

## SAP-Schnittstellenprogrammierung

» Hier geht's  
direkt  
zum Buch

# DIE LESEPROBE

# Kapitel 8

## SOAP-Webservices

*In diesem Kapitel zeigen wir Ihnen, wie Sie auf der ABAP-Plattform schnell und einfach SOAP-Clients und SOAP-Server in ABAP und Java entwickeln können. Ebenso lernen Sie, wie Sie die SOAP-Services aus C# nutzen.*

SOAP, ursprünglich als Akronym für *Simple Object Access Protocol* stehend, ist seit Version 1.2 kein Akronym mehr und wird nur noch als SOAP bezeichnet. SOAP hat sich innerhalb sehr kurzer Zeit als wichtiger Standard der Interprozesskommunikation durchgesetzt und ist auch heute noch, über 20 Jahre nach Veröffentlichung der Version 1.2, von Bedeutung. Das liegt vor allem an seiner hohen Plattformunabhängigkeit und der Nutzung von Internetstandards: Die Nachrichten zwischen zwei Prozessen werden als XML-Nachrichten transportiert, und der Transport erfolgt in der Regel über HTTP. Sowohl XML als auch HTTP sind plattformunabhängig und werden von zahlreichen Plattformen unterstützt, so auch von der ABAP-Plattform.

In diesem Kapitel zeigen wir Ihnen mit vielen Programmierbeispielen, wie SOAP-Komponenten, die in unterschiedlichen Programmiersprachen entwickelt wurden, Nachrichten miteinander austauschen können. Als etablierte und standardisierte Technologie wird SOAP häufig für die Integration mit älteren SAP-Systemen oder bestehenden Enterprise-Anwendungen genutzt.

### 8.1 Inside-Out-Webservices und -Webclients mit der ABAP-Plattform

Für die Entwicklung von Webservices und Webclients bietet die ABAP Workbench einen Wizard, der Sie in wenigen Schritten schnell zum Ziel führt. Zuerst zeigen wir Ihnen in diesem Abschnitt die Entwicklung eines ABAP-Webservice, den Sie danach aus einem ABAP-Client aufrufen können. Dabei gehen wir nach dem *Inside-Out-Ansatz* vor (auch als *Code-First-Ansatz* bekannt). Bei diesem Ansatz existiert bereits ein Business-Objekt mit BAPIs oder eine Funktionsgruppe mit Funktionsbausteinen, die wir anschließend als Webservice veröffentlichen möchten. Den *Outside-In-Ansatz*

(auch bekannt als *Contract-First-Ansatz*), bei dem wir zuerst den Webservice beschreiben und dann die Funktionalität in ABAP bereitstellen, erläutern wir in Abschnitt 8.2, »Outside-In-Webservices und -Webclients mit der ABAP-Plattform«.

### 8.1.1 SOAP-Webservice mit ABAP

Es ist einfach, auf der ABAP-Plattform einen SOAP-Webservice zu entwickeln. Als Beispiel erstellen wir in diesem Abschnitt einen Webservice, der die Funktionalität des Business-Objekts *Order* aus Abschnitt 1.4, »Überblick über die Schnittstellentechnologien von SAP«, per SOAP bereitstellt. Mit diesem Webservice können Sie Bestellungen anlegen, ändern, suchen und lesen. Wir verwenden dazu Transaktion SE80 anstelle der ABAP Development Tools (ADT) in Eclipse, weil Letztere nur den Outside-In-Ansatz unterstützen und dieser Ansatz zur Anbindung eines Legacy-Systems in der Praxis selten verfolgt wird. Wir werden dabei eine WSDL-Datei (Web Service Description Language) anlegen.

#### Neun Schritte zum Webservice

Starten Sie daher Transaktion SE80, den Object Navigator. Lassen Sie sich das Paket anzeigen, das Ihren Webservice enthalten soll. Im Kontextmenü des Pakets starten Sie über **Anlegen • Enterprise Service** den Webservice-Wizard, der Sie in den folgenden neun Schritten zum fertigen Webservice führt:

#### 1. Objekttyp

Im ersten Schritt zeigt Ihnen der Wizard eine Übersicht über die Objekttypen an, die Sie anlegen können. Dort haben Sie unter anderem die Auswahl zwischen **Service-Provider** (Webservice) und **Service-Consumer** (Webservice-Client). Hier wählen Sie **Service-Provider** als Objekttyp und klicken dann den Button **Weiter** an, um zum nächsten Bildschirm zu gelangen.

#### 2. Art des Service-Providers

Im nächsten Schritt wählen Sie den Typ des zu generierenden Service-Providers aus. Sie haben die Wahl zwischen **Backend**, **Enterprise Services Repository**, **Vorhandenes ABAP-Objekt (Inside Out)** und **Externe(s) WSDL/Schema**. Wählen Sie hier **Vorhandenes ABAP-Objekt (Inside Out)**, und klicken Sie dann auf den Button **Weiter**.

#### 3. Service

Im Eingabefeld **Service-Definition** geben Sie einen Namen für den Webservice ein, z. B. »ZIFP\_ORDER\_SERVICE«. Im Eingabefeld **Beschreibung** können Sie eine Kurzbeschreibung eingeben.

#### 4. Endpunkttyp

Im vierten Schritt wählen Sie den Endpunkttyp aus. Hier haben Sie die Wahl zwischen **Funktionsgruppe**, **Funktionsbaustein** und **BAPI**. Für dieses Beispiel wählen Sie den Eintrag **BAPI** aus.

#### 5. Endpunkt-BAPI

Je nach Endpunkttyp, den Sie im vorangegangenen Schritt ausgewählt haben, geben Sie hier den Namen eines Funktionsbausteins, einer Funktionsgruppe oder eines Business-Objekts an. Für dieses Beispiel wählen Sie das BAPI (eigentlich Business-Objekt) *Order*, das Sie in Abschnitt 3.5, »Business-Objekte und BAPIs«, innerhalb der Anwendung *Z* angelegt haben. Wählen Sie **Namen zuordnen**, um die Namen der Funktionsbausteine zuzuordnen. Anfangsbuchstaben werden großgeschrieben und Unterstriche entfernt.

#### 6. BAPI

In diesem Schritt wählen Sie die Operationen, die Ihr Webservice anbieten soll. Der Schritt wird nur bei den Endpunkttypen **Funktionsgruppe** und **BAPI** angezeigt, da eine Funktionsgruppe mehrere Funktionsbausteine bzw. ein Business-Objekt mehrere BAPIs enthalten kann. In diesem Beispiel sollen alle vier BAPIs auch als Operationen im Webservice angeboten werden. Zusätzlich legen Sie über den Button **BAPI Commit/Rollback** fest, dass auch die Methoden *TransactionCommit* und *TransactionRollback* des Business-Objekts *BapiService* als Operationen in den Webservice aufgenommen werden sollen. Abbildung 8.1 zeigt diesen Schritt.

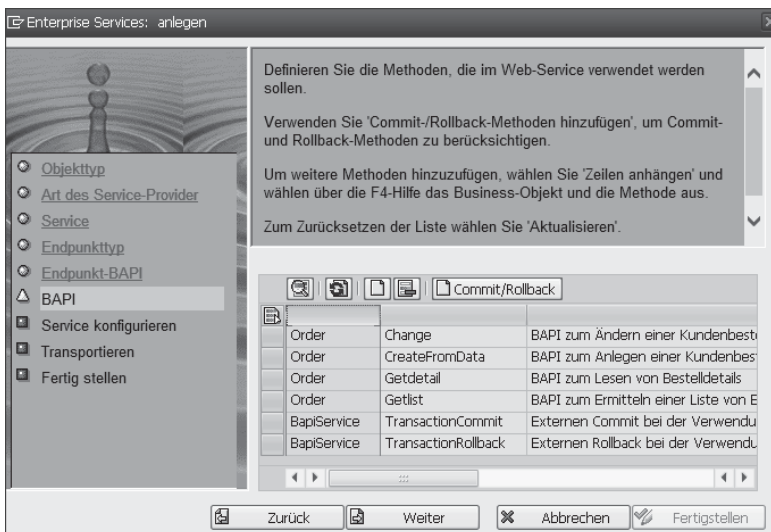


Abbildung 8.1 Operationen des Webservice wählen

7. Service konfigurieren

In diesem Schritt wählen Sie ein Profil für die Sicherheitseinstellungen aus, das Ihren Anforderungen entspricht. In der Drop-down-Liste **Profil** haben Sie die Wahl zwischen vier verschiedenen Profilen:

- Authentifizierung mit Zertifikaten und Transportgarantie
- Authentifizierung mit Benutzer und Kennwort, keine Transportgarantie
- Authentifizierung mit Benutzer, Kennwort und Transportgarantie
- keine Authentifizierung und keine Transportgarantie

8. Transportieren

Im vorletzten Schritt geben Sie ein Paket und einen Entwicklungsauftrag an.

9. Fertigstellen

Im letzten Schritt wird Ihnen angezeigt, dass Ihr Webservice angelegt werden wird. Klicken Sie auf den Button **Fertigstellen**.

Schnittstelle eines Webservice

Abbildung 8.2 zeigt die externe Sicht auf die Schnittstelle des Webservice. Sie sehen, dass für jedes BAPI eine Operation erzeugt wurde.

