How to use Web Service facility in MXQuery

1. Introduction

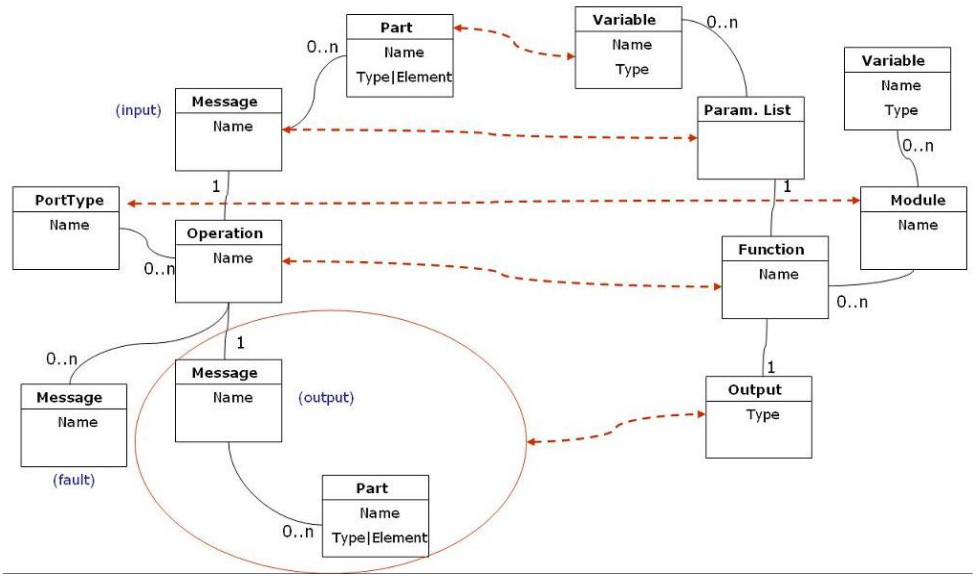
Since web services rely on XML both as a specification language (through WSDL), and as a messaging layer (through SOAP), potentially they can be benefitted from XQuery as an XML processing language. In contrast to usual programming infrastructures like java, XQuery does have the overhead of the Object/XML mapping.

Currently, the standard XQuery language specification doesn't support web services. We have proposed to extend the language with relevant constructs in order to support web services. This extension covers both of the SOAP based and RESTful web services. MXQuery has almost fully implemented these proposals. Following sections include a brief description of the proposals, and some examples on how to use them.

2. WSDL/SOAP based web services

2.1. Background

We have exploited the natural parallelism between XQuery modules and WSDL portTypes as depicted in the figure below. Detailed mappings are discussed in [1].



For example the following WSDL contract

```
<definitions
targetNameSpace="http://www.ethz.ch/"
xmlns:tns="http://www.ethz.ch/"
xmlns="http://schemas.xmlsoap.org/wsdl/" >
        <message name = "add">
```

```
            <part name="param1" type="xs:integer"/>
            <part name="param2" type="xs:integer"/>
        </message>
        <message name="addResponse">
            <part name="result" type="xs:integer"/>
        </message>
        <portType name="addPortType">
            <operation name="add">
                <input message="tns:add"/>
                <output message="tns:addResponse"/>
            </operatiaon>
        </portType>
        ...
        <service name="FirstExample">
            <port name="FirstExamplePort" ... >
                ...
            </port>
        </service>
</definitions>
```

is mapped to a XQuery module like:

```
module namespace ws = "http://www.ethz.ch/";
declare function ws:add($param1 as xs:integer, $param2 as xs:integer) as xs:integer;
```

notice that this library module only includes the signature of the functions and the when they are called, the runtime environment of the XQuery has to bind the call with the remote implementation of it. Hence the it's completely transparent from the programmer's point of view. Following subsections provide examples using our implementation of this extension in MXQuery.

2.2. Examples on import web service into the MXQuery environment

There are two ways of calling a remote web service from an XQuery code excerpt. One is

Example 1: Calling the Google's spell-checking service (The corresponding WSDL file can be found here [2]) to find the correct spelling of the word 'relattion'.

```
import module namespace ws="urn:GoogleSearch" at "http://api.google.com/GoogleSearch.wsdl" ;
ws:doSpellingSuggestion("oIqddkdQFHIlwHMXPerc1KlNm+FDcPUf","relattion")
```

It should be noted that the first argument is a string value key, required by Google to keep track of the clients.

Example 2: Composing two remote web services, a spell‐checking service [2] and a dictionary service [3], using XQuery (instead of BPEL!)

```
import module namespace ggl="urn:GoogleSearch" at "http://api.google.com/GoogleSearch.wsdl";
```

```
import module namespace dct="http://services.aonaware.com/webservices/" at
"http://services.aonaware.com/DictService/DictService.asmx?WSDL";

let $spelledWord := ggl:doSpellingSuggestion("oIqddkdQFHIlwHMXPerc1KlNm+FDcPUf",$input)
return
    if (fn:string-length($spelledWord)=0)
        then (dct:Define(<Define
xmlns="http://services.aonaware.com/webservices/"><word>{$input}</word></Define>)//dct:WordDe
finition)
         else (dct:Define(<Define
xmlns="http://services.aonaware.com/webservices/"><word>{$spelledWord}</word></Define>)//dct:
WordDefinition)
```

briefly, this short XQuery code first checks whether the input word has the correct spelling or not. If yes (result of the spell checking service is empty), returns the definition for the original word, otherwise returns the definition of the corrected word.

