XQuery 1.0 Web Services Facility

(Proposal)

Kyumars Sheykh Esmaili, Peter Michael Fischer, Jerome Simeon

kyumarss@inf.ethz.ch, peter.fischer@inf.ethz.ch, simeon@us.ibm.com

October 30, 2008

Abstract

XQuery is a powerful language for processing XML. In the context of the Web, XML data is often provided through Web services. Several XQuery implementations already support access to such services (for both WSDL and REST). We propose here an extension to XQuery 1.0 that allows interoperability with WSDL services in a simple and transparent way.

1 Introduction

XQuery[1] is an expressive language to process XML data. It builds on existing XML technologies such as XPath and XML Schema. Web services also rely on XML both as a specification language (through WSDL[2]), and as a messaging layer (through SOAP[3]). This makes XQuery a particularly good match to build or access Web Services. This note proposes extensions to XQuery 1.0 to import WSDL-based Web services in queries, and to export queries as Web services.

We believe this extension is useful for XQuery users, as it enables:

- Direct manipulation or creation of Web Services using a high-level, and expressive language for XML such as XQuery.
- Access to the type information for the Web services data by leveraging the operations descriptions present in WSDL (specified in XML Schema).
- Transparent access to WSDL operations as standard XQuery functions.

We provide usecases as illustrate and motivation Section 3. We believe this extension is a good candidate for standardization for the following reasons.

- It is a feature present in the XQuery 1.1. requirements [2.3.21 Invocation of external functionality].
- It is a fairly simple extension to XQuery 1.0. This is possible by exploiting the natural parallelism between XQuery modules and WSDL services[4, 5].
- Several vendors and implementations already provide such functionality, which shows an interest in that feature. BEA AquaLogic[6], Stylus Studio[7], Galax[8] and MXQuery[9] (possibly others) provide web service support by extending the syntax of the language in a way which is similar to the proposal here.

Before we present the proposal, we briefly review some important concepts about WSDL. A sample WSDL description (parts of Google Search's web service) is provided in section 3.4.2. Every WSDL description provides the following information[10]: 1) *what* are the offered functionalities, 2) *how* are they provided?, and 3) *where* are they accessible?

The *portType* component which represents the abstract definition of web services answers the first question and therefore it must be a part of the language syntax.

The *binding* element in WSDL tells the service requestor how to format the message in a protocol specific manner. It answers the *how* question and so, it is an implementation issue that should be taken into account by the environment. Finally, the *service* element in a WSDL answers the *where* question by specifying the name of service, its ports and network address of each port. Since these elements partly depend on the *binding* component (as an attribute of *port*[1] element) and server configuration (as a subelement of it) they are likely to be set by environment, but it can be useful to allow the XQuery programmer to assign the name of both of *service* and *port* elements.

The rest of this note is organized as follows. Section 2 reviews the key requirements that we believe should be met when extending XQuery with Web services support. Section 3 presents some usecases, and illustrate how they are implemented using our proposal. Section 4 describes changes to the XQuery 1.0 and XPath 2.0 Functions and Operators documents. Section 5 describes changes to the XQuery 1.0 language document, notably grammar changes. The semantic of the Web services import and export features is based on a correspondence between WSDL and XQuery modules which is described in Section 6. The Section 7 lists some open issues.

2 Requirements

Since we propose to provide the web service facility at the level of module, we would have the following requirement:

- Exposing XQuery functions as web service: which lets the remote developers -whatever their development language and environment is- to use our operational functionalities. So, we need to generate a WSDL document for each library module in XQuery which describes all included functions in a way that every remote invoker would have all required items to call these functions.
- High level access to the web service functions in XQuery programs: this would help us to easily interact with remote processing nodes in a very high level and transparent way. It means we need to have a local stub module for each remote web service.
- Low level access to the web service functions in XQuery programs: this provides more extensible and powerful interactions. Hence we should have simple built-in functions which allow us to compose custom request messages. Such functions can be also re-used in implementing the high level web service import.

3 Use Cases

Here are some use cases for the requirements mentioned in previous section

3.1 Use Case 'R' - Publish a Relational Database on the Web

One important use of web services in XQuery will be to access data stored in non-local (relational) databases. This access will be provided by exposing (parameterized) XML views to the data, and utilizing web services as a standard way to provide the parameters and to transfer the data.

