WSDL Versioning Best Practise

Adrian Trenaman
Principal Consultant, IONA Technologies.
adrian.trenaman@iona.com

Table of Contents

Table of Figures

Table of Code-snaps

Amendment History

Date	Author	Amendment
Tuesday, 10 April 2007	Adrian Trenaman	First cut of document created.
Wednesday, 11 April 2007	Adrian Trenaman	Additional work. Submitted for review in draft status.

1 Introduction

This document describes a scheme for versioning of WSDL artifacts, allowing major and minor releases to be easily tracked in a coherent manner. Appropriate semantics for *major* and *minor* releases are defined in terms of on-the-wire backwards compatibility; the versioning scheme proposed supports the evolutionary deployment of services with both incremental and major updates. The scheme uses common-sense naming conventions, and does not require additional tooling beyond currently available WSDL editors or text editors such as Notepad, Textpad or VI.

An overview of the scheme is presented in Section 2. Section 3 provides a deeper dive into the mechanics of the approach, with extracts from worked example of a versioned WSDL contract for an "Address Book" service. Finally, Section 4 provides the full, versioned WSDL contract for reference.

1.1 Intended Audience

This paper is intended for SOA architects and designers with experience in interface design using WSDL. The tone is reasonable technical, and assumes a good understanding of both WSDL and XSD, along with the wrapped-doc-literal convention of WSDL.

2 Major/Minor Versioning for WSDL contracts

In software, major-minor versioning is used accommodate two levels of change.

- A major change (that is, a change to the first digit) indicates a large update that creates an incompatibility with existing deployments. Major changes typically indicate large-scale revisions of the product, with significant new features and bug fixes.
- A minor change, that is, a change to second and subsequent digits, indicates an update that is backwards compatible with existing deployments of the software that share the same major version. Minor revisions typically contain a number of small new features, bug fixes and issue resolutions that do not break compatibility.

The major and minor release numbers are typically embedded in the software packing, using a format such as "<product> <major>.<minor>" for example, Artix 1.3, Artix 1.4 or Artix 2.0.

In the context of WSDL versioning, major-minor versioning can be used to distinguish between changes to a service contract that maintain backwards on-the-wire compatibility with existing service consumers, and those that break this compatibility and force a re-write of service consumer code. By adopting a coherent approach to major-minor versioning of WSDL artifacts, services can be deployed and evolved in a structured fashion.

Consider the service shown below in Figure 1. Version 1 of the service's WSDL contract defines the operations that the service implements, and the request and response messages that constitute each operation. These operations are marked as being part of v1.0.

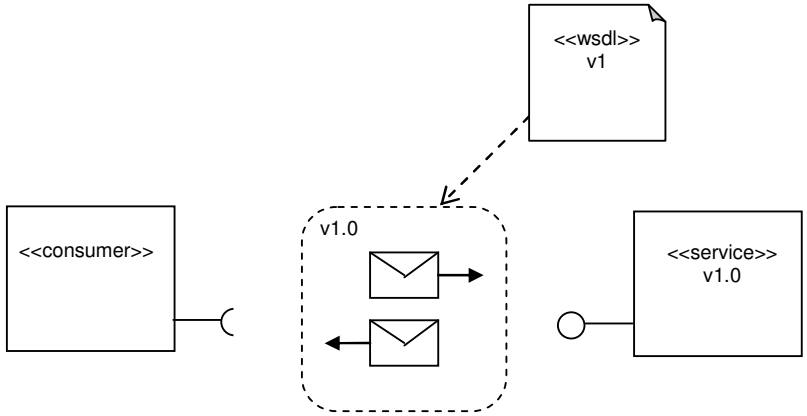

Figure 1: The WSDL contract defines the messages that pass between a consumer and the service.

Some time after deployment, a requirement emerges to introduce a new operation, in Version 1.1 of the WSDL. The WSDL can be designed in such a way that this new version of the service accepts the original messages from v1.0 clients, while also providing support for the additional messages for v1.1. This is shown below in Figure 2.

