

WSDL QUICK START

1. Introduzione

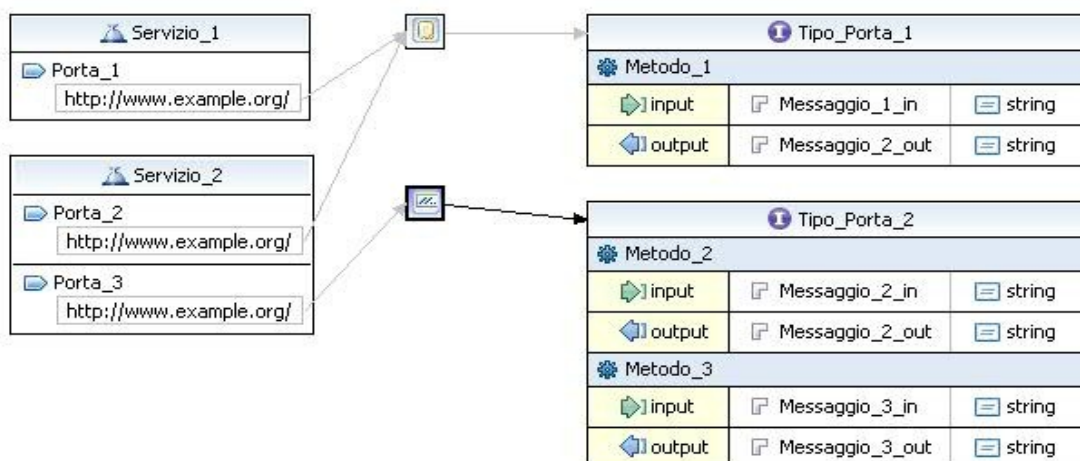
Il formato WSDL (Web Services Description Language) è un formato XML utilizzato per descrivere i parametri, l'ubicazione (End Point Reference) e i metodi offerti da un web service.

2. Struttura

Un documento WSDL è composto da 5 elementi principali:

- **types:** il tipo di dati utilizzati dal web service, definiti da un XML Schema posto all'interno del WSDL stesso (per questo motivo il *targetNamespace* del WSDL e del XSD solitamente coincidono).
- **message:** definizione della struttura dei parametri d'ingresso dei metodi del web service (gli argomenti della *firma* di un'operazione) e dei dati di ritorno.
- **portType:** definisce il web service vero e proprio, descrivendo i nomi dei metodi (*operation*) offerti dal servizio e i messaggi utilizzati da tali metodi (argomenti di input ed output).
- **binding:** specifica il mapping tra l'interfaccia esposta (le *porte* del servizio) e le *portType* definite sopra.
- **service:** definizione globale del servizio (riassume le definizioni precedenti).

Tale struttura permette di pensare ad un singolo *servizio* come ad un insieme di *porte*, che corrispondono agli **endpoint** (espressi con la sintassi URL) del servizio. Ciascuna porta è associata ad uno ed un solo *binding*. Ogni *binding* rappresenta una mappa verso un unico *portType*, per cui ogni chiamata è indirizzata senza ambiguità verso una singola *portType*. All'interno della *portType* bersaglio, in base al messaggio, viene attivato il metodo (*operation*) richiesto. Notare che porte diverse, appartenenti a servizi diversi, possono puntare allo stesso *binding* e quindi raggiungere la stessa *portType* (vedasi Porta_1 e Porta_2 in figura). Siccome un servizio può offrire più di una porta, ciascun servizio può raggiungere tutte le *portType* esistenti. L'insieme di tutti i servizi, con le relative porte, costituisce il *web service* completo.



Esistono quattro tipologie di operazioni (o metodi) chiamate MEP (Message Exchange Pattern):

- **One way operation:** il metodo accetta solamente dati in input (IN).
- **Request response operation:** ad un input segue sempre un output (IN-OUT).
- **Solicit response operation:** il metodo sollecita al client l'invio di un messaggio (OUT-IN).
- **Notification operation:** il metodo fornisce solamente dati in output (OUT).