3.1.1 Description

A (relational) database system might present a view in which each table (relation) takes the form of an XML document. These individual tables are joined into a parameterized views using XQuery and expressed as a user-defined function. This user-defined function is then exposed as a web service.

The following database sample is taken from the XQuery Use Cases [11]: Consider a relational database used by an online auction. The auction maintains a USERS table containing information on registered users, each identified by a unique userid, who can either offer items for sale or bid on items. An ITEMS table lists items currently or recently for sale, with the userid of the user who offered each item. A BIDS table contains all bids on record, keyed by the userid of the bidder and the item number of the item to which the bid applies.

[1] The 'port' element is renamed to 'endpoint' in WSDL 2.0, here we use both terms interchangeably

3.1.2 Document Type Definition

```
<!DOCTYPE users [ <!ELEMENT users
                        (user_tuple*)> <!ELEMENT user_tuple
                        (userid, name, rating?)> <!ELEMENT userid
                        (#PCDATA)> <!ELEMENT name
                        (#PCDATA)> <!ELEMENT rating
                        (#PCDATA)> ]> <!DOCTYPE items [
                        <!ELEMENT items (item_tuple*)>
                        <!ELEMENT item_tuple (itemno, description,
                        offered_by, start_date?, end_date?, reserve_price?
                        )> <!ELEMENT itemno (#PCDATA)>
                        <!ELEMENT description (#PCDATA)>
                        <!ELEMENT offered_by (#PCDATA)>
                        <!ELEMENT start_date (#PCDATA)>
                        <!ELEMENT end_date (#PCDATA)>
                        <!ELEMENT reserve_price (#PCDATA)>
                        ]> <!DOCTYPE bids [ <!ELEMENT bids
                        (bid_tuple*)> <!ELEMENT bid_tuple (userid,
                        itemno, bid, bid_date)> <!ELEMENT userid
                        (#PCDATA)> <!ELEMENT itemno
                        (#PCDATA)> <!ELEMENT bid (#PCDATA)>
                        <!ELEMENT bid_date (#PCDATA)>
                    ]>
```

3.1.3 Sample Data

```
<!-- Items.Xml -->
<items>
  <item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>1999-01-05</start_date>
    <end_date>1999-01-20</end_date>
    <reserve_price>40</reserve_price>
  </item_tuple>
  ...
</items>

<!-- Users.Xml -->
<users>
  <user_tuple>
    <userid>U01</userid>
    <name>Tom Jones</name>
    <rating>B</rating>
  </user_tuple>
  ...
</users>

<!-- Bids.Xml -->
<bids>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>35</bid>
    <bid_date>1999-01-07</bid_date>
    </bid_tuple>
  </bid_tuple>
  ...
</bids>
```

3.1.4 Q1

find the highest bid of a given userid on a given itemno.

Solution in XQuery:

```
module namespace exm="http://example.net";

declare function exm:highest-bid($userid as xs:string, $itemno as xs:integer) as xs:double
{
max(doc("bids.xml")//bid_tuple[userid = $userid and itemno = $itemno]/bid)
};
```

Expected Web Service Interface (WSDL):

```
<definitions
  targetNamespace="http://example.net"
  xmlns:typens="http://example.net"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="highest-bidRequest">
    <part name="userid" type="xsd:string"/>
    <part name="itemno" type="xsd:integer"/>
  </message>
  <message name="highest-bidResponse">
    <part name="return" type="xsd:double"/>
  </message>
  <portType name="highest-bidPortType">
    <operation name="highest-bid">
      <input message="typens:highest-bidrRequest"/>
      <output message="typens:highest-bidResponse"/>
    </operation>
  </portType>
  <binding name="highest-bidSoapBinding" type="typens:highest-bidPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="highest-bid">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="highest-bidService">
    <port name="highest-bidPort" binding="typens:highest-bidSoapBinding">
      <soap:address location="...."/>
    </port>
  </service>

</definitions>
```

3.1.5 Q2

This query is a generalized form of query *1.4.4.3 Q3* in the XQuery Use Cases[11]:

Find cases where a user with a rating worse (alphabetically, greater) than a given rating is offering an item with a reserve price of more than a given price.