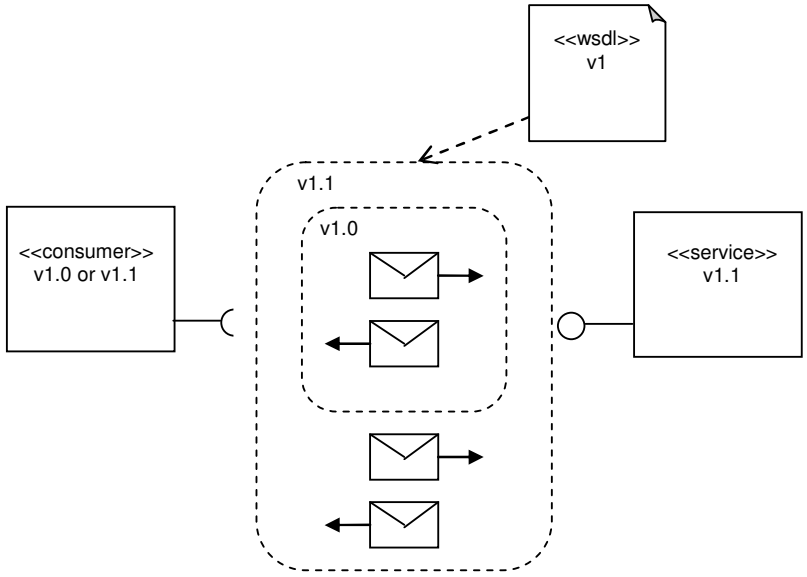

Figure 2: A minor update to the contract can be accommodated in a backwards compatible way by supporting all existing message interactions for previous versions, while adding new interactions as appropriate.

Some change requests will create unavoidable incompatibilities with existing service consumers. In this case, a new major release (v2.0) of the interface is made available. To facilitate a smooth transition over time, the last major release of the service (in this case, v1.1) should be kept live in a deprecated mode, until all consumers have moved to the new interface.

3 Best Practises for WSDL Versioning

This section provides detailed best practises for WSDL versioning. We use as a motivating example an “Address Book” service. Version 1.0 of the interface provides methods to create, retrieve and update contact information, where contact data-types have been provided as a separate XML schema definition. Version 1.1 of the interface

adds a new lookup method in a backwards compatible manner. Code that demonstrates the on-the-wire interoperability, developed using Artix™, is available from the author.

3.1 Do not use date (year/month) for WSDL versioning.

Do not use date-based versioning schemes, such as 2007/04, in your WSDL target namespace. This approach is used by XSD designers, where the date is embedded in the schema target namespace to indicate major release information and also give historical reference. Some WSDL designers have in the past used this approach in WSDL contracts; while it does facilitate major versioning it introduces a new lexical namespace with every contract and so renders the new contract incompatible with previous versions.

3.2 Place the major version number in the WSDL target namespace, and in the name of the WSDL file.

When creating a contract, embed the major number of the release in the WSDL target namespace, as shown in Code-snap 1:

```
<!-- Don't do this -->
<definitions name="AddressBook.wsdl"
    targetNamespace="http://www.iona.com/ps/wsdl/AddressBook"
     ...

<!-- Do this! -->
<definitions name="AddressBook-v1.wsdl"
    targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1"
     ...
```

Code-snap 1: Add major-version information to the WSDL target namespace and file name.

By encoding only the major release number in the namespace, successive minor releases share the same namespace and so are compatible.

3.3 Version data types according to XSD conventions, and import types into your WSDL contract

As mentioned above in Section 3.1, XML schema definitions are often versioned by embedding a numeric date like `2007/04` in the target namespace. It is appropriate to use this approach for schema; alternatively a major-only versioning scheme (such as a `-v1` or `-v2` suffix) is also appropriate. While the latter may be more in keeping with the versioning scheme advocated for WSDL in this paper, the date convention is more current with the XSD community and so that may be preferable.

The data-types a service uses should always be placed in separate `.xsd` files, with their own namespaces, rather than bundled in the `types` section of a WSDL document where they can become cluttered and confusing to the untrained eye. Use the `xsd:import` element to import data-types, as shown below in Code-snap 2.

```
<wsdl:types>
    <xsd:schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <xsd:import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
            schemaLocation="./AddressBook.xsd"/>

            ...
    </xsd:schema>
</wsdl:types>
```

Code-snap 2: Use of `xsd:import` to import data types from a separate namespace.