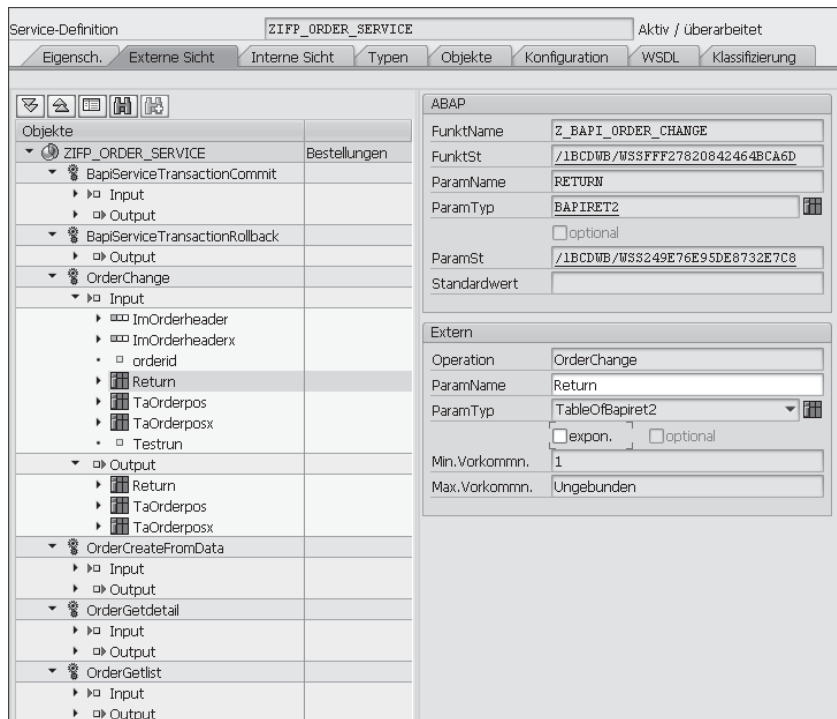


Abbildung 8.2 Serviceschnittstelle des Webservice

Jede Operation hat einen Importparameter namens `Input` und einen Exportparameter namens `Output`. Im Importparameter finden Sie Entsprechungen zu den Import- und Tabellenparametern des Funktionsbausteins, im Exportparameter Entsprechungen zu dessen Export- und Tabellenparametern.

Dass die Tabellenparameter zweimal erscheinen, liegt daran, dass sie bei Funktionsbausteinen sowohl als Import- als auch als Exportparameter dienen können. Wenn Sie in der Strukturübersicht der Serviceschnittstelle auf einen solchen Parameter klicken, sehen Sie rechts die Zuordnung des Parameters von **ABAP** (oben) zum entsprechenden Parameter in der Sicht **Extern** (unten). Über die Auswahl **expon.** können Sie festlegen, ob der Parameter überhaupt in der Serviceschnittstelle erscheinen soll. In Abbildung 8.2 haben wir z. B. den Tabellenparameter `Return` aus dem Input-Parameter herausgenommen, da dieser Tabellenparameter definitiv ein Rückgabeparameter ist.

Wenn Sie mögen, können Sie an dieser Stelle den Webservice aktivieren und anschließend mit der Funktionstaste **F8** testen. Im Testdialog wählen Sie zuerst die Operation aus. Danach wird Ihnen ein Template für ein XML-Request-Dokument generiert, das Sie auch noch editieren können, bevor Sie es ebenfalls mit der Taste **F8** an den Webservice schicken. Wurde die Methode erfolgreich ausgeführt, zeigt Ihnen die Testumgebung das XML-Response-Dokument an.

Webservice testen

Damit Sie den Service nutzen können, müssen Sie noch einen *Endpunkt* anlegen. Dazu wechseln Sie in die Leseansicht und starten den SOA-Manager über die Tastenkombination **Strg** + **F10**. Der SOA-Manager ist eine WebDynpro-Anwendung für das SOA-Management. Klicken Sie auf **Service anlegen**. Es startet ein Wizard, der Sie in vier Schritten durch die Konfiguration des Endpunkts führt.

SOA-Manager

### 1. Service- und Binding-Namen

Geben Sie hier einen beliebigen Service- und Binding-Namen ein. Dies sind zwei Schlüsselwerte, die Ihre Endpunktkonfiguration identifizieren. Klicken Sie danach auf **Weiter**.

### 2. Provider-Sicherheit

Hier können Sie unter verschiedenen Transportgarantie- und Authentifizierungseinstellungen wählen und diese konfigurieren. Für unser einfaches Beispiel wählen Sie **Benutzer-ID und Kennwort** im Bereich **Authentifizierung auf Transportebene**. Klicken Sie dann auf **Weiter**.

### 3. SOAP-Protokoll

Hier können Sie eine alternative Zugriffs-URL angeben und den Typ des

*Identifiable Business Context* auswählen. Identifiable Business Contexts sind SAP-proprietäre Konzepte zur Identifizierung der an einem Integrationszenario beteiligten Sender und Empfänger. Für unser Beispiel geben wir hier nichts ein und klicken auf **Weiter**.

4. **Operationseinstellungen**

Hier werden nochmals alle Operationen aufgelistet. An dieser Stelle klicken Sie auf **Fertig**. Es wird dann die Liste der Konfigurationen wie in Abbildung 8.3 angezeigt.

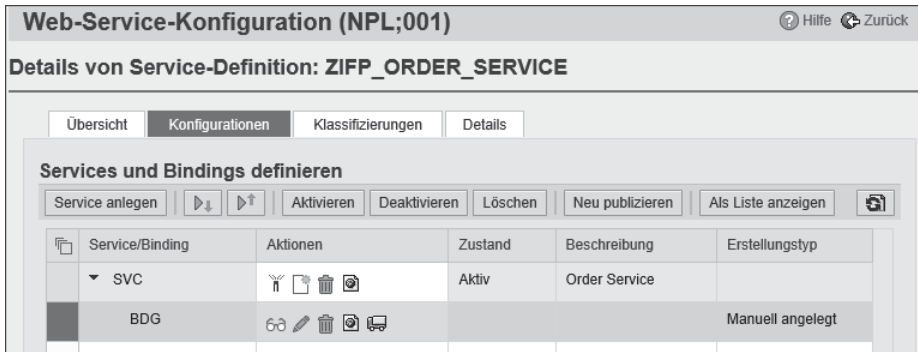


Abbildung 8.3 SOA-Manager

**WSDL-Generierung**

Wenn Sie im SOA-Manager für den Service auf das Icon (**WSDL-Generierung für Binding öffnen**) klicken, öffnet sich ein Dialog, in dem Sie verschiedene Optionen für die Generierung einer WSDL-Datei (WSDL = Web Services Description Language) wählen können. Jede Kombination von Einstellungen, die Sie in den Drop-down-Listen im oberen Bereich wählen können, definiert ein *Flavor*, das Sie unter einem frei wählbaren Namen speichern können. Dieses Flavor wird auch in der URL der WSDL-Datei als hexadezimale ID benutzt. Ändern sich die Einstellungen, ändert sich auch die URL. Die gültige WSDL-Datei kann über den Dialog direkt heruntergeladen werden.

**WSDL-Format/  
Bindungsstil wählen**

Schließt man den SOA-Manager, kann man auf der Registerkarte **WSDL** ebenso eine WSDL-Datei samt URL einsehen. Über das Drop-down-Menü **SOAP-Style** können Sie zwischen **RPC** und **Dokument** wählen (siehe Abbildung 8.4). Der Unterschied zwischen den WSDL-Formaten bzw. *Bindungsstilen* RPC und Dokument liegt in ihrer Struktur und ihrem Verwendungszweck. Die an dieser Stelle heruntergeladene WSDL-Datei hat jedoch kein `<wsdl:service>`-Element, definiert also keinen Endpunkt, entspricht aber dem zuvor eingestellten Flavor.

### WSDL-Generierung für Binding: BDG ☰

---

#### WSDL-Flavors

Flavor: Einfach 🗑️

SAP-Assertions:	<span style="border: 1px solid gray; padding: 2px;">Ohne</span>	WSP-Version:	<span style="border: 1px solid gray; padding: 2px;">Keine Policy</span>	SOAP-Action:	<span style="border: 1px solid gray; padding: 2px;">Ohne</span>
Security-Assertions:	<span style="border: 1px solid gray; padding: 2px;">Standard</span>	WSP-Style:	<span style="border: 1px solid gray; padding: 2px;">Einzel-Binding</span>		
WSDL-Section:	<span style="border: 1px solid gray; padding: 2px;">AllInOne</span>	SOAP-Version:	<span style="border: 1px solid gray; padding: 2px;">SOAP 1.2</span>		
WSDL-Version:	1.1	SOAP-Style:	<span style="border: 1px solid gray; padding: 2px;">Dokument</span>		

---

#### Optionen für WSDL-Zugriff und -URLs

Standard

Alternative URL

Alternativer Host:	<input style="width: 90%;" type="text"/>	Alt. Port (http):	<input style="width: 90%;" type="text"/>
Metadatenprotokoll:	<input style="width: 90%;" type="text"/>	Alt. Port (https):	<input style="width: 90%;" type="text"/>

URL-Optionen anwenden

---

#### WSDL-Generierung

🔄 🗑️

WSDL-URL für Binding: http://vhcalnplci:8000/sap/bc/srt/wSDL/fiv\_10200A1012D0/bndg\_url/sap/bc/srt/rfc/sap/zifp\_order\_service/001/svc/bdg?sap-client=001

Abbildung 8.4 WSDL-Generierung

## Bindungsstile

Innerhalb von SOAP werden zwei verschiedene Bindungsstile verwendet, die zwei unterschiedliche Auffassungen des Datenaustauschs (Versenden von Dokumenten vs. Aufruf von Funktionen) reflektieren.

- Im *Dokument-Stil* werden die Input- und Output-Parameter der Operationen als Dokumente betrachtet, die eine beliebige Struktur haben können. Die ausgetauschten Nachrichten werden als nicht mehr weiter unterteilbar betrachtet; sie bestehen demnach aus nur einem Teil:

```
<wsdl:message name="OrderChange">
  <wsdl:part name="parameters" element="tns:OrderChange" />
</wsdl:message>
```

- Beim *RPC-Stil* wird davon ausgegangen, dass die Input- und Output-Parameter in einzelne, benannte Parameter unterteilt werden können. Die ausgetauschten Nachrichten bestehen demnach aus mehreren Teilen:



```

<wsdl:message name="OrderChange">
  <wsdl:part name="ImOrderheader" type="tns:Zifporder" />
  <wsdl:part name="ImOrderheaderx" type="tns:Zifporderx" />
  <wsdl:part name="TaOrderpos"
    type="tns:TableOfZifporderpos" />
  <wsdl:part name="TaOrderposx"
    type="tns:TableOfZifporderposx" />
  <wsdl:part name="Testrun" type="n0:char1" />
  <wsdl:part name="orderid" type="n0:numeric10" />
</wsdl:message>

```

**WSDL-Datei** Grundsätzlich ist eine WSDL-Datei ein XML-Dokument, das den Webservice genau definiert. Es handelt sich dabei also um einen Schnittstellenvertrag. Abbildung 8.5 zeigt Teile der WSDL-Datei für unseren Webservice. Im Element `<wsdl:service>` der WSDL-Datei steht im Kind-Element `<soap:address>` im Attribut `location` der sogenannte *Endpunkt*, also die URL, unter der der Webservice aufgerufen werden kann. Normalerweise schauen Sie sich eine solche WSDL-Datei wahrscheinlich nicht an. Sie benötigen sie hier aber, um den Code für Clients zu generieren, die den Webservice verwenden sollen.