- **Low level soap call:**

In order to have a more flexible and powerful interaction with the SOAP based Web Services, MXQuery also provides a low level access to the web service functions from the XQuery programs which allows programmers to have more control over the interactions. It has been done by adding two simple functions to the set of XQuery built-in functions which let us to compose custom request messages. Signatures of these functions are as follow:

```
fn:soap-call($locationURI as xs:anyURI,$xmlcontent as node()?) as document()?

fn:soap-call($locationURI as xs:anyURI, $method as xs:string, $header as xs:string,$xmlcontent as
node()?) as document()?
```

Example 3: calling the 'doSpellingSuggestion' method from the Google's search service by manually created the soap message:

```
let $input:=
    <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
                <SOAP-ENV:Body>
                    <tns:doSpellingSuggestion xmlns:tns='urn:GoogleSearch' SOAP-
                    ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
                        <key xsi:type='xsd:string'>oIqddkdQFHIlwHMXPerc1KlNm+FDcPUf</key>
                            <phrase xsi:type='xsd:string'>relattion</phrase>
                    </tns:doSpellingSuggestion>
            </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>

let $temp:= soap-call(xs:anyURI("http://api.google.com/search/beta2"),"POST","",$input)
   return $temp//return/text()
```

2.3. Examples on exposing MXQuery library modules as web services

In order to expose our XQuery programs, we need to use a web server and also a web based version of the MXQuery implementation. In current release of the MXQuery there is ".war" version of the MXQuery which you can deploy it to any java servlet container.

To expose your XQuery library modules as a web service, you need to
1) copy your XQuery file (with extension '.xquery') to the root directory of MXQuery web application on the server. Please notice that there are some are sample modules already copied there.

2) if your library module is using (importing) schema types, you also need to copy those type definitions (as '.xsd' files) to the root directory of the MXQuery web application. (as in the provided samples)

by doing this, you should be able to view the list of exposed modules (and schemas) at: http://serveraddress:port/MXQuery/

Example 1: as a complementary for the previous subsection, if you want to expose the result of a web service composition as another web service on its own, you just need to wrap the composition code in an XQuery function/module declaration and then copy it to the MXQuery directory on the server. For example, our complex spell-checking/dictionary operation can be wrapped in following fashion

```
module namespace cmp ="http://ethz.ch/";
import module namespace ggl="urn:GoogleSearch" at "http://api.google.com/GoogleSearch.wsdl";
import module namespace dct="http://services.aonaware.com/webservices/" at
"http://services.aonaware.com/DictService/DictService.asmx?WSDL";
declare function cmp:spellAndDict($input as  xs:string) {
    let $spelledWord := ggl:doSpellingSuggestion("oIqddkdQFHIlwHMXPerc1KlNm+FDcPUf",$input)
    return
        if (fn:string-length($spelledWord)=0)
            then (dct:Define(<Define
xmlns="http://services.aonaware.com/webservices/"><word>{$input}</word></Define>)//dct:WordDe
finition)
            else (dct:Define(<Define
xmlns="http://services.aonaware.com/webservices/"><word>{$spelledWord}</word></Define>)//dct:
WordDefinition)
 };
```

And after copying it to the MXQuery directory on the server, following WSDL will be generated

```
<definitions name="MXQueryWebService"  targetNamespace="http://ethz.ch/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"  xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://ethz.ch/"  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/" >
  <message name="spellAndDict">
```

```
  <part name="p0" type="xs:string"/>
 </message>
 <message name="spellAndDictResponse">
  <part name="return" type="xs:anyType"/>
 </message>
 <portType name="port">
  <operation name="spellAndDict">
   <input name="input1" message="tns:spellAndDict"/>
   <output name="output1" message="tns:spellAndDictResponse"/>
  </operation>
 </portType>
 <binding name="binding" type="tns:port">
   ...
 </binding>
 <service name="MXQueryWebService">
   ...
 </service>
</definitions>
```

NOTE: there is no need to restart the servlet container each time you copy a new XQuery module. You only need to refresh the page including list of available services.

3. RESTful Web services

REpresentational State Transfer (REST) is an architectural style for distributed systems in general and web services in particular. As an alternative for traditional SOAP/WSDL web services, it has drawn a lot of attention recently. The main advantage of RESTful web services is their natural simplicity. Here web service functions are same as the HTTP 'verbs' which are applied to different resources which have different URLs . See [4] for more information on the REST style.

In order to provide the RESTful web service facility in any computing environment, one has to provide a function for each of the HTTP verbs. In MXQuery, we have adopted the approach taken by the Zorba XQuery engine implementation as discussed explained in [5]. Detailed signature of the REST functions are described in [6].

In order to use REST functionality, the module "http://www.zorba-xquery.com/zorba/rest-functions" has to be included at the beginning of the query.

Example: the 'GET' function

```
import module namespace zorba-rest = "http://www.zorba-xquery.com/zorba/rest-functions";

let $result := zorba-rest:get("http://tidy.sourceforge.net/docs/quickref.html")
let $payload := $result//zorba-rest:payload
return  $payload
```

Example: the POST

First deploy the provided simple server application (RestSOAPTestServer.war) on a servlet container and then execute the following query

```
import module namespace zorba-rest = "http://www.zorba-xquery.com/zorba/rest-functions";
zorba-rest:post('http://serveraddress:port/RestSOAPTestServer/RestParamTest',
<payload>123456</payload>)
```


[1] Kyumars S. Esmaili, Peter M. Fischer, Jerome Simeon, "XQuery 1.0 Web Services Facility", XQuery extension proposal presented to XQuery working group.
[2] http://api.google.com/GoogleSearch.wsdl
[3] http://services.aonaware.com/DictService/DictService.asmx?WSDL
[4] http://en.wikipedia.org/wiki/Representational_State_Transfer
[5] Zorba Development Team, "A REST proposal for XQuery".
[6] http://www.zorba-xquery.com/doc/zorba-0.9.5/zorba/html/rest.html