3.4 Encode major and minor version in the target namespace of the WSDL `<types>` section

Following on from Section 3.3, the data-types used by WSDL interfaces should be versioned separately. The WSDL contract typically imports these data-types for use in the definition of wrapped-doc-literal wrapper elements for each operation's messages. When defining the wrapper elements, define them in a namespace with an explicit major and minor number. This facilitates an explicit system of record for the incremental changes to the interface. An example is shown in Code-snap 3 below, where wrappers for an `addContact()` operation are defined in a namespace marked `v1.0`.

```
<types>
    <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">

        <import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
            schemaLocation="./AddressBook.xsd"/>

        <element name="addContact">
            <complexType>
                <sequence>
                    <element name="contact" type="addr:Contact"/>
                </sequence>
            </complexType>
        </element>
        <element name="addContactResponse">
            <complexType>
                <sequence>
                    <element name="id" type="int"/>
                </sequence>
            </complexType>
        </element>

    </schema>
</types>
```

Code-snap 3: Applying major-minor versioning to wrapper elements in the WSDL contract.

3.5 Avoid embedding versioning information in message names

Version information should *not* appear in message names. The reason for this is that WSDL code generators that adhere to the wrapped-doc-literal convention require that the name of the message be the same as the name of the operation, so that they can detect the presence of the wrapped convention and generated appropriate method signatures. Sample messages for an `addContact()` operation, appearing in `v1.0` of the `AddressBook` interface, are shown below in Code-snap 4.

The only exception to this rule is when you want a method to be explicitly versioned, and you want this version name to seen in the generated code.

```
<definitions name="AddressBook-v1.wsdl"
    targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.iona.com/ps/wsdl/AddressBook-v1"
    xmlns:v1_0="http://www.iona.com/ps/wsdl/AddressBook-v1.0">
    <types>
        <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">

            <!-- Types for v1.0 wrapper elements -->
            <element name="addContact">
                <complexType>
                    <sequence>
                        <element name="contact" type="addr:Contact"/>
                    </sequence>
                </complexType>
            </element>

            <element name="addContactResponse">
                <complexType>
                    <sequence>
                        <element name="id" type="int"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <message name="addContact">
        <part element="v1_0:addContact" name="parameters"/>
    </message>

    <message name="addContactResponse">
        <part element="v1_0:addContactResponse" name="parameters"/>
    </message>

</definitions>
```

Code-snap 4: Message names should not contain embedded version information, as this violates wrapped-doc-literal code generation cues.

3.6 Embed major-minor version in the interface (`portType`) name

It is a good idea to create an explicit interface (`portType`) for each version of an interface, embedding the major and minor number in the interface. For example, successive versions for the `AddressBook` service are named `AddressBook_v1_0`, `AddressBook_v1_1`, `AddressBook_v2_0`, etc. The interface name is mapped to the service endpoint interface in a destination language; by making the interface version explicit we allow the same code-base to implement different versions of the same interface. Explicit naming of versioned interfaces in this way also has the benefit of avoiding linkage errors or CLASSPATH issues in Java.

```
    <portType name="AddressBook_v1_0">
        <operation name="addContact">
            <input message="tns:addContact" name="addContact"/>
            <output message="tns:addContactResponse" name="addContactResponse"/>
        </operation>
    </portType>
```

3.7 Embed major-minor version in the service name.

Embed the major-minor version in the service name; effectively this follows the practise of using the same name for the service as that used for the interface. An example service for the `AddressBook_v1_0` interface is shown below in Code-snap 5.

```
    <service name="AddressBook_v1_0">
        <port binding="tns:AddressBook_v1_0_SOAP11Binding" name="SOAPOverHTTP">
            <soap:address location="http://localhost:9000/addressbook"/>
        </port>
    </service>
```

Code-snap 5: Embedding the version name into the service name.

3.8 Facilitate the addition of new operations as minor point releases

New operations can now be facilitated as minor-point releases by creating their wrapper elements in an appropriate namespace. Consider the change highlighted in Code-snap 6, where a new operation, `findContactByEmail()`, is added in the new `v1.1` namespace.

```
<types>
    <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">

        <import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
            schemaLocation="./AddressBook.xsd"/>

        <!-- Wrapper elements as before for version 1.0 -->

    </schema>

    <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.1"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">
        <import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
            schemaLocation="./AddressBook.xsd"/>
        <element name="findContactByEmail">
            <complexType>
                <sequence>
                    <element name="email" type="string"/>
                </sequence>
            </complexType>
        </element>
        <element name="findContactByEmailResponse">
            <complexType>
                <sequence>
                    <element name="id" nillable="true" type="int"/>
                </sequence>
            </complexType>
        </element>
    </schema>
</types>
```

Code-snap 6: Adding a new operation to a WSDL contract, using a major minor number.