**Aufbau der WSDL-Datei**

Wie Abbildung 8.5 zeigt, besteht die WSDL-Datei aus fünf Teilen:

**1 Definition der einfachen und komplexen Datentypen**

Die vom Webservice verwendeten Datentypen werden im XML-Element `<wsdl:types>` in der *XML Schema Definition Language* (XSD) beschrieben.

**2 Beschreibung der Nachrichtentypen**

Die Beschreibung der Nachrichtentypen erfolgt in den XML-Elementen `<wsdl:message>`. Jede Operation des Webservices hat dabei eine Input- und eine Output-Message, z. B. `OrderChange` und `OrderChangeResponse`. Die Nachrichtendefinitionen beziehen sich auf die vorangegangenen Typdefinitionen in `<wsdl:types>`.

**3 Beschreibung der Operationen**

Innerhalb von `<wsdl:portType>`-Elementen beschreibt die WSDL-Datei, welche Operationen der Webservice zur Verfügung stellt und welche Input- und Output-Message jeweils von einer Operation erwartet und zurückgeschickt wird.

**4 Bindungsstil und SOAP-Action**

Im Abschnitt `<wsdl:binding>` wird definiert, an welche SOAP-Actions die Operationen gebunden sind und welchen Bindungsstil die Nachrichten haben. Die SOAP-Action wird beim Aufruf per HTTP im HTTP-Header mitgegeben und hilft dem Webservice dabei, die richtige Operation aus-

zuwählen. Die Bindungsstile wurden im Hinweiskasten »Bindungsstile« erläutert.

### 5 Servicename und URL


Innerhalb eines `<wsdl:service>`-Elements wird nun der Webservice benannt und die URL angegeben, unter der er per HTTP aufgerufen werden kann (Endpunkt des Webservice).

```

<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions targetNamespace="urn:sap-com:document:sap:soap:functions:mc-style" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="urn:sap-com:document:sap:soap:functions:mc-com:document:sap:rfc:functions">
+ <wsdl:documentation>
+ <wsdl:types> ①
+ <wsdl:message name="BapiServiceTransactionCommit"> ②
+ <wsdl:message name="BapiServiceTransactionCommitResponse">
+ <wsdl:message name="BapiServiceTransactionRollback">
+ <wsdl:message name="BapiServiceTransactionRollbackResponse">
+ <wsdl:message name="OrderChange">
+ <wsdl:message name="OrderChangeResponse">
+ <wsdl:message name="OrderCreateFromData">
+ <wsdl:message name="OrderCreateFromDataResponse">
+ <wsdl:message name="OrderGetdetail">
+ <wsdl:message name="OrderGetdetailResponse">
+ <wsdl:message name="OrderGetlist">
+ <wsdl:message name="OrderGetlistResponse">
- <wsdl:portType name="ZIFP_ORDER_SERVICE"> ③
+ <wsdl:operation name="BapiServiceTransactionCommit">
+ <wsdl:operation name="BapiServiceTransactionRollback">
+ <wsdl:operation name="OrderChange">
- <wsdl:operation name="OrderCreateFromData">
  <wsdl:input message="tns:OrderCreateFromData" />
  <wsdl:output message="tns:OrderCreateFromDataResponse" />
</wsdl:operation>
+ <wsdl:operation name="OrderGetdetail">
+ <wsdl:operation name="OrderGetlist">
</wsdl:portType>
- <wsdl:binding name="BDG_soap12" type="tns:ZIFP_ORDER_SERVICE"> ④
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
+ <wsdl:operation name="BapiServiceTransactionCommit">
+ <wsdl:operation name="BapiServiceTransactionRollback">
+ <wsdl:operation name="OrderChange">
+ <wsdl:operation name="OrderCreateFromData">
+ <wsdl:operation name="OrderGetdetail">
+ <wsdl:operation name="OrderGetlist">
</wsdl:binding>
- <wsdl:service name="SVC"> ⑤
- <wsdl:port name="BDG_soap12" binding="tns:BDG_soap12">
  <soap12:address location="http://vhcalnplci:8000/sap/bc/srt/rfc/sap/zifp_order_service/001/svc/bdg" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Abbildung 8.5 Bestandteile der WSDL-Datei

Obwohl die WSDL-Datei auf den ersten Blick beeindruckend ist, sind die ausgetauschten Dokumente einfach aufgebaut. Wenn Sie im SOA-Manager in der Webservice-Administration auf das Icon  (Web-Service-Navigator für gewähltes Binding öffnen) klicken, öffnet sich ein Webbrowser-Fenster und zeigt eine Java-Anwendung an, die das WSDL-Dokument liest und eine grafische Benutzeroberfläche zum Testen des Webservice erzeugt. Dieses

SOAP-Dokumente

Testwerkzeug zeigt Ihnen auch die XML-Dateien an, die per HTTP-POST ausgetauscht werden. Listing 8.1 zeigt das Request-Dokument.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <!-- ... -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:OrderGetlist xmlns:ns1='urn:sap... '>
      <ImRefid></ImRefid>
    </ns1:OrderGetlist>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

#### Listing 8.1 SOAP-Request

##### SOAP-Envelope

Das Wurzelement ist ein SOAP-Envelope, der einen optionalen SOAP-Header und einen SOAP-Body enthält. Im SOAP-Body sehen Sie die Request-Nachricht `OrderGetList`. Der SOAP-Header enthält SAP-spezifische Informationen über das Management der Sitzung.

Listing 8.2 zeigt die Antwort des Servers. Wiederum ist das Antwortdokument, in diesem Fall `OrderGetlistResponse`, in das SOAP-Body-Element innerhalb des SOAP-Envelope-Elements eingebettet.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Body>
    <n0:OrderGetlistResponse xmlns:n0="urn:sap... ">
      <Return>
        <Type>S</Type>
        <Id></Id>
        <Number>000</Number>
        <Message></Message>
        <LogNo></LogNo>
        <LogMsgNo>000000</LogMsgNo>
        <MessageV1></MessageV1>
        <MessageV2></MessageV2>
        <MessageV3></MessageV3>
        <MessageV4></MessageV4>
```

```

    <Parameter></Parameter>
  <Row>0</Row>
  <Field></Field>
  <System></System>
</Return>
<TaOrders>
  <item>
    <Mandt>001</Mandt>
    <Orderid>0000000040</Orderid>
    <Type>S0</Type>
    <Refid>0000000002</Refid>
    <Buyer>Käufer</Buyer>
    <Seller>Lieferant</Seller>
    <Orderdate>2008-12-19</Orderdate>
  </item>
  <item>
    <Mandt>001</Mandt>
    <Orderid>0000000053</Orderid>
    <Type>S0</Type>
    <Refid>0000000001</Refid>
    <Buyer>Buchhandlung Pehnke</Buyer>
    <Seller>Grosshandel Englbrecht</Seller>
    <Orderdate>2018-01-31</Orderdate>
  </item>
</TaOrders>
</n0:OrderGetlistResponse>
</soap-env:Body>
</soap-env:Envelope>

```

### Listing 8.2 SOAP-Response

Sie können den Exportparameter `Return` und die Tabelle `TaOrders` mit zwei Einträgen erkennen. In der Antwort fehlt der SOAP-Header sogar ganz. Wie Sie im Listing z. B. am Element `Orderid` erkennen können, werden ABAP-Felder vom Typ `N` bzw. Dictionary-Felder vom Typ `NUMC` mit führenden Nullen bis zu deren Gesamtlänge aufgefüllt. In einem Request-Dokument mit derartigen Feldern müssen Sie darauf achten, diese selbst mit führenden Nullen aufzufüllen.

#### 8.1.2 SOAP-Webclient mit ABAP

Einen ABAP-Client zu schreiben, der die Operationen des Webservice aufruft, ist ebenfalls nicht schwierig und in drei Schritten erledigt. Zuerst las-

sen Sie sich eine Proxy-Klasse aus der WSDL-Datei generieren, konfigurieren danach einen logischen Port, der auf den Endpunkt des Webservice zeigt, und schreiben zum Schluss ein Programm, das die Proxy-Klasse instanziiert und deren Methoden verwendet.

Starten Sie dazu Transaktion SE80, den Object Navigator. Lassen Sie sich das Paket anzeigen, das Ihre Proxy-Klasse enthalten soll. Im Kontextmenü des Pakets starten Sie über **Anlegen • Enterprise Service** den Webservice-Wizard, der für Sie die Proxy-Klasse generiert.

**Angabe der WSDL-Datei**

Im ersten Schritt wählen Sie nun **Service-Consumer** als Objekttyp und klicken dann auf den Button **Weiter**, um zum nächsten Bildschirm zu gelangen. Im zweiten Schritt müssen Sie die Herkunft der WSDL-Datei angeben, die den Webservice beschreibt, für den Sie einen Client-Proxy erzeugen möchten. Hier geben Sie dem Wizard an, ob er die WSDL-Datei aus dem Backend, aus dem Enterprise Services Repository oder aus einer externen WSDL-Datei beziehen wollen. Für den vorliegenden Webservice, der in Abschnitt 8.1.1, »SOAP-Webservice mit ABAP«, erzeugt wurde, wählen Sie hier **Externe(s) WSDL/Schema** an und selektieren im nächsten Schritt die Option **URL**. Geben Sie im nächsten Schritt die URL ein, die Sie zuvor aus dem Browser kopiert haben, als Sie sich die WSDL-Datei in Transaktion SOAMANAGER angeschaut haben. Wenn Sie die WSDL-Datei in einer lokalen Datei gespeichert haben, geben Sie diese an.