3. Messaggi

Solitamente i messaggi d'ingresso hanno un nome del tipo `metodoXXXRequest`, mentre i messaggi d'uscita hanno un nome del tipo `metodoXXXResponse`; all'interno della definizione del messaggio si trova spesso una tag `<part>` che definisce gli elementi del messaggio. Ad esempio:

```
<wsdl:message name="Message_in">
  <wsdl:part name="param_1" element="sa:element_1"></wsdl:part>
</wsdl:message>
```

Nel caso di elementi complessi, la tag `<part>` contiene solamente un riferimento alla definizione dell'elemento, che è specificata dall'elemento `<xsd:schema>` all'interno della sezione `<wsdl:types>`.

In questo caso, la chiamata via SOAP al servizio potrebbe essere qualcosa del genere:

```
<ns:Message_in><element_1>foo</q0:element_1></ns:Message_in >
```

Mentre la chiamata via HTTP potrebbe essere una semplice invocazione GET:

```
http://localhost:8080/axis2/services/Servizio_1/Metodo_1?param_1=foo
```

Qualsiasi sia il binding utilizzato occorre fare attenzione alla dichiarazione del namespace (vedi sotto).

4. Port Type

E' l'elemento più importante di un documento WSDL, perché definisce l'associazione tra le operazioni offerte da una `portType` e i messaggi ad essa associati. Vediamo un esempio di *request-response operation*:

```
<wsdl:portType name="Tipo_Porta_1">
  <wsdl:operation name="Metodo_2">
    <wsdl:input message="sa:Message_in"></wsdl:input>
    <wsdl:output message="sa:Message_out"></wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

5. Binding SOAP

E l'elemento che definisce i protocolli di trasporto supportati dal web service. Nel caso del binding SOAP si dovrebbe avere qualcosa del genere:

```
<binding name="Tipo_Porta_1_SOAP" type="Tipo_Porta_1">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Metodo_2">
    <soap:operation style="document" soapAction="http://namespace.org/Metodo_2"/>
    <input><soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

L'attributo **name** può essere generico. L'attributo **type** deve corrispondere ad un'operazione esistente, in questo una tag del tipo `<portType name="Tipo_Porta_1">`. In questo modo ogni operazione offerta dal `PortType` è associata ad un elemento `<operation>` che definisce i messaggi d'ingresso e d'uscita.

L'attributo **style** può essere "rpc" oppure "document" (vedi più avanti), mentre l'attributo **transport** definisce il tipo di trasporto e protocollo: in questo caso si usa il trasporto `http` e il binding SOAP.

6. Binding HTTP

Nel caso del binding HTTP il documento WSDL deve contenere un ulteriore elemento del tipo:

```
<wsdl:binding name="Tipo_Porta_2_HTTP" type="Tipo_Porta_2">
  <http:binding verb="GET"/>
  <wsdl:operation name="Metodo_2">
    <http:operation location="/Metodo_2"/>
    <wsdl:input><http:urlEncoded/></wsdl:input>
    <wsdl:output><mime:content part="element_1" type="text/xml"/></wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Gli attributi **name** e **type** vanno definiti come nel caso del binding SOAP. L'attributo **verb** può essere POST o GET. L'attributo **location** definisce l'ultima parte dell'URL al quale sarà disponibile il servizio. In questo caso l'invocazione del servizio avverrà inviando una richiesta HTTP di tipo GET ad un URL del tipo:

```
http://localhost:8080/axis2/services/Servizio_2/Metodo_2
```

il parametro d'ingresso viene passato come in una comune richiesta GET, ad esempio appendendo la stringa ? Param_1=foo all'URL indicato. Il riferimento al nome del parametro (in questo caso Param_1) è dichiarata all'interno dell'elemento `<message>` che definisce il messaggio metodoXXXRequest del servizio.

7. Service

E' l'elemento riassuntivo del servizio (l'attributo *name* è indipendente dagli elementi precedenti):

```
<wsdl:service name="Servizio_1">
  <wsdl:port name="Servizio_1_SOAP" binding="Tipo_Porta_1_SOAP ">
    <soap:address location="http://www.example.org/">
  </wsdl:port>
</wsdl:service>
```

L'attributo *location* viene solitamente valorizzato dinamicamente a run-time dall'Application Server che espone il servizio, in modo da essere consistente con l'endpoint definito durante il deploy del web service.

8. Namespace

Abbiamo già visto che il documento WSDL contiene una sezione `<types>` per definire gli elementi XML utilizzati dal web service. Nella tag `<definitions>` all'inizio del documento il namespace degli elementi XML viene associato ad un prefisso, ad esempio `xmlns:gc="http://www.test.it/Servizio_1/"`. L'eventuale client SOAP generalmente utilizzerà un prefisso diverso, ad esempio:

```
<soapenv:Envelope xmlns:q0="http://www.test.it/Servizio_1/">
```

I prefissi lato client e lato server possono essere diversi, come in questo caso, ma devono essere riferiti allo stesso namespace, altrimenti gli elementi XML non verranno riconosciuti.

9. WSDL styles

RPC/Encoded e RPC/Literal

Questi stili non sono facilmente validabili perché usano tag non definite in alcun schema, in quanto fanno riferimento solamente ai tipi di dati standard previsti dalla normativa XMLSchema. Per questo motivo solitamente non utilizzano dati complessi (tag nidificate) ma solamente dati semplici (*simple types*).

Lo stile RPC/Literal soddisfa i requisiti WS-I (Web Service Interoperability), mentre lo stile RPC/Encoded NON gli soddisfa. Un messaggio SOAP in questo stile solitamente è qualcosa del genere:

RPC/Literal	RPC/Encoded
<pre> <soap:envelope> <soap:body> <myMethod> <x>42</x> <y>17.0</y> </myMethod> </soap:body> </soap:envelope> </pre>	<pre> <soap:envelope> <soap:body> <myMethod> <x xsi:type="xsd:int">42</x> <y xsi:type="xsd:float">17.0</y> </myMethod> </soap:body> </soap:envelope> </pre>

Document/Literal

Permette la validazione dell'XML rispetto un XSD perché lo stile contiene la definizione dello schema usato all'interno della tag `<wsdl:types>`. Per questo motivo è facile definire elementi complessi (tag nidificate).

Il messaggio SOAP nello stile Document/Literal ha una forma del tipo:

```

<soap:envelope>
  <soap:body>
    <xElement>42</xElement>
    <yElement>17.0</yElement>
  </soap:body>
</soap:envelope>

```

Che non soddisfa completamente gli standard WS-I, perché lo standard WS-I prevede l'esistenza di un unico figlio della tag `<soap:body>`, mentre in questo caso ne abbiamo due. Un altro svantaggio è l'assenza del nome del metodo da invocare, per cui il dispatcher dell'Application Server potrebbe avere difficoltà nel capire dove inoltrare il messaggio. Il problema viene definitivamente risolto dallo stile Document/literal wrapped.

Document/literal wrapped

E' una versione più complicata del precedente, che però soddisfa lo standard WS-I.

Il messaggio SOAP è del tipo:

```

<soap:envelope>
  <soap:body>
    <myMethod>
      <x>42</x>
      <y>17.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>

```

Adesso il nome del metodo appare come unico figlio della tag `<soap:body>`, risolvendo i problemi precedenti. L'unico prezzo da pagare è che il documento WSDL diventa molto più lungo e complesso, poiché definisce tutti i *data types* e i messaggi in modo separato dai metodi offerti dal servizio.

Per questo motivo lo stile Document/Literal wrapped è il più usato. Ad esempio il plug-in *Web Tool Platform* di Eclipse, così come il tools *wsdl2Java* di Axis2, propongono come scelta di default per lo stile Document/Literal.

Quando usare lo stile RPC?

Da quanto detto è evidente che conviene quasi sempre utilizzare lo stile Document/Literal (si dà spesso per sottinteso il termine *wrapped*). L'unico caso in cui può risultare conveniente ricorrere allo stile RPC è quando il linguaggio che implementa il servizio sfrutta pesantemente l'*overload* dei metodi. Poiché non è possibile avere due figli con lo stesso nome della tag `<soap:body>`, lo stile Document/Literal può rendere difficoltosa l'invocazione del metodo corretto. In alcuni di questi casi lo stile RPC può migliorare le performance del servizio.

Lo stile RPC/Encoded risulta molto utile per definire delle "mappe logiche" o "strutture ad albero", detti *data graphs*, poiché permette di specificare l'attributo `href` direttamente nella tag XML, in modo da collegare tra loro i vari elementi. Lo svantaggio è che non esiste uno standard per tale codifica, per cui l'interoperabilità del servizio non è garantita.