When a new minor version is created, an appropriate interface should be created for the new version, as shown below in Code-snap 7. As WSDL does not support inheritance or aggregation of interfaces, we must copy in the operations from previous minor versions of the interface. This will change in WSDL2.0, when interface inheritance will be supported.

```
<portType name="AddressBook_v1_1">
    <!-- Backward compatible operations from v1.0 -->
    <operation name="addContact">
        <input message="tns:addContact" name="addContact"/>
        <output message="tns:addContactResponse" name="addContactResponse"/>
    </operation>
    <!-- New operations from v1.1 -->
    <operation name="findContactByEmail">
        <input message="tns:findContactByEmail" name="findContactByEmail"/>
        <output message="tns:findContactByEmailResponse"
            name="findContactByEmailResponse"/>
    </operation>
</portType>
```

Code-snap 7: Creating a new minor version of an interface.

3.9 Create a new service for each version of an interface

As mentioned previously in Section 3.7, each version of an interface should have its own service. This is shown below in Code-snap 8.

```
<service name="AddressBook_v1_0">
    <port binding="tns:AddressBook_v1_0_SOAP11Binding" name="SOAPOverHTTP">
        <soap:address location="http://localhost:9000/addressbook"/>
    </port>
</service>
<service name="AddressBook_v1_1">
    <port binding="tns:AddressBook_v1_1_SOAPBinding" name="SOAPOverHTTP">
        <soap:address location="http://localhost:9000/addressbook"/>
    </port>
</service>
```

Code-snap 8: Creating an appropriately named service for each version of the interface.

Note that both services share the same endpoint/address information. This allows us to take down version 1.0 of the service, restart with version 1.1 listening on the original endpoint, and now accept messages from clients using both version 1.0 and version 1.1 of the interface.

3.10 Facilitate change of existing operations only as major point releases

Sometimes operations are modified as part of the evolution to a contract. For example, a designer may with to change an operation name, or add or remove a parameter. A change of operation name or removal of a parameter will create an on-the-wire incompatibility, and is this forbidden in the context of a minor release.

It is possible to add new *optional* parameters to an operation signature, using the `minOccurs="0"` attribute. However, this practise should be avoided. When this practise is employed, it is not clear to the service consumer whether the parameter is optional because it is truly semantically optional, or whether it is optional only because an XSD hack was used to get around a versioning issue.

If an operation is to be changed, then this can be facilitated as part of a new minor release by adding a new operation signature. Note that WSDL does not support overloading of operation names, so a new operation name will have to be chosen. In the event that selecting a new name becomes difficult, then choose to add a version number to the original name, for example, `addCustomer_v1_0()`. While this may seem unergonomic, it is preferable to the alternative XSD hack described above as it makes the version change explicit while being backward-compatible.

3.11 Regard any change to the XSD type system as a major release

WSDL contracts often depend on schema definitions from a third-party. For example, data definitions for business artifacts such as invoices and purchase orders are defined by OASIS in the Unified Business Language (UBL) specification. If a new release of the data dictionary is to be adopted by your organisation, it will involve a new

XSD namespace: when the new namespace is imported to replace the old namespace in your contract then it will break on-the-wire compatibility.

Instead, choose to roll out new versions of data-dictionaries alongside a major release of your interface contract. The use of a major release will indicate to consumers that they must make appropriate code changes to use the new service.

3.12 Keep major releases of deprecated services live until they are no longer in use

New major releases of services are typically not deployed overnight, due to the downstream effects a major release will have on existing clients. Plan to keep deprecated versions of services in production for a phase-out period, allowing clients to be migrated as appropriate. Only when there is documented evidence that a service is no longer in use can it be gracefully removed from the SOA infrastructure.

As an alternative to keeping deprecated versions of services live, consider creating transformation services that use message transformation technologies such as XSLT to appropriately “upgrade” and “downgrade” messages on-the-wire. This approach can sometimes be effectively be used to allow clients use a deprecated interface to contact an otherwise incompatible new release.