**Auswahl von Paket und Präfix**

Im nächsten Schritt wählen Sie das Paket aus, in dem die generierten Objekte abgelegt werden sollen. Außerdem geben Sie ein Präfix an, das vor alle generierten Objektnamen gestellt werden soll. Wir haben hier als Paket ZIFP und als Präfix ZIFP\_ gewählt. Möglicherweise erhalten Sie eine Warnung, dass Ihr Paket keine Verwendungserklärung für das Paket SAI\_TOOLS hat. Diese Verwendungserklärung können Sie nachträglich noch einfügen, indem Sie im Object Navigator Ihr Paket öffnen, zur Registerkarte **Verwendungserklärungen** gehen und dort eine neue Verwendungserklärung für das Paket SAI\_TOOLS anlegen. Sie erhalten zudem wahrscheinlich eine Warnung, dass es zu Namenskollisionen und Namensverkürzungen gekommen ist. Sie haben aber die Möglichkeit, die generierten Namen auf der Registerkarte **Namensprobleme** zu ändern, bevor Sie den generierten Proxy aktivieren.

Abbildung 8.6 zeigt den generierten Webservice-Client-Proxy. Sie sehen, dass die generierte Proxy-Klasse ZIFP\_CO\_ZIFP\_ORDER\_SERVICE sechs Methoden hat, eine für jede Operation des Webservice. Jede Operation verfügt über einen Import- und einen Exportparameter. Die einzelnen Komponenten dieser Parameter entsprechen den Import- und Exportparametern des Webservice.

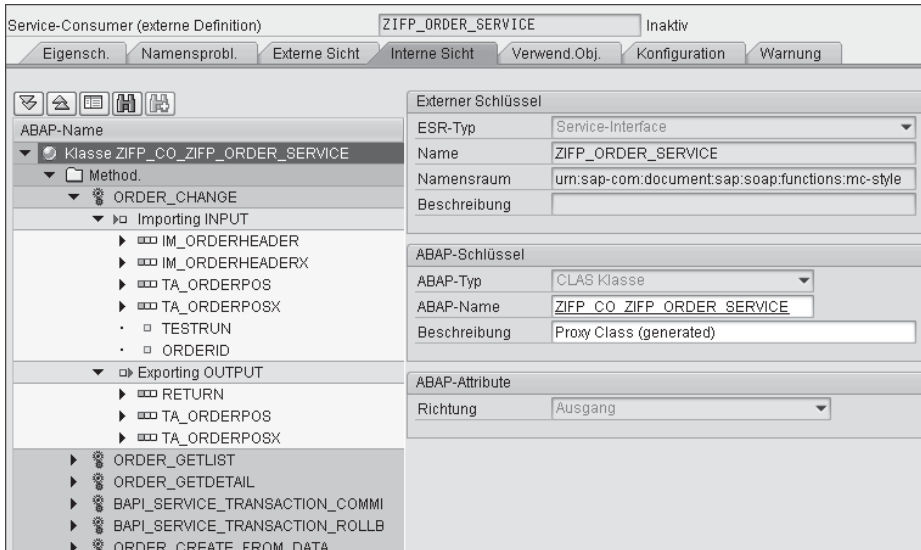



Abbildung 8.6 Generierter Webservice-Client-Proxy

Bevor Sie den Proxy testen und verwenden können, müssen Sie mit Transaktion SOAMANAGER noch einen sogenannten *logischen Port* für die generierte Klasse anlegen. Die generierte Proxy-Klasse enthält nämlich keine Informationen über die URL, über die der Webservice aufgerufen werden kann. Diese Informationen werden in einem logischen Port hinterlegt, der bei der Instanziierung der Proxy-Klasse im Konstruktor mitgegeben wird. Für eine Proxy-Klasse können Sie in Transaktion SOAMANAGER mehrere Ports anlegen. Damit können Sie flexibel auf verschiedene URLs verweisen, bei denen die gleichen Webservices erreichbar sind. Sie können auf die Übergabe eines Ports bei der Instanziierung verzichten, wenn Sie einen der Ports als Default-Port markieren.

#### Logische Ports

Klicken Sie auf das Icon  (**SOAMANAGER für Konfiguration starten**), um den SOA-Manager zu starten. Klicken Sie im SOA-Manager auf **Anlegen**, und wählen Sie **WSDL-basierte Konfiguration** aus. Es erscheint ein Wizard, der Sie in acht Schritten durch die Konfiguration führt, wobei nur die ersten vier Schritte für unser Beispiel relevant sind.

#### 1. Name des logischen Ports

Geben Sie in das Eingabefeld **Name des logischen Ports** einen beliebigen Namen für den logischen Port ein, den Sie erstellen möchten. Markieren Sie außerdem das Auswahlfeld **Logischer Port ist Standard**, um Ihren Port, den Sie gerade erstellen, als Standardport zu markieren. Geben Sie zusätzlich noch eine kurze Beschreibung für den neuen Port ein. Klicken Sie dann auf **Weiter**.

## 2. WSDL-Informationen

Hier haben Sie wieder die Wahl, ob Sie auf die WSDL per HTTP zugreifen oder diese aus einer lokalen Datei auslesen möchten. Wählen Sie die erste Möglichkeit, müssen Sie noch die URL der WSDL sowie Benutzer und Passwort für den WSDL-Zugriff angeben. Bei der zweiten Möglichkeit genügt es, wenn Sie den Dateipfad angeben. Klicken Sie dann auf **Weiter**.

## 3. Binding-Auswahl

Falls die WSDL mehrere Bindings enthält, wählen Sie in diesem Schritt eines der angebotenen Bindings aus und klicken dann auf **Weiter**.

## 4. Consumer-Sicherheit

Geben Sie hier einen Benutzernamen und ein Passwort für den Zugriff auf den Webservice an, und klicken Sie dann auf **Weiter**.

## 5. HTTP-Einstellungen

Hier zeigt der Wizard die URL des Webservices an, und Sie können weitere Angaben zum HTTP-Proxy und zum Binding vornehmen. Klicken Sie anschließend auf **Weiter**.

## 6. SOAP-Protokoll

Hier können Sie Änderungen zu den Details des SOAP-Protokolls vornehmen und mit einem Klick auf **Weiter** zum nächsten Schritt gehen.

## 7. Identifiable Business Context

Hier können Sie Angaben zum Identifiable Business Context machen und mit einem Klick auf **Weiter** zum letzten Schritt gehen.

## 8. Operationseinstellungen

Hier zeigt Ihnen der Wizard nochmals alle Operationen an, und Sie können die Standardeinstellungen zu jeder Operation ändern. Klicken Sie hier auf **Fertig**.

### Test des Proxys

Schließen Sie den Dialog und gehen Sie nun zurück in den Object Navigator, um Ihren Proxy zu testen. Klicken Sie in der Anwendungsfunktionsleiste auf **Testen** (**F8**). Geben Sie den Namen des logischen Ports an, den Sie soeben angelegt haben. Geben Sie ebenfalls eine Operation an, die Sie testen möchten, z. B. ORDER\_GETDETAIL. Wählen Sie **Ausführen** (**F8**), und klicken Sie dann in der Anwendungsfunktionsleiste auf **XML-Editor** (**Strg** + **F2**).

Ersetzen Sie die generierte orderid (Bestellnummer) durch eine orderid, von der Sie wissen, dass sie im Zielsystem existiert, damit Sie auch sinnvolle Daten zurückbekommen. Klicken Sie nochmals auf **Ausführen** (**F8**); dann können Sie sich die Antwortnachricht anzeigen lassen.

Nach dem erfolgreichen Test können Sie nun ein Programm schreiben, das den Webservice aufruft. Listing 8.3 zeigt ein ABAP-Programm, das die Methode `order_getlist` der generierten Proxy-Klasse aufruft und die zurückgegebenen Bestelldetails in einer ABAP-Liste ausgibt.

Client-Programm

```
REPORT zifp_test_webservice.
DATA:
  r_proxy TYPE REF TO zifp_co_zifp_order_service,
  output TYPE zifp_order_getlist_response,
  input TYPE zifp_order_getlist,
  wa TYPE zifp_zifporder.
TRY.
  CREATE OBJECT r_proxy
* EXPORTING
*   LOGICAL_PORT_NAME =

  CATCH cx_ai_system_fault.
ENDTRY.
TRY.
  CALL METHOD r_proxy->order_getlist
    EXPORTING
      input = input
    IMPORTING
      output = output.
  CATCH cx_ai_system_fault.
  CATCH cx_ai_application_fault.
ENDTRY.
LOOP AT output-ta_orders-item INTO wa.
  WRITE:/ wa-orderid, wa-type, wa-refid,
          wa-buyer, wa-seller, wa-orderdate.
ENDLOOP.
```

**Listing 8.3** ABAP-Programm für einen Webservice-Aufruf

Das Programm erzeugt zuerst ein Objekt der Proxy-Klasse. Da die Methode `order_getlist` keine relevanten Input-Parameter hat, müssen diese vor dem Aufruf auch nicht gefüllt werden. Nach dem Aufruf der Methode wird die interne Tabelle `output-ta_orders-item` mit den Bestelldetails ausgegeben.

