu eine Lösung. Sun gibt die Empfehlung [19], die Optimierung von Nachrichten ab einer Größe von 1 KB zu nutzen.

Weiterführende Informationen zum Thema *Message Optimization* finden Sie in Kapitel 11.

### Reliable Messaging Technology

Verlässliche Dienste (oder genauer gesagt die verlässliche Auslieferung von Nachrichten) sind eine der Grundlagen moderner serviceorientierter Konzepte. Darunter ist der garantierte Transport einer Nachricht von Punkt A zu Punkt B zu verstehen.

Die *Reliable Messaging Technology* sichert uns zu, dass Nachrichten garantiert einmal ausgeliefert werden. Und nur einmal. Im Weiteren können wir die Reihenfolge der Nachrichten in einer Sequenz bestimmen, die es einzuhalten gilt. Sollte wirklich einmal eine Nachricht verloren gehen, oder die Reihenfolge beeinträchtigt sein, wird das zugrunde liegende System den Ursprung wieder herstellen können.

Speicherplatz

Ein Hinweis sei jedoch gestattet. *Reliable Message Technology* bedeutet das Speichern von Nachrichten, bis ein eindeutiges OK vom Empfänger das Eintreffen bestätigt. Es wird also mehr Speicherplatz und unter Umständen mehr Rechenzeit benötigt. Dienste, die *Reliable Messaging* nutzen, können nicht mit Dienstenutzern interagieren, die dieses Prinzip nicht kennen.

Für weiterführende Informationen siehe Kapitel 12.

### Security Technology

Der *Metro-Web-Service-Stack* implementiert zusätzlich zur Sicherheit auf der Transportebene (SSL) den *WS-Security*-Standard. Weitere sicherheitsbezogene Standards von Metro beinhalten:

- **Web Services Trust**

Mit diesem Standard können Anwendungen Sicherheits-Token anfordern und somit eine vertrauenswürdige Verbindung zwischen Dienst und Dienstenutzer aufbauen.

- **WS-Secure Conversation**

Dieser Standard ermöglicht es, einen verteilten Sicherheitskontext

zwischen Dienst und Dienstenutzer herzustellen, wenn die erste Nachricht ausgetauscht wurde. Weitere Nachrichten nutzen dann in der Session abgeleitete Schlüssel, was die Sicherheit erhöht und gleichzeitig die Verarbeitungsgeschwindigkeit steigert.

Neben den grundlegenden sicherheitsrelevanten Standards ist die *Web Services Security Policy* vorhanden und implementiert. Das bedeutet aber auch, dass gesicherte Dienste klar ihre Fähigkeiten und Anforderungen in Bezug auf die Sicherheit formulieren und in der WSDL ablegen können. Diese Dienste sind somit kompatibel zu anderen Diensten und Dienstenutzern, die die Policies verstehen (Java Metro oder auch .NET).

Für weiterführende Informationen siehe Kapitel 13.

### 10.5.2 Ein illustrierendes Beispiel

Die Grundlagen klingen ja vielversprechend. Zeit also, die Interoperabilität auszuprobieren. Als Erstes wagen wir den Versuch, die Optimierungstechnologien hinzuzuschalten. Dabei gehen wir aber nicht weiter ins Detail, sondern beschränken uns auf die Nutzung. Die genauen Eigenschaften und Erklärungen finden Sie im Kapitel 11.

*Beispiel-Workspace  
»interop«*

#### Einschalten der Optimierung

Wir wählen die Optimierung *MTOM* (Message Transfer Optimization) und den TCP-Transport. Bei Letzterem wird die SOAP-Nachricht nicht über HTTP übertragen, sondern über TCP. Diese Technik steigert die Übertragungsgeschwindigkeit beachtlich, da eine gesamte Protokollebene ignoriert bzw. ausgelassen wird und damit nicht interpretiert werden muss.