4 Reference

Full WSDL for the versioned “Address Book” Service example is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AddressBook-v1.wsdl"
    targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:tns="http://www.iona.com/ps/wsdl/AddressBook-v1"
    xmlns:v1_0="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
    xmlns:v1_1="http://www.iona.com/ps/wsdl/AddressBook-v1.1"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <types>
        <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.0"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">
            <import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
                schemaLocation="./AddressBook.xsd"/>
            <element name="addContact">
                <complexType>
                    <sequence>
                        <element name="contact" type="addr:Contact"/>
                    </sequence>
                </complexType>
            </element>
            <element name="addContactResponse">
                <complexType>
                    <sequence>
                        <element name="id" type="int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="removeContact">
                <complexType>
                    <sequence>
                        <element name="id" type="int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="removeContactResponse">
                <complexType>
                    <sequence>
                        <element name="ret" type="boolean"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateContact">
                <complexType>
                    <sequence>
                        <element name="id" type="int"/>
                        <element name="contact" type="addr:Contact"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateContactResponse">
                <complexType>
                    <sequence>
                        <element name="ret" type="boolean"/>
                    </sequence>
                </complexType>
            </element>
            <element name="getContactUsingId">
                <complexType>
                    <sequence>
                        <element name="id" type="int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="getContactUsingIdResponse">
                <complexType>
                    <sequence>
                        <element name="contact" nillable="true"
type="addr:Contact"/>
                    </sequence>
                </complexType>
            </element>
```

```
            <element name="findContactByName">
                <complexType>
                    <sequence>
                        <element name="firstName" type="string"/>
                        <element name="lastName" type="string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="findContactByNameResponse">
                <complexType>
                    <sequence>
                        <element name="id" nillable="true" type="int"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
        <schema targetNamespace="http://www.iona.com/ps/wsdl/AddressBook-v1.1"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:addr="http://www.iona.com/ps/xsd/AddressBook/2007/04">
            <import namespace="http://www.iona.com/ps/xsd/AddressBook/2007/04"
                schemaLocation="./AddressBook.xsd"/>
            <element name="findContactByEmail">
                <complexType>
                    <sequence>
                        <element name="email" type="string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="findContactByEmailResponse">
                <complexType>
                    <sequence>
                        <element name="id" nillable="true" type="int"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>
    <message name="addContact">
        <part element="v1_0:addContact" name="parameters"/>
    </message>
    <message name="addContactResponse">
        <part element="v1_0:addContactResponse" name="parameters"/>
    </message>
    <message name="getContactUsingId">
        <part element="v1_0:getContactUsingId" name="parameters"/>
    </message>
    <message name="getContactUsingIdResponse">
        <part element="v1_0:getContactUsingIdResponse" name="parameters"/>
    </message>
    <message name="removeContact">
        <part element="v1_0:removeContact" name="parameters"/>
    </message>
    <message name="removeContactResponse">
        <part element="v1_0:removeContactResponse" name="parameters"/>
    </message>
    <message name="updateContact">
        <part element="v1_0:updateContact" name="parameters"/>
    </message>
    <message name="updateContactResponse">
        <part element="v1_0:updateContactResponse" name="parameters"/>
    </message>
    <message name="findContactByName">
        <part element="v1_0:findContactByName" name="parameters"/>
    </message>
    <message name="findContactByNameResponse">
        <part element="v1_0:findContactByNameResponse" name="parameters"/>
    </message>
    <message name="findContactByEmail">
        <part element="v1_1:findContactByEmail" name="parameters"/>
    </message>
    <message name="findContactByEmailResponse">
        <part element="v1_1:findContactByEmailResponse" name="parameters"/>
    </message>
    <portType name="AddressBook_v1_0">
        <operation name="addContact">
            <input message="tns:addContact" name="addContact"/>
```