Sobald Ihnen eine WSDL-Datei vorliegt, können Sie sich einen ABAP-Proxy für den Aufruf des Webservices generieren lassen, der in der WSDL-Datei beschrieben ist. In der Praxis wird es leider nicht möglich sein, den zuvor für den ABAP-Webservice generierten Proxy wiederzuverwenden, da die

**Aufruf eines  
externen  
Webservice**

WSDL-Dateien für den ABAP-Service und für den Java-Service subtile und eventuell weniger subtile Unterschiede aufweisen, obwohl beide Services das Gleiche tun. Zum Beispiel könnte der eine WSDL-Generator den Dokumentenbindungsstil bevorzugen und der andere den RPC-Bindungsstil. Der eine Generator setzt Listen als eine Hierarchie von zwei XML-Elementen um (eine für die Liste mit `maxOccurs=1`; das Kindelement für das Listenelement mit `maxOccurs=unbounded`), und der andere Generator verzichtet auf dieses übergeordnete Listenelement. Der Ausgangspunkt für die Entwicklung war nicht die WSDL-Datei, sondern die WSDL-Datei wurde aus dem ABAP- bzw. Java-Code heraus generiert. Den umgekehrten Ansatz, nämlich zuerst die Schnittstellenbeschreibung zu erstellen und dann einen Server-Proxy aus der Schnittstellenbeschreibung für unterschiedliche Programmiersprachen generieren zu lassen, lernen Sie im nächsten Abschnitt kennen.

## 8.2 Outside-In-Webservices und -Webclients mit der ABAP-Plattform

In diesem Abschnitt erläutern wir die Entwicklung von Webservices und Webservice-Clients nach dem Outside-In-Verfahren (auch als Contract-First-Ansatz bekannt). Dabei beschreiben Sie die Schnittstelle zuerst im Enterprise Services Repository (ESR) oder im Backend Repository und generieren dann daraus Server- und Client-Proxys. Das ESR wird sowohl mit SAP Process Integration (PI, ist eine Komponente von SAP Process Orchestration) als auch direkt mit SAP Process Orchestration ausgeliefert, während das Backend Repository Bestandteil einer jeden ABAP-Plattform ist. Damit Sie im Enterprise Services Repository eine Schnittstelle beschreiben können, müssen Sie diese einer Softwarekomponentenversion zuordnen, die Sie im *System Landscape Directory* (SLD) anlegen und danach in das Enterprise Services Repository importieren müssen. Daher erklären wir Ihnen in den nächsten vier Abschnitten zunächst das System Landscape Directory, danach die Schnittstellen im Enterprise Services Repository, anschließend die Schnittstellen im Backend-Repository und schließlich die Generierung von ABAP-Proxys.

### 8.2.1 System Landscape Directory

Das SLD speichert Informationen zur System- und Softwarelandschaft eines Unternehmens. Diese Informationen werden auch von anderen Komponenten der ABAP-Plattform verwendet, z. B. vom SAP Solution Manager.

Hier beschreiben wir nur die Funktionen, die das Enterprise Services Repository nutzt.

Das SLD speichert Informationen über Softwareprodukte, deren Hersteller und Versionen. Ein Beispiel für die vielen Softwareproduktversionen, die im System Landscape Directory gespeichert sind, ist SAP SCM 7.0. Die Softwareproduktversion SAP SCM 7.0 ist die Version 7.0 des Produkts SAP Supply Chain Management des Herstellers SAP. Zwischen einem Softwareprodukt und seinen Versionen gibt es eine 1:n-Beziehung.

**Softwareprodukte**

Eine Softwareproduktversion umfasst eine oder mehrere Softwarekomponentenversionen. Dabei werden die Softwarekomponenten und ihre Versionen unabhängig von den Softwareprodukten gepflegt. Daher gibt es eine m:n-Beziehung zwischen Softwareproduktversionen und Softwarekomponentenversionen. Oder anders gesagt: Eine Softwarekomponentenversion kann in mehreren Softwareproduktversionen enthalten sein. Zum Beispiel ist die Softwarekomponentenversion SAP\_BASIS 7.5 in vielen Softwareproduktversionen von SAP enthalten, unter anderem in SAP ERP 6.0 EHP8.

**Softwarekomponenten**

Für unser Szenario legen Sie im SLD die Softwareproduktversion SP\_IFP 2.0 mit der einzigen Softwarekomponentenversion SC\_IFP 2.0 an. Starten Sie das SLD im Webbrowser über die URL `http://server:port/sld`, und melden Sie sich dort an. Abbildung 8.7 zeigt den Einstiegsbildschirm nach der erfolgreichen Anmeldung am SLD.

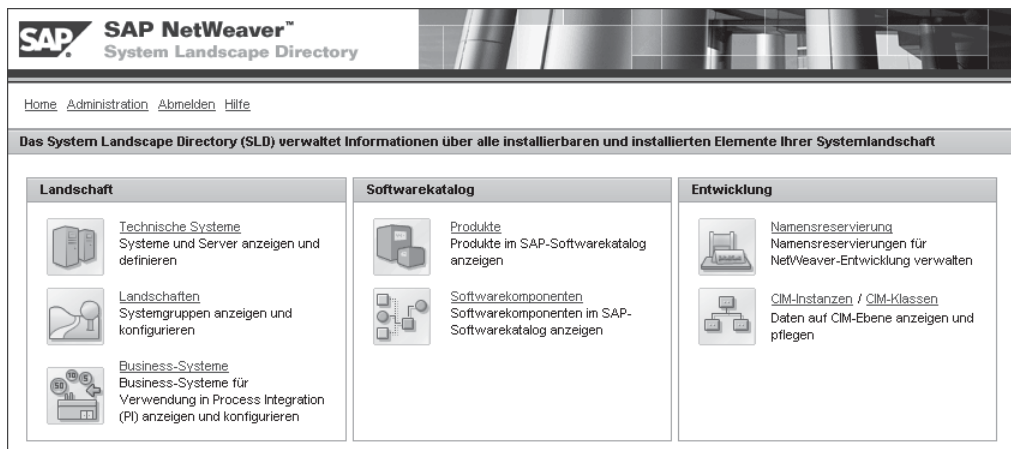


Abbildung 8.7 Einstiegsbildschirm des System Landscape Directorys

Um die Softwarekomponentenversion anzulegen, die Sie nachher im Enterprise Services Repository benötigen, wählen Sie den Link **Produkte** und erzeugen von dort aus zuerst eine neue Softwareproduktversion. Dazu geben Sie den Namen des Herstellers, den Namen des Softwareprodukts und die

Version ein. Anschließend werden Sie aufgefordert, eine Software-Unit und eine Softwarekomponentenversion innerhalb der Produktversion anzulegen.

### 8.2.2 Service-Interfaces

#### Datentypen und Nachrichtentypen

Die Softwarekomponentenversion SC\_IFP 2.0, die Sie zuvor angelegt haben, importieren Sie nun ins Enterprise Services Repository. Danach legen Sie im Enterprise Services Repository die notwendigen Datentypen für die Nachrichten an, die die Operationen der Service-Interfaces austauschen sollen. Dies sind unter anderem Datentypen für Bestellkopf, Bestellposition, Bestellung, Bestellnummer und Bestellreferenz. Als Nachrichtentypen benötigen Sie nun für jede Operation eigene Request- und Response-Nachrichten.

Abbildung 8.8 zeigt als Beispiel den Nachrichtentyp MT\_ORDERCREATE\_REQ. Dieser Nachrichtentyp verweist auf den Datentyp DT\_IFP\_ORDER mit den beiden Komponenten ORDERHEAD und ORDERPOS, die ihrerseits auf die beiden einfacheren Datentypen DT\_IFP\_ORDERHEAD und DT\_IFP\_ORDERPOS verweisen. Links in Abbildung 8.8 sind weitere Datentypen und Nachrichtentypen aufgelistet.

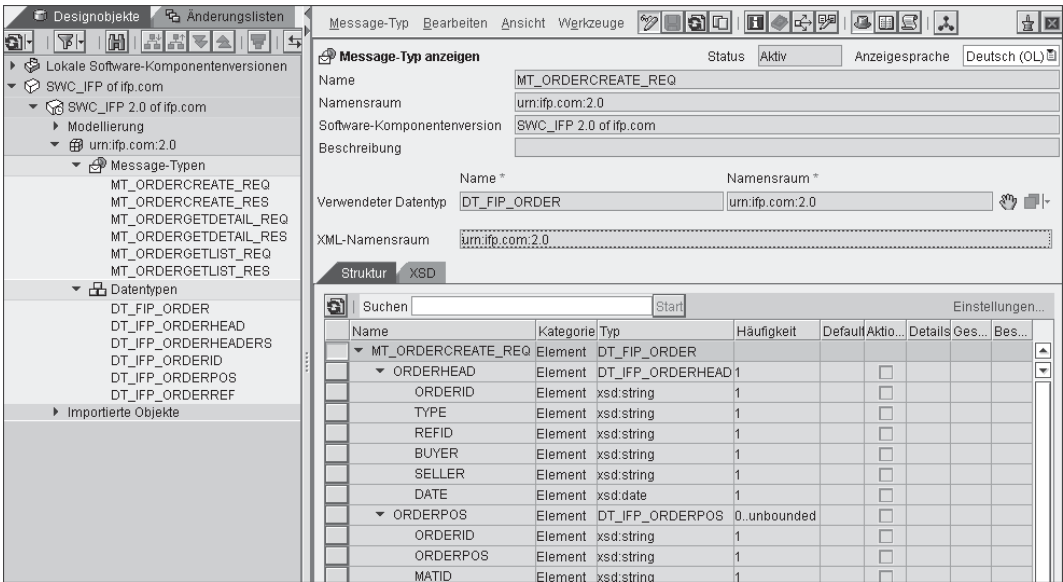


Abbildung 8.8 Nachrichtentyp im Enterprise Services Repository

#### Service-Interfaces

Um die Arbeiten im Enterprise Services Repository abzuschließen, legen Sie die benötigten Service-Interfaces an. Ein Service-Interface eines Senders

(Client) gehört der Kategorie `outbound` an, ein Service-Interface eines Empfängers (Server) der Kategorie `inbound`. Jede einzelne Operation im Service-Interface hat entweder den Modus `synchron` oder `asynchron`.

Ein Service-Interface kann einem von vier Interface-Patterns zugeordnet werden:

**Interface-Patterns**

#### ■ Stateless

Die Operationen dieser Interfaces hinterlassen auf dem Server keinen Zustand, der von anderen Operationen desselben Interface in weiteren Aufrufen desselben Clients verwendet werden kann. Dies ist das bei Weitem wichtigste Interface-Pattern.