Solution in XQuery:

```
module namespace exm="http://example.net";

declare function exm:warning($rating as xs:string, $price as xs:integer) as element()
{
<result>
{
    for $u in doc("users.xml")//user_tuple
    for $i in doc("items.xml")//item_tuple
    where $u/rating > $rating
       and $i/reserve_price > $price
       and $i/offered_by = $u/userid
    return
        <warning>
            { $u/name }
            { $u/rating }
            { $i/description }
            { $i/reserve_price }
        </warning>
    }
</result>
};
```

Expected Web Service Interface (WSDL):

```
<definitions
  targetNamespace="http://example.net"
  xmlns:typens="http://example.net"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="warningRequest">
    <part name="rating" type="xsd:string"/>
    <part name="price" type="xsd:integer"/>
  </message>
  <message name="warningResponse">
    <part name="return" type="anyType"/>
  </message>
  <portType name="warningPortType">
    <operation name="warning">
      <input message="typens:warningRequest"/>
      <output message="typens:warningResponse"/>
    </operation>
  </portType>

  <binding name="warningSoapBinding" type="typens:warningPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="warning">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="warningService">
```

```
    <port name="warningPort" binding="typens:warningSoapBinding">
      <soap:address location="...."/>
    </port>
  </service>

</definitions>
```

3.2 Use Case 'STRONG' - Distributed Processing of the Strongly Typed Data

3.2.1 Description

Strongly typed and weakly typed data are both important kinds of XML data. This use case explores XQuery Web service's support for types, using data that is governed by a strongly typed XML Schema and is adapted from the XQuery Use Cases [11].

3.2.2 Schemas

The schema for this example is International Purchase Order schema ('ipo.xsd') which imports a schema for addresses ('address.xsd'). Additionally we have a schema for American zip codes ('zip.xsd'). All of these schemas are available in the XQuery Use Cases [11].

3.2.3 Q1

This query is same as the query *1.9.4.2 Q2* in the XQuery Use Cases[11]

A function that tests an American address to check if it has the right zip code.

Solution in XQuery:

```
module namespace z="http://www.example.com/xq/zips";
import schema namespace ipo = "http://www.example.com/IPO" at "ipo.xsd";
import schema namespace zips = "http://www.example.com/zips" at "zips.xsd";

declare function z:zip-ok($a as element(*, ipo:USAddress))
  as xs:boolean
{
  some $i in doc("zips.xml")/zips:zips/element(zips:row)
  satisfies $i/zips:city = $a/city
        and $i/zips:state = $a/state
        and $i/zips:zip = $a/zip
 };
```

Expected Web Service Interface (WSDL):

```
<definitions
  targetNamespace="http://example.net"
  xmlns:typens="http://example.net"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ipo="http://www.example.com/IPO"
  xmlns:zip="http://www.example.com/zips"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
   <xsd:schema targetNamespace="http://example.net" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.example.com/IPO"/>
    <xsd:import namespace="http://www.example.com/zips"/>
   </xsd:schema>
  </types>
  <message name="zip-okRequest">
    <part name="a" type="ipo:USAddress"/>
  </message>
```

```
  <message name="zip-okResponse">
    <part name="return" type="xsd:boolean"/>
  </message>
  <portType name="zip-okPortType">
    <operation name="zip-ok">
      <input message="typens:zip-okRequest"/>
      <output message="typens:zip-okResponse"/>
    </operation>
  </portType>
  <binding name="zip-okSoapBinding" type="typens:zip-okPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="zip-ok">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="zip-okService">
    <port name="zip-okPort" binding="typens:zip-okSoapBinding">
      <soap:address location="...."/>
    </port>
  </service>

</definitions>
```

3.3 Use Case 'Mashup' - Web Service Combination

3.3.1 Description

This use case shows how to produce a new local functionality by combining remote ones. Two currency exchange web services are used for finding the maximum conversion rate between 'USD' and 'CHF'.