*MTOM und  
TCP-Transport  
einschalten*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:tns="http://www.fi.de/atm"
  xmlns:atm-types="http://www.fi.de/atm/types"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  name="ATMService"
  targetNamespace="http://www.fi.de/atm">
```

**Listing 10.11**  
*Neue Eigenschaften für  
den Dienst*

```

<wsp:UsingPolicy />
<wsp:Policy wsu:Id="ATMPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsam:Addressing wsp:Optional="false"/>
      <ns1:OptimizedTCPTransport
        xmlns:ns1=
          "http://java.sun.com/xml/ns/wsit/2006/09/
            policy/soaptcp/service"
        enabled="true" />
      <ns2:OptimizedMimeSerialization
        xmlns:ns2=
          "http://schemas.xmlsoap.org/ws/2004/09/
            policy/optimizedmimeserialization"
      />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
...

<wsdl:binding name="ATMServiceSOAP" type="tns:ATMServicePort">
  <wsp:PolicyReference URI="#ATMPortBindingPolicy" />
  ...
</wsdl:binding>
...
</definitions>

```

### Sun Java und unser ATM-Dienst

Beispiel »interop/ATM«

Der Dienst ist bis auf die WSDL identisch mit dem Beispiel aus dem *Enterprise Service Bus*-Kapitel (siehe Kapitel 9). Lediglich die Eigenschaften für den TCP-Transport und die Nachrichtenoptimierung werden in der Beschreibung des Dienstes hinzugefügt. Vergessen Sie bitte nicht, das Projekt noch einmal zum ESB zu deployen.

Beispiel »interop/  
ATMJSPWebClient«

Auch der Dienstenutzer ist bis auf die neuen Eigenschaften identisch mit der vorherigen Version. Wir nutzen die gleiche JSP-Seite mit dem dort enthaltenen Aufruf der Dienstoperation zum Bestellen der Theaterkarten. Wir können diesen also wie gewohnt ausführen. Wenn Sie sich jetzt schon fragen, ob das Ganze nur ein Spaß war, können wir dies glücklicherweise mit »Nein« beantworten. Unser Herr Kainer hat gerade die Mächtigkeit der Policies erlebt (wobei er das ja eigentlich nicht mitbekommt). Neue Eigenschaften des Dienstes werden deklarativ hinzugefügt. Aber jetzt nichts wie ans Ausprobieren (Abbildung 10-7)!

**Abb. 10-7**  
Java/JSP-Dienstenutzer

Als Ergebnis bekommen wir die Bestätigung, wie in Abbildung 10-8 dargestellt.

**Abb. 10-8**  
Antwort vom  
ATM-Dienst

Natürlich sollten wir uns schon fragen, was denn eigentlich hinter den Kulissen passiert ist. Dazu nehmen wir einmal die Eigenschaft TCP-Transport heraus. Damit wollten wir ja die Leistung der Anwendung steigern. Betrachtet man jetzt einmal die transportierten Daten aus dem ESB-Kapitel (siehe 9) kann man sehr gut die übertragenen Daten sowie die Verwendung von SOAP über HTTP erkennen. Sicherlich – bei einer kleinen Nachricht ist das noch nicht sehr viel, aber die Nachrichten werden garantiert umfangreicher.

```
INFO: ---[HTTP response 200]---
INFO: <?xml version="1.0" ?>
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">
      http://www.w3.org/2005/08/addressing/anonymous
    </To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">
      http://www.fi.de/atm/ATMServicePort/bookTicketResponse
    </Action>
    <MessageID
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:128fc015-3b04-4782-a317-6ed3b26980dc
    </MessageID>
```

**Listing 10.12**  
Ohne MTOM und  
TCP-Transport

```

<RelatesTo
  xmlns="http://www.w3.org/2005/08/addressing">
  uuid:01d5b786-4ab3-4292-b79f-c49ebf169dcc
</RelatesTo>
</S:Header>
<S:Body>
  <ns2:ATMOrderConfirmation
    xmlns:ns2="http://www.fi.de/atm/types">
    . . .
  </ns2:ATMOrderConfirmation>
</S:Body>
</S:Envelope>

```

Nach dem Einschalten der neuen Eigenschaften verändert sich auch die Übertragung der Nachricht. Im Log sind jetzt Informationen zum Transport hinzugekommen, darunter auch das Content-Transfer-Encoding: binary.

**Listing 10.13**  
 Mit MTOM und  
 TCP-Transport