#### ■ Stateless (SAP-XI-3.0-kompatibel)

Diese Interfaces wurden in SAP Process Integration 7.0 (jetzt Teil von SAP Process Orchestration) aus Gründen der Rückwärtskompatibilität zu den Message-Interfaces der SAP Exchange Infrastructure 3.0 eingeführt, die es ab SAP Process Integration 7.0 nicht mehr gibt. Sie haben genau eine Operation mit dem gleichen Namen wie das Interface.

#### ■ Tentative Update & Confirm/Compensate (TU&C/C)

Diese Interfaces umfassen vier Arten von Operationen. Tentative-Update-Operationen erzeugen oder ändern Daten auf dem Server vorläufig. Diese Operationen sind synchron. Eine asynchrone `Confirm`-Operation bestätigt die Änderungen und schreibt sie fest. Eine asynchrone `Compensate`-Operation verwirft die Änderungen und macht sie rückgängig.

#### ■ Stateful

Die Operationen dieser Interfaces hinterlassen auf dem Server einen Zustand, der von anderen Operationen desselben Interface in weiteren Aufrufen desselben Clients verwendet werden kann. Dieses Pattern wird nur für ganz spezielle technische Zwecke benötigt.

Für das Beispiel legen Sie die beiden Service-Interfaces `SI_IFP_ORDER_IN` und `SI_IFP_ORDER_OUT` mit den beiden Operationen `GetList` und `GetDetail` an. Abbildung 8.9 zeigt die Operation `GetList` des Interface `SI_IFP_ORDER_IN`. Der Message-Typ für die Request-Nachricht ist `MT_ORDERLIST_REQ`, der Message-Typ für die Response-Nachricht ist `MT_ORDERLIST_RES`. Für die beiden Interfaces werden Sie Server- und Client-Proxys in ABAP erzeugen lassen und diese Proxys verwenden, um Webservices (Provider) und Webservice-Clients (Consumer) zu entwickeln.

**Service-Interfaces anlegen**

Das Service-Interface `SI_IFP_ODERCREATE_IN` illustriert das Interface-Pattern `TU&C/C` (siehe Abbildung 8.10).

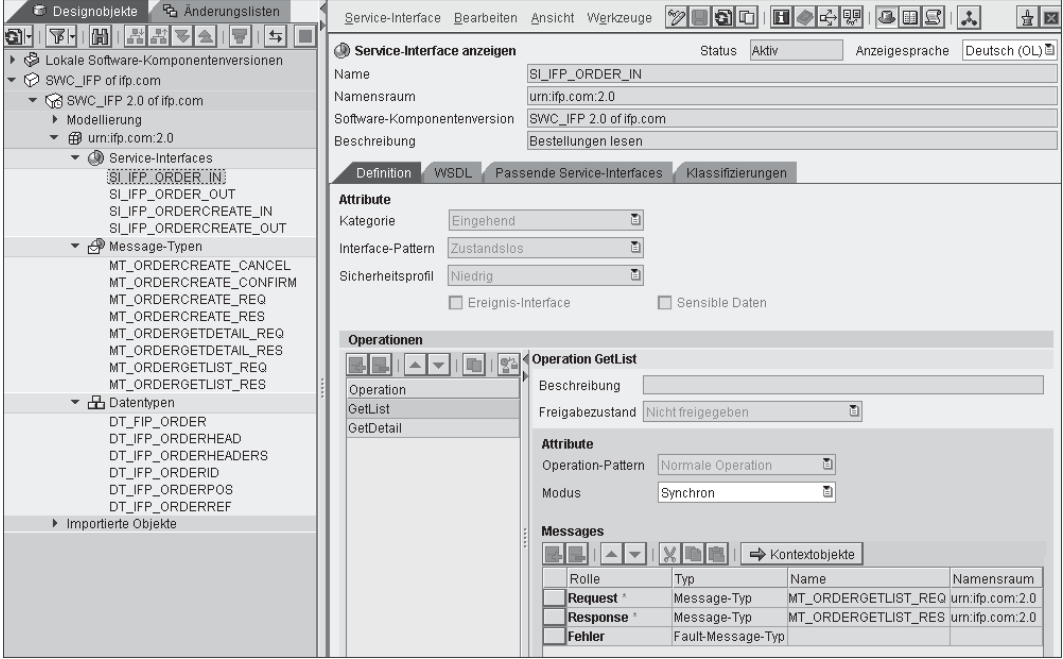


Abbildung 8.9 Service-Interface mit zwei Operationen

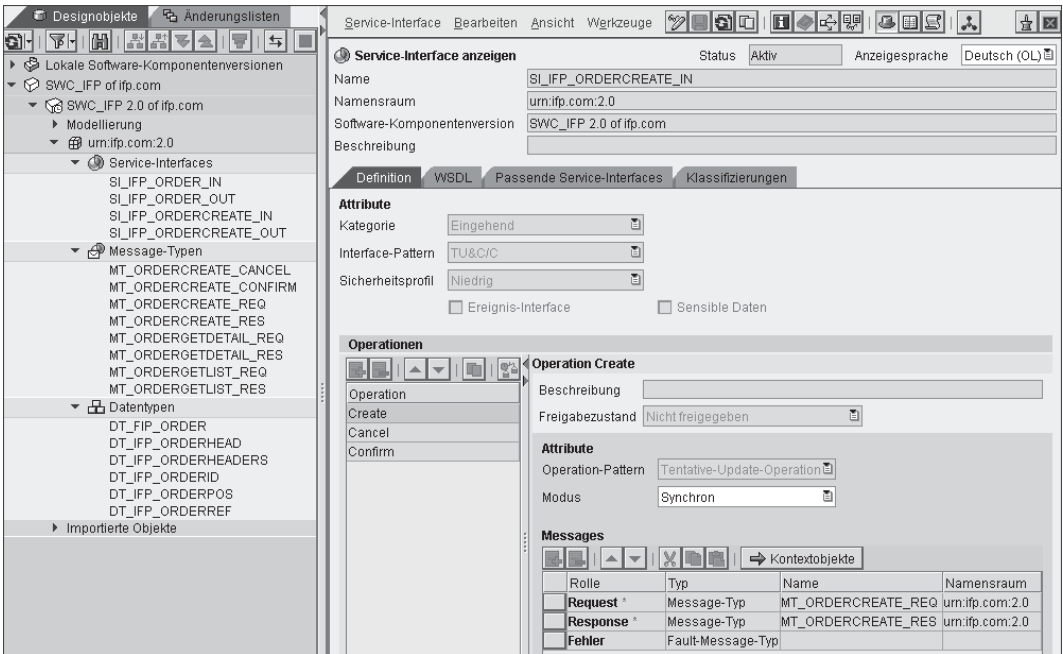


Abbildung 8.10 Service-Interface mit dem Pattern »TU&C/C«

Es verfügt über die synchrone Tentative-Update-Operation `Create`, die eine Bestellung nur vorläufig anlegt. Erst durch die asynchrone `Confirm`-Operation namens `Confirm` wird diese Bestellung bestätigt. Falls der Client, der mit der Tentative-Update-Operation eine vorläufige Bestellung angelegt hat, diese aber doch wieder stornieren möchte, kann er sie mittels der `Compensate`-Operation `Cancel` wieder rückgängig machen.

### 8.2.3 Das Backend Repository

Auch der SAP NetWeaver Application Server ABAP stellt ein Repository für Datentypen, Nachrichtentypen und Service-Interfaces bereit, das sogenannte *Backend Repository*. In diesem Abschnitt beschreiben wir die Möglichkeiten des Backend Repositories anhand eines einfachen Beispiels:

Zuerst legen Sie mit Transaktion SPXNGENAPPL fest, welche Namensräume dem Backend Repository und welche dem Enterprise Services Repository zugeordnet sind. Abbildung 8.11 zeigt beispielsweise, wie der Namensraum `urn:zifp` dem Backend Repository zugeordnet wird. Erst nachdem Sie dem Backend Repository Namensräume zugeordnet haben, können Sie mit dem Enterprise Service Wizard im Object Navigator die weiteren Objekte anlegen.

Namensräume zuordnen

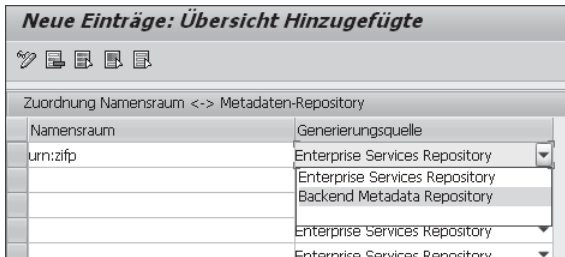


Abbildung 8.11 Namensräume zuordnen

Wir wollen als einfaches Beispiel ein Service-Interface bereitstellen, das arithmetische Operationen zur Verfügung stellt. Dazu definieren wir zuerst zwei Datentypen für die Operatoren und das Ergebnis der Operation. Legen Sie dazu im Object Navigator ein neues Objekt vom Typ *Enterprise Service* an, indem Sie im Kontextmenü Ihres Pakets den Eintrag **Anlegen • Enterprise Service** auswählen. Wählen Sie anschließend im Enterprise Service Wizard den Eintrag **Datentyp** aus, und bestätigen Sie mit **Weiter**. Wählen Sie nun im nächsten Schritt den Eintrag **Backend** aus, und klicken Sie erneut auf **Weiter**. Vergeben Sie im nächsten Schritt einen Namen für den Datentyp (z. B. »ZIFP\_OPERANDS«), wählen Sie den Namensraum `urn:zifp` aus,

Datentypen anlegen

und bestätigen Sie mit **Weiter**. Im letzten Schritt geben Sie Paket, Workbench-Auftrag und Namenspräfix an und wählen **Fertigstellen**.