```
            <output message="tns:addContactResponse" name="addContactResponse"/>
        </operation>
        <operation name="getContactUsingId">
            <input message="tns:getContactUsingId" name="getContactUsingId"/>
            <output message="tns:getContactUsingIdResponse"
                name="getContactUsingIdResponse"/>
        </operation>
        <operation name="removeContact">
            <input message="tns:removeContact" name="removeContact"/>
            <output message="tns:removeContactResponse"
name="removeContactResponse"/>
        </operation>
        <operation name="updateContact">
            <input message="tns:updateContact" name="updateContact"/>
            <output message="tns:updateContactResponse"
name="updateContactResponse"/>
        </operation>
        <operation name="findContactByName">
            <input message="tns:findContactByName" name="findContactByName"/>
            <output message="tns:findContactByNameResponse"
                name="findContactByNameResponse"/>
        </operation>
    </portType>
    <portType name="AddressBook_v1_1">
        <operation name="addContact">
            <input message="tns:addContact" name="addContact"/>
            <output message="tns:addContactResponse" name="addContactResponse"/>
        </operation>
        <operation name="getContactUsingId">
            <input message="tns:getContactUsingId" name="getContactUsingId"/>
            <output message="tns:getContactUsingIdResponse"
                name="getContactUsingIdResponse"/>
        </operation>
        <operation name="removeContact">
            <input message="tns:removeContact" name="removeContact"/>
            <output message="tns:removeContactResponse"
name="removeContactResponse"/>
        </operation>
        <operation name="updateContact">
            <input message="tns:updateContact" name="updateContact"/>
            <output message="tns:updateContactResponse"
name="updateContactResponse"/>
        </operation>
        <operation name="findContactByName">
            <input message="tns:findContactByName" name="findContactByName"/>
            <output message="tns:findContactByNameResponse"
                name="findContactByNameResponse"/>
        </operation>
        <operation name="findContactByEmail">
            <input message="tns:findContactByEmail" name="findContactByEmail"/>
            <output message="tns:findContactByEmailResponse"
                name="findContactByEmailResponse"/>
        </operation>
    </portType>
    <binding name="AddressBook_v1_0_SOAP11Binding" type="tns:AddressBook_v1_0">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="addContact">
            <soap:operation soapAction="" style="document"/>
            <input name="addContact">
                <soap:body use="literal"/>
            </input>
            <output name="addContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="getContactUsingId">
            <soap:operation soapAction="" style="document"/>
            <input name="getContactUsingId">
                <soap:body use="literal"/>
            </input>
            <output name="getContactUsingIdResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="removeContact">
```

```
            <soap:operation soapAction="" style="document"/>
            <input name="removeContact">
                <soap:body use="literal"/>
            </input>
            <output name="removeContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="updateContact">
            <soap:operation soapAction="" style="document"/>
            <input name="updateContact">
                <soap:body use="literal"/>
            </input>
            <output name="updateContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="findContactByName">
            <soap:operation soapAction="" style="document"/>
            <input name="findContactByName">
                <soap:body use="literal"/>
            </input>
            <output name="findContactByNameResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <binding name="AddressBook_v1_1_SOAPBinding" type="tns:AddressBook_v1_1">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="addContact">
            <soap:operation soapAction="" style="document"/>
            <input name="addContact">
                <soap:body use="literal"/>
            </input>
            <output name="addContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="getContactUsingId">
            <soap:operation soapAction="" style="document"/>
            <input name="getContactUsingId">
                <soap:body use="literal"/>
            </input>
            <output name="getContactUsingIdResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="removeContact">
            <soap:operation soapAction="" style="document"/>
            <input name="removeContact">
                <soap:body use="literal"/>
            </input>
            <output name="removeContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="updateContact">
            <soap:operation soapAction="" style="document"/>
            <input name="updateContact">
                <soap:body use="literal"/>
            </input>
            <output name="updateContactResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="findContactByName">
            <soap:operation soapAction="" style="document"/>
            <input name="findContactByName">
                <soap:body use="literal"/>
            </input>
            <output name="findContactByNameResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="findContactByEmail">
```

```
            <soap:operation soapAction="" style="document"/>
            <input name="findContactByEmail">
                <soap:body use="literal"/>
            </input>
            <output name="findContactByEmailResponse">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="AddressBook_v1_0">
        <port binding="tns:AddressBook_v1_0_SOAP11Binding" name="SOAPOverHTTP">
            <soap:address location="http://localhost:9000/addressbook"/>
        </port>
    </service>
    <service name="AddressBook_v1_1">
        <port binding="tns:AddressBook_v1_1_SOAPBinding" name="SOAPOverHTTP">
            <soap:address location="http://localhost:9000/addressbook"/>
        </port>
    </service>
</definitions>
```