```

INFO: ---[HTTP response 200]---
INFO: --uuid:0546ebb2-cb6b-4b91-b89a-afac8b281788
Content-Id: <rootpart*0546ebb2-cb6b-4b91-b89a-afac8b281788@
example.jaxws.sun.com>
Content-Type: application/xop+xml;charset=utf-8;
type="application/soap+xml"
Content-Transfer-Encoding: binary
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">
      http://www.w3.org/2005/08/addressing/anonymous
    </To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">
      http://www.fi.de/atm/ATMServicePort/bookTicketResponse
    </Action>
    <MessageID
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:128fc015-3b04-4782-a317-6ed3b26980dc
    </MessageID>
    <RelatesTo
      xmlns="http://www.w3.org/2005/08/addressing">
      uuid:01d5b786-4ab3-4292-b79f-c49ebf169dcc
    </RelatesTo>
  </S:Header>
  <S:Body>
    <ns2:ATMOrderConfirmation
      xmlns:ns2="http://www.fi.de/atm/types">
      . . .

```

```

</ns2:ATMOrderConfirmation>
</S:Body>
</S:Envelope>
--uuid:0546ebb2-cb6b-4b91-b89a-afac8b281788--

```

Alles in allem haben wir jetzt den ersten Schritt zu einer optimierten Übertragung getan. Unsere Nachricht wird mit der besten Transportmöglichkeit zwischen Dienst und Dienstenutzer ausgetauscht.

### Microsoft .NET und unser ATM-Service

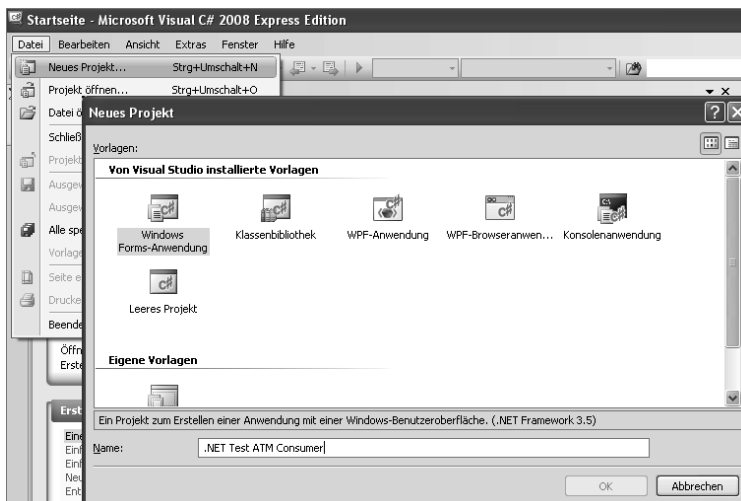
Mit viel positiver Erfahrung aus dem Java-Projekt als Rückenwind lässt Herr Kainer nun eine andere Sprache, ein anderes System als Dienstenutzer ausprobieren. Und was wäre da besser geeignet als Microsoft .NET und C#? Bevor Sie jetzt aber flammende und entrüstete E-Mails an die Autoren schreiben (bezüglich unserer Wahl) – natürlich können Sie auch jedes beliebige andere System nutzen, hauptsächlich die *Web Service Policies* werden verstanden.

Der Dienst an sich wurde im letzten Abschnitt entwickelt, konfiguriert und ist hoffentlich noch gestartet. Wenn nicht, gehen Sie wieder in das Java-Projekt und starten Sie den Enterprise Service Bus. Dadurch wird gleichzeitig auch unser ATM-Dienst gestartet.

Als Erstes werden wir die Benutzerschnittstelle des ATM-Dienstenutzers schreiben. Dazu legen wir ein neues Windows-Application-Projekt an (siehe Abbildung 10-9).

Beispiel »interop/ATM«

Beispiel »atm.net«

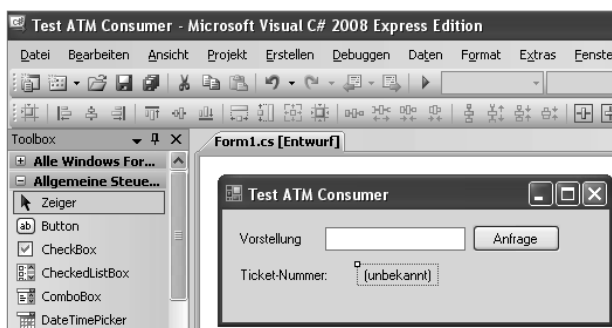


**Abb. 10-9**

Neues  
Windows-Projekt

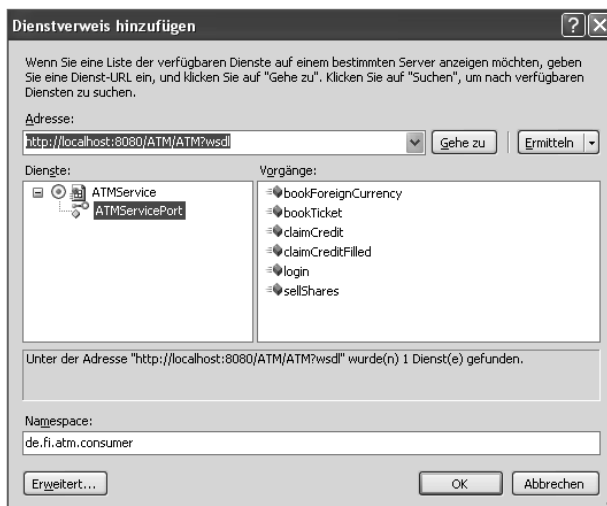
In diesem neuen Projekt entwerfen wir eine Minimalbenutzeroberfläche. Diese besteht lediglich aus einer Eingabezeile für den Namen der gewünschten Theatervorstellung (an dieser Stelle lassen wir auch alle weiteren Informationen weg), dem Button zum Absenden der Anfrage und einem Label, in dem Sitzplatz und Reihe angezeigt werden. Sie sehen schon, eine sehr minimalistische Variante, aber Herr Kainer (und wir auch!) möchte ja auch nur das prinzipielle Funktionieren überprüfen. In Abbildung 10-10 ist die Benutzeroberfläche im Designer dargestellt.