Der Datentyp wird nun angezeigt; er ist vom XSD-Typ `xsd:string` und vom ABAP-Typ `String`. Sie können daraus einen komplexen Datentyp machen, indem Sie im Kontextmenü des Datentyps den Eintrag **Zu Complex Content Type wechseln** auswählen und dann über das Kontextmenü weitere Elemente oder Attribute hinzufügen. Fügen Sie auf diese Weise zwei Elemente namens `Operand1` und `Operand2` vom XSD-Typ `xsd:int` hinzu, wie es Abbildung 8.12 zeigt. Speichern und aktivieren Sie Ihren Datentypen, und legen Sie auf die gleiche Art einen zweiten Datentyp namens »ZIFP\_RESULT« vom XSD-Typ `xsd:int` an.

**Nachrichtentypen anlegen**

Legen Sie als Nächstes die beiden Nachrichtentypen `ZIFP_ADD_REQ` und `ZIFP_ADD_RES` an, die auf die beiden Datentypen `ZIFP_OPERANDS` und `ZIFP_RESULT` verweisen, die Sie soeben angelegt haben.

Dazu starten Sie wieder den Enterprise Service Wizard und wählen im ersten Schritt den Eintrag **Message-Typ** aus. Im Proxy-Editor des neuen Message-Typs wählen Sie dann im Kontextmenü den Eintrag **Datentyp setzen • Vorhandenen Datentyp auswählen** aus, um den Datentyp zu setzen.

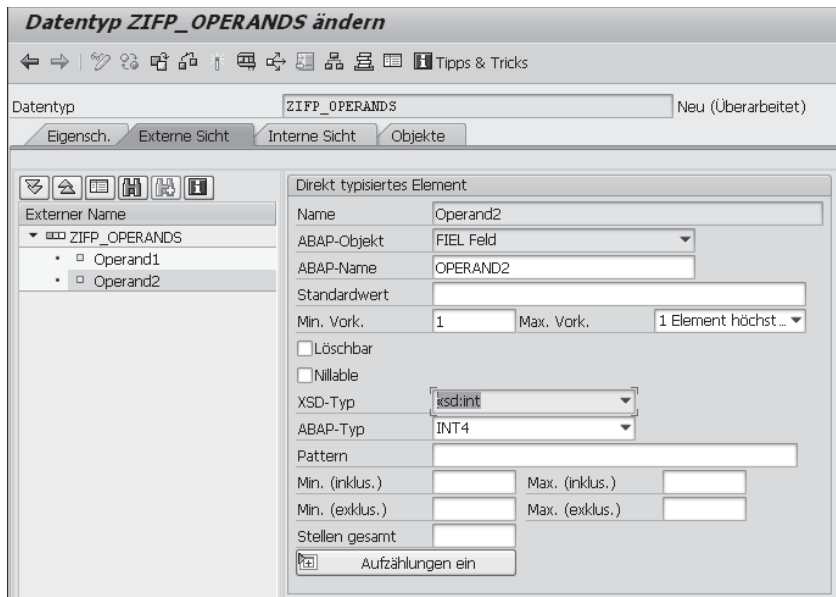


Abbildung 8.12 Datentyp »ZIFP\_OPERANDS«

Abbildung 8.13 zeigt das Ergebnis für den Nachrichtentyp `ZIFP_ADD_REQ`.

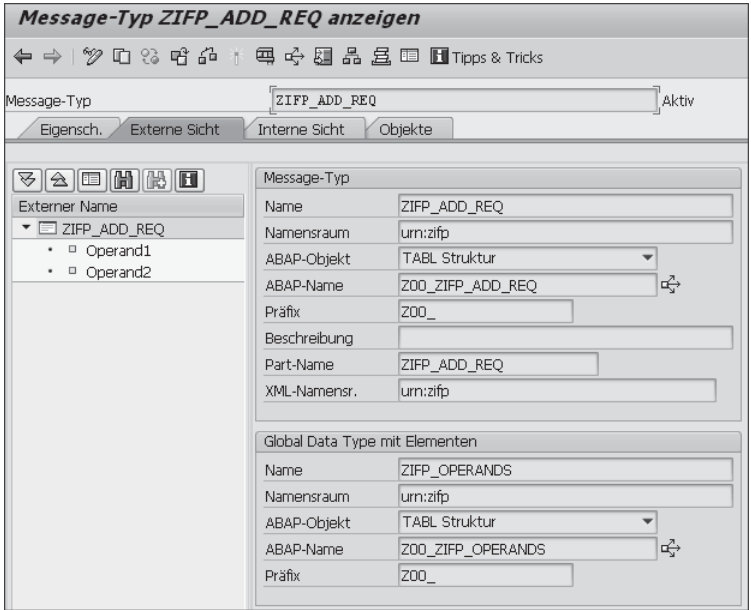


Abbildung 8.13 Nachrichtentyp »ZIFP\_ADD\_REQ«

Nun können Sie einen Service-Provider anlegen. Gehen Sie dazu so vor wie bisher, und wählen Sie im ersten Schritt im Enterprise Service Wizard den Eintrag **Service-Provider**. Vergeben Sie im zweiten Schritt den Namen für den Service-Provider, z. B. »ZIFP\_ARITHMETIC\_SRV«. Im Proxy-Editor wählen Sie im Kontextmenü den Eintrag **Operation hinzufügen** aus, um eine Operation namens »Add« hinzuzufügen. Im Kontextmenü der neuen Operation wählen Sie **Request setzen • Vorhand. Message-Typ auswäh.**, um den Nachrichtentyp der Request-Nachricht zu setzen. Analog wählen Sie **Response setzen • Vorhand. Message-Typ auswäh.**, um den Nachrichtentyp der Request-Nachricht zu setzen. Abbildung 8.14 zeigt das Ergebnis.

Service-Provider anlegen

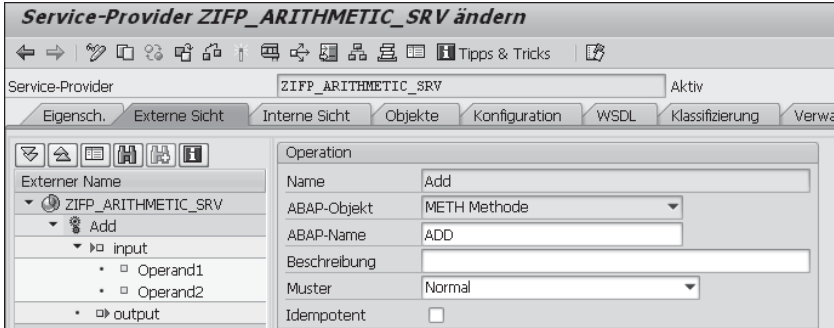


Abbildung 8.14 Service-Provider »ZIFP\_ARITHMETIC\_SRV«

**Service-Provider  
implementieren**

Wenn Sie auf die Registerkarte **Eigensch.** klicken, sehen Sie, dass für den Service-Provider ein ABAP-Interface mit den Methodendeklarationen und eine implementierende Klasse erzeugt wurden. Die Methoden dieser Klasse sind noch leer und müssen als Nächstes implementiert werden. Öffnen Sie dazu die generierte Methode der generierten Klasse, und geben Sie den Code aus Listing 8.4 ein.

```
METHOD z00_ii_zifp_arithmetic_srv~add.  
    output-zifp_add_res =  
        input-zifp_add_req-operand1 +  
        input-zifp_add_req-operand2.  
ENDMETHOD.
```

**Listing 8.4** Implementierung der Add-Methode

Nachdem Sie alles aktiviert haben, können Sie wieder zurück zu Ihrem Service-Provider gehen, ihn testen und einen Endpunkt dafür anlegen, so wie wir dies schon im Abschnitt 8.1.2, »SOAP-Webclient mit ABAP«, beschrieben haben.

**8.2.4 SOAP-Proxys in ABAP****Proxys erzeugen**

In diesem Abschnitt erzeugen wir für die eingehende Serviceschnittstelle `SI_IFP_ORDER_IN` einen Server- oder Provider-Proxy und für die ausgehende Serviceschnittstelle `SI_IFP_ORDER_OUT` einen Client- oder Consumer-Proxy. Wir schreiben einen ABAP-Report, der den Consumer-Proxy verwendet, um den Webservice aufzurufen, der durch den Provider-Proxy implementiert wird. Die Proxys können Sie entweder in Transaktion `SPROXY` erzeugen, oder Sie verwenden Transaktion `SE80` (Object Navigator), wie im Folgenden beschrieben.

**Provider****Enterprise  
Service anlegen**

Melden Sie sich am Empfangssystem an, und legen Sie dort im Object Navigator (Transaktion `SE80`) ein neues Paket an. In diesem Paket legen Sie wiederum ein neues Objekt vom Typ **Enterprise Service** an. Dies startet einen Wizard, der Sie in mehreren Schritten zum fertigen Provider-Proxy führt.

Im ersten Schritt nach der Übersicht wählen Sie die Option **Objekttyp Service-Provider**. Im nächsten Schritt selektieren Sie die Option **ESR-Service-Interface (Outside-in)**. Wählen Sie dann das ESR-Interface `SI_IFP_ORDER_IN` aus, und geben Sie ein Paket, ein Präfix und einen Workbench-Auftrag an. Sorgen Sie auch dafür, dass der Webservice sofort freigegeben wird, indem

Sie das Ankreuzfeld **Serv. deployen** auswählen. Markieren Sie das Feld nicht, müssen Sie den Webservice später in Transaktion SOAMANAGER manuell freigeben. Wählen Sie **Fertigstellen und Speichern**, und aktivieren Sie Ihren Proxy. Abbildung 8.15 zeigt das Ergebnis.

Es wurde ein Server-Proxy namens ZIFPII\_SI\_IFP\_ORDER\_IN erzeugt, der von der Klasse ZIFPCL\_SI\_IFP\_ORDER\_IN implementiert wird. Außerdem wurde die Webservice-Definition ZSI\_IFP\_ORDER\_IN angelegt. Da Sie im Service-Interface zwei Methoden definiert haben, verfügt auch der generierte Proxy über zwei Methoden (siehe Abbildung 8.15). Sie sehen die beiden Methoden GET\_DETAIL und GET\_LIST. Jede dieser Methoden hat einen Importparameter namens INPUT und einen Exportparameter namens OUTPUT. Für die Parameter wurden gemäß den Request- und Response-Message-Typen entsprechende ABAP-Strukturen generiert.