**Abb. 10-10**  
Die Benutzeroberfläche  
im Visual Studio  
Designer



Nachdem die Benutzeroberfläche angelegt ist, kann der Dienst-Stub importiert werden. Natürlich ist damit eine Beschreibung des Dienstes und der .NET-C#-Stub für den Zugriff gemeint. Dazu öffnen Sie den Wizard für den Dienstverweis unter *Projekt->Rechte Maustaste ->Dienstverweis hinzufügen...* Es sollte jetzt der Konfigurationsdialog für einen Dienstverweis zu sehen sein (Abbildung 10-11).

**Abb. 10-11**  
Dienstverweis  
hinzufügen



Jetzt wird es richtig interessant: Wir stellen uns die Frage, was mit den Eigenschaften des Dienstes eigentlich passiert ist. Richtig: Damit alles funktioniert, müssen auf beiden Seiten die richtigen kompatiblen Handler und APIs vorhanden, installiert und aktiviert sein. Dazu kann man sich am besten die generierte `app.config` anschauen. Die Datei ist als Ergebnis eines internen WSDL-zu-C#-Übersetzungsprozesses entstanden. Wir haben ja schon das Ant-Skript `wsd12java.xml` kennen gelernt. Im Visual Studio passiert etwas Ähnliches, nur sehr viel stärker integriert, und `app.config` ist neben den C-Klassen ein weiteres Ergebnis der WSDL-Analyse.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  ...
  <binding name="ATMServiceSOAP">
    <!-- WsdImporter ermittelte unerkannte Richtlinienassertionen
         in ServiceDescription "http://www.fi.com/atm": -->
    <!-- <wsdl:binding name='ATMServiceSOAP'> -->
    <!-- <ns1:OptimizedTCPTransport
         xmlns:ns1="http://java.sun.com/xml/ns/...
         </ns1:OptimizedTCPTransport> -->
    <mtomMessageEncoding maxReadPoolSize="64"
      maxWritePoolSize="16" messageVersion="Soap11"
      maxBufferSize="65536" writeEncoding="utf-8">
      <readerQuotas maxDepth="32"
        maxStringContentLength="8192" maxArrayLength="16384"
        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
    </mtomMessageEncoding>
    <httpTransport manualAddressing="false"
      maxBufferPoolSize="524288"
      maxReceivedMessageSize="65536" allowCookies="false"
      authenticationScheme="Anonymous" bypassProxyOnLocal="false"
      hostNameComparisonMode="StrongWildcard"
      keepAliveEnabled="true"
      maxBufferSize="65536" proxyAuthenticationScheme="Anonymous"
      realm="" transferMode="Buffered"
      unsafeConnectionNtlmAuthentication="false"
      useDefaultWebProxy="true" />
  </binding>
  ...
</configuration>
```

**Listing 10.14***Erkannte Eigenschaften*

Wir lesen uns die Datei natürlich interessiert durch und bekommen dabei den ersten Schreck. Was ist eigentlich mit dem TCP-Transport geschehen? Die Lösung ist natürlich recht einfach. Wie der Namespace schon aussagt (`xmlns:ns1="http://java.sun.com/..."`), handelt es sich

*Der erste Schreck*

hierbei um eine Erweiterung der Eigenschaften eines Dienstes von Sun. Diese ist in der von uns benutzten Version von .NET noch nicht bekannt und kann demnach auch nicht automatisch behandelt werden. Im Gegensatz dazu sind die MTOM-Eigenschaften sehr wohl standardisiert und damit im Handling unproblematisch anzuwenden.

*Warum wir uns  
dennoch freuen*

Damit kommen wir wieder zurück zu den Eigenschaften (Policies) von Diensten. Heute sind schon die wichtigsten davon für alle standardisiert, weitere kommen in Zukunft dazu. Gehen Sie gedanklich einfach mal ein paar Jahre zurück: Ohne Policies müssten Sie jetzt alles von Hand konfigurieren ...

Was bedeutet das aber jetzt für uns?

- Dienst – ist kein Problem: Binding, Ports und Datentypen sind standardisiert und wurden erkannt.
- Eigenschaft MTOM – ist kein Problem: Policies sind standardisiert und wurden erkannt.
- Eigenschaft TCP-Transport – wurde nicht erkannt. Im Fall der TCP-Übertragung können wir uns jedoch auf den internen Mechanismus der verwendeten Service-Engines berufen. Da beide mit diesem Fall umgehen können, stellt dies auch kein Problem dar. Das Vorgehen bei solchen nicht standardisierten Eigenschaften und die daraus resultierenden Auswirkungen können meistens der Dokumentation entnommen werden. Nicht zuletzt ist auch die Suche in Newsgroups oder das Konsultieren eines Experten hilfreich. Seien Sie jedoch im Allgemeinen vorsichtig bei unbekanntem Policies – es könnte zu Problemen kommen. Diese Warnung gilt für alle Service-Engines.

*Vorsicht vor  
unbekannten Policies*

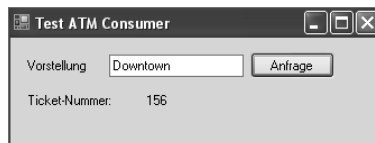
Aber zurück zum Beispiel. Nach dem ersten Schrecken kommen wir zur Implementierung der Funktion für das Bestellen der Theaterkarten.