Server-Proxy

Den Code für die beiden Methoden müssen Sie selbst bereitstellen. Listing 8.5 zeigt dies am Beispiel der Methode GET\_LIST.

```
METHOD zifpii_si_ifp_order_in~get_list.
  DATA:
    orders TYPE STANDARD TABLE OF zifporder,
    wa_orders TYPE zifporder,
    wa TYPE zifpdt_ifp_orderhead,
    ordertype TYPE zifpordertype,
    refid TYPE zifprefid.
  ordertype = input-mt_ordergetlist_req-type.
  refid = input-mt_ordergetlist_req-refid.
  IF ordertype IS INITIAL.
    ordertype = 'SO'.
  ENDIF.
  CALL FUNCTION 'Z_IFP_ORDER_GETLIST'
    EXPORTING
      im_ordertype = ordertype
      im_refid      = refid
  TABLES
    ta_orders      = orders.
  LOOP AT orders INTO wa_orders.
    MOVE-CORRESPONDING wa_orders TO wa.
    APPEND wa TO output-mt_ordergetlist_res-orderheader.
  ENDLOOP.
ENDMETHOD.
```

**Listing 8.5** Implementierung der Methode »get\_list« in der Service-Provider-Klasse

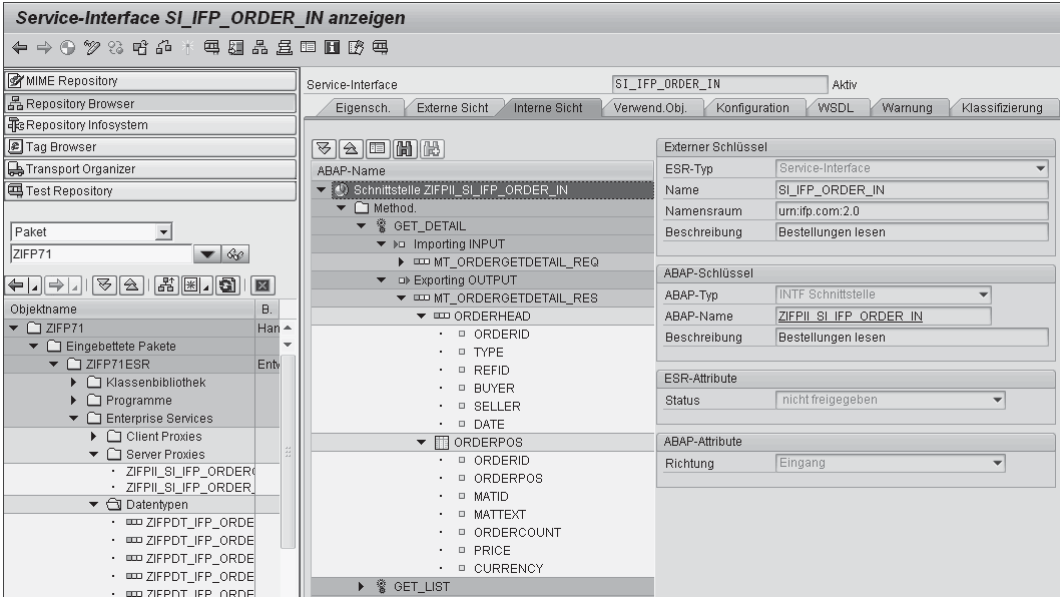


Abbildung 8.15 Generierter Service-Provider

Über Transaktion SOAMANAGER können Sie, wie schon bei der Outside-In-Entwicklung, einen Endpunkt anlegen, die WSDL-Datei generieren und Ihren Webservice testen. Diese WSDL-Datei benötigen Sie, wenn Sie später einen logischen Port für den Consumer anlegen.

Damit ist die Entwicklung des Service-Providers abgeschlossen, und Sie können sich nun dem Consumer zuwenden.

### Consumer

Melden Sie sich am Sendesystem an, und legen Sie dort im Object Navigator (Transaktion SE80) in einem Entwicklungspaket ein neues Objekt vom Typ *Enterprise Service* an. Dies startet einen Wizard, der Sie in mehreren Schritten zum fertigen Consumer-Proxy führt.

#### Service-Consumer

Im ersten Schritt nach der Übersicht wählen Sie die Option **Objekttyp Service-Consumer**. Im nächsten Schritt selektieren Sie die Option **ESR-Service-Interface (Outside-in)**. Wählen Sie das ESR-Interface `SI_IFP_ORDER_OUT` aus, und geben Sie ein Paket, ein Präfix und einen Workbench-Auftrag an. Wählen Sie **Fertigstellen und Speichern**, und aktivieren Sie Ihren Proxy. Abbildung 8.16 zeigt das Ergebnis. Es wurde ein Proxy namens `ZIFPCO_SI_IFP_ORDER_OUT` mit zwei Methoden erzeugt.

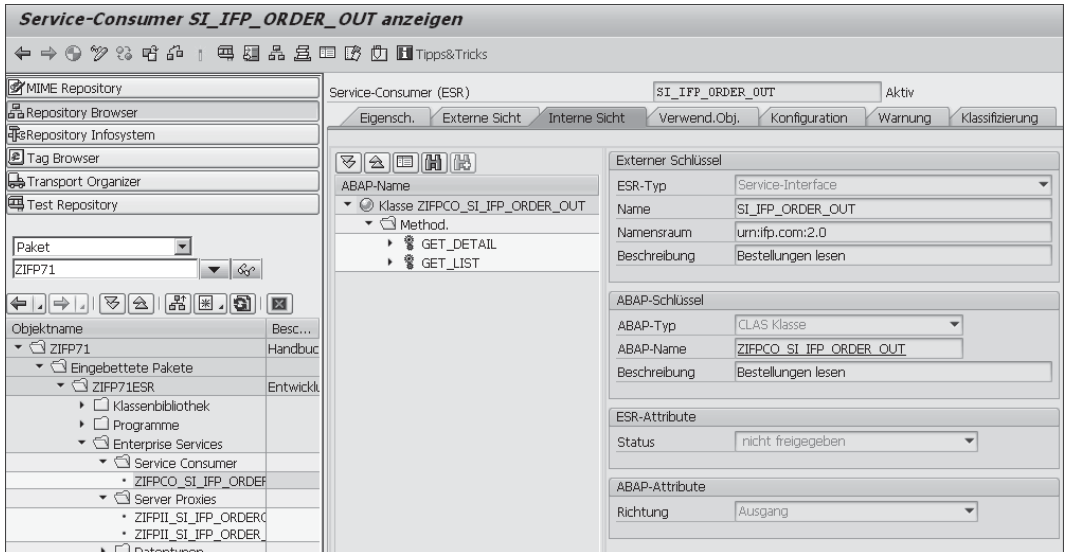


Abbildung 8.16 Consumer-Proxy

Damit Sie diesen Proxy aus einem ABAP-Programm nutzen können, müssen Sie so wie in Abschnitt 8.1.2, »SOAP-Webclient mit ABAP«, beschrieben, einen logischen Port anlegen.

**Logischer Port**

Sobald Sie den logischen Port angelegt haben, können Sie Ihren Consumer-Proxy in Transaktion SE80 testen. Sie können aber auch gleich den kleinen Test-Report aus Listing 8.6 schreiben. In diesem Report instanziiieren Sie den Consumer-Proxy und rufen anschließend die Methode `get_list` auf. Die zurückgegebenen Bestellköpfe geben Sie in einer Schleife aus.

**Consumer-Proxy testen**

```
REPORT zifp_test_esr.
DATA:
  rp TYPE REF TO zifpco_si_ifp_order_out,
  input TYPE zifpmt_ordergetlist_res,
  output TYPE zifpmt_ordergetlist_req,
  wa TYPE zifpdt_ifp_orderhead.
START-OF-SELECTION.
  TRY.
    CREATE OBJECT rp
      EXPORTING
        logical_port_name = 'LPO0'.
    CATCH cx_ai_system_fault.
  ENDTRY.
  TRY.
    CALL METHOD rp->get_list
```

```
EXPORTING
  output = output
IMPORTING
  input = input.
LOOP AT input-mt_ordergetlist_res-orderheader INTO wa.
  WRITE:/
    wa-orderid,
    wa-type,
    wa-refid,
    wa-buyer,
    wa-seller,
    wa-date.
ENDLOOP.
CATCH cx_ai_system_fault.
CATCH cx_ai_application_fault.
ENDTRY.
```

**Listing 8.6** Webservice-Consumer in ABAP

Damit sind die ABAP-Szenarien abgeschlossen, und wir wenden uns in den nächsten Abschnitten den Java-Szenarien zu.

## 8.3 SOAP-Programmierung mit Java

Auch wenn der SAP NetWeaver Application Server Java von SAP noch bis 2030 unterstützt wird, wird er nicht mehr weiterentwickelt. Daher wenden wir uns den im Standard-Java-Umfeld gebräuchlichen Werkzeugen und Standards zu. Wir diskutieren dabei die Verwendung der in Java 21 (Long Term Support, LTS) enthaltenen Mittel. Dabei gehen wir sowohl auf Werkzeuge als auch auf die zu verwendenden Java-Klassen ein. Auch wenn der Fokus von SAP in den letzten Jahren stärker auf RESTful APIs lag, ist die SOAP-basierte Webservice-Entwicklung weiterhin relevant – insbesondere im Enterprise-Umfeld, wo SOAP aufgrund seiner Standardisierung, Sicherheit und Transaktionsunterstützung oft bevorzugt wird.

### 8.3.1 Jakarta XML Web Services

#### Jakarta XML Web Services

Java bietet eine Reihe von eigenen Spezifikationen für die Arbeit mit Webservices an. Diese sind unter dem Namen *Jakarta XML Web Services* bekannt und bauen auf den Spezifikationen *Jakarta SOAP with Attachments* und *Jakarta Web Services Metadata* auf.