**Listing 10.15**  
*ATM-Dienst nutzen*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Test_ATM_Consumer.de.fi.atm.consumer;
namespace Test_ATM_Consumer
{
    public partial class Form1 : Form
```

```
{
    public Form1()
    {
        InitializeComponent();
    }
    private void btnSend_Click(object sender, EventArgs e)
    {
        ATMServicePortClient cl = new ATMServicePortClient();
        ATMOrderType orders = new ATMOrderType();
        PersonType[] persons = new PersonType[1];
        persons[0] = new PersonType();
        persons[0].lastname = "Kainer";
        orders.People = persons;
        ATMOrderConfirmation[] cons = cl.bookTicket(orders);
        lblNo.Text = cons[0].ticketno;
    }
}
```

Bleibt am Ende noch das Ausprobieren übrig. Wenn Sie das Beispiel starten, ist der Dialog aus Abbildung 10-12 zu sehen. Im Textfeld tippen wir die gewünschte Vorstellung ein und bekommen im Anschluss die Ticketnummer angezeigt.



**Abb. 10-12**  
Dienstnutzer  
ausführen

### 10.5.3 Ausblick

Ein Ausblick in die Zukunft der Nutzung und Weiterentwicklung der Metadaten fällt leicht, oder doch nicht? Auf jeden Fall bewegen wir uns hier etwas in die spekulative Zone. Es ist richtig, dass der Einsatz von Policies das Leben des Entwicklers bei der Erstellung von Diensten wesentlich vereinfacht. Im Kapitel zur Web-Service-Sicherheit werden Sie die Möglichkeit zur Definition von Eigenschaften schätzen lernen. Probieren Sie es ruhig mit einem .NET-C#-Dienstnutzer einfach mal aus. Auf der anderen Seite ist es natürlich den Firmen überlassen, Standards zu setzen und zu implementieren. Seit .NET 3.0 und Java 5 arbeiten zumindestens Sun und Microsoft gemeinsam an der Interoperabilität. Weitere Web-Service-Engine-Hersteller sind gefolgt. Um bei Java zu bleiben, kann man ruhigen Gewissens *Apache CXF* und *Apache Axis 2* dazu zählen. Das hat auch die enorm positive Eigenschaft,

dass innerhalb der Java-Welt (!) nun auch eine gewisse Austauschbarkeit von Diensten gewährleistet ist.

*Policies einsetzen!*

Zusammenfassend kann man sagen, dass dem Einsatz von Policies nichts im Wege steht.

## 10.6 Zusammenfassung

In diesem Kapitel haben Sie erfahren:

- Metadaten und Eigenschaften (Policies) können in der WSDL eines Dienstes beschrieben werden.
- Die Eigenschaften reichen von einfachen Dingen (wie der Transportoptimierung) bis hin zu komplexen Szenarien (wie der Sicherheit).
- Die wichtigsten Eigenschaften sind schon standardisiert und somit auf den meisten modernen Web-Service-Engines verfügbar.
- Web-Service-Engines können diese Eigenschaften für die Erkennung von speziellen Fähigkeiten nutzen. Dazu haben wir als Beispiel einen Java-Dienstnutzer entwickelt.
- Werkzeuge wie *Microsoft Visual Studio*, können diese Eigenschaften aus der WSDL auslesen und für die Konfiguration nutzen. Dazu haben wir als Beispiel einen .NET-C#-Dienstnutzer entwickelt.
- *WS-Metadata* und *WS-Transfer* bilden die Grundlage für den Austausch von Metainformationen. Beide Spezifikationen sind normalerweise unsichtbar für uns.
- *WS-Addressing* wird für das Routing der Nachrichten vom Sender zum Empfänger und zurück genutzt. Es ist jedoch nicht immer notwendig, diese Angaben in den Kopfteil einer SOAP-Nachricht aufzunehmen (besonders bei Punkt-zu-Punkt-Verbindungen).

## 10.7 Übungsaufgaben

1. Beschreiben Sie kurz, was WS-Addressing ist.
2. Welche Bedeutung hat der WS-Policy-Standard?
3. Welche hauptsächlichen Bereiche werden durch den Interoperabilitätsstandard (WSIT) innerhalb von Metro berücksichtigt?
4. Welche Vorgehensstrategie wählen Sie, wenn eine Policy nicht von einem Teilsystem unterstützt wird?