3.3.2 Basic Web Services

The first basic web serive which we use here is CurrencyConvertor[2]:

```
<wsdl:definitions targetNamespace="http://www.webserviceX.NET/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET/">
      <s:element name="ConversionRate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="FromCurrency" type="tns:Currency"/>
            <s:element minOccurs="1" maxOccurs="1" name="ToCurrency" type="tns:Currency"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:simpleType name="Currency">
        <s:restriction base="s:string">
          <s:enumeration value="AFA"/>
            ...
          <s:enumeration value="TRY"/>
        </s:restriction>
      </s:simpleType>
      <s:element name="ConversionRateResponse">
        <s:complexType>
```

[2] http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

```
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="ConversionRateResult" type="s:double"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="double" type="s:double"/>
  </s:schema>
</wsdl:types>
<wsdl:message name="ConversionRateSoapIn">
  <wsdl:part name="parameters" element="tns:ConversionRate"/>
</wsdl:message>
<wsdl:message name="ConversionRateSoapOut">
  <wsdl:part name="parameters" element="tns:ConversionRateResponse"/>
</wsdl:message>
....
<wsdl:portType name="CurrencyConvertorSoap">
  <wsdl:operation name="ConversionRate">
    <documentation> Get conversion rate from one currency to another currency ...
    </documentation>
    <wsdl:input message="tns:ConversionRateSoapIn"/>
    <wsdl:output message="tns:ConversionRateSoapOut"/>
  </wsdl:operation>
</wsdl:portType>
....
<wsdl:binding name="CurrencyConvertorSoap" type="tns:CurrencyConvertorSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="ConversionRate">
    <soap:operation soapAction="http://www.webserviceX.NET/ConversionRate" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
...
<wsdl:service name="CurrencyConvertor">
  <wsdl:port name="CurrencyConvertorSoap" binding="tns:CurrencyConvertorSoap">
    <soap:address location="http://www.webservicex.net/CurrencyConvertor.asmx"/>
  </wsdl:port>
  ...
</wsdl:service>
</wsdl:definitions>
```

And the second one is provided by ServiceObjects[3]:

```
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.serviceobjects.com/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.serviceobjects.com/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.serviceobjects.com/">
    <s:element name="GetExchangeRate">
        <s:complexType>
```

[3] http://trial.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL

```
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ConvertFromCurrency" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="ConvertToCurrency" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="LicenseKey" type="s:string" />
        </s:sequence>
      </s:complexType>
  </s:element>
    <s:element name="GetExchangeRateResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetExchangeRateResult"
          type="tns:ExchangeRateResponse" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="ExchangeRateResponse">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="ConvertFromCurrency" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="ConvertToCurrency" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="ExchangeRate" type="s:double" />
        <s:element minOccurs="0" maxOccurs="1" name="Error" type="s:string" />
      </s:sequence>
    </s:complexType>
      ...
  </s:schema>
</wsdl:types>

<wsdl:message name="GetExchangeRateSoapIn">
  <wsdl:part name="parameters" element="tns:GetExchangeRate" />
</wsdl:message>
<wsdl:message name="GetExchangeRateSoapOut">
  <wsdl:part name="parameters" element="tns:GetExchangeRateResponse" />
</wsdl:message>
  .....
<wsdl:portType name="DOTSCurrencyExchangeSoap">
  <wsdl:operation name="GetExchangeRate">
    <wsdl:input message="tns:GetExchangeRateSoapIn" />
    <wsdl:output message="tns:GetExchangeRateSoapOut" />
  </wsdl:operation>
    ....
</wsdl:portType>
  ....
<wsdl:binding name="DOTSCurrencyExchangeSoap" type="tns:DOTSCurrencyExchangeSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetExchangeRate">
    <soap:operation soapAction="http://www.serviceobjects.com/GetExchangeRate" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
    ....
</wsdl:binding>
  .....
<wsdl:service name="DOTSCurrencyExchange">
  <wsdl:port name="DOTSCurrencyExchangeSoap" binding="tns:DOTSCurrencyExchangeSoap">
    <soap:address location="http://trial.serviceobjects.com/ce/CurrencyExchange.asmx" />
  </wsdl:port>
  ....
```

```
  </wsdl:service>
</wsdl:definitions>
```

3.3.3 Q1

A mashup for getting the maximum conversion rate of USD and Euro.

Solution in XQuery:

```
import module namespace ws1="http://www.serviceobjects.com/"
  at "http://trial.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL" ;
import module namespace ws2="http://www.webserviceX.NET/"
  at "http://www.webservicex.net/CurrencyConvertor.asmx?WSDL" ;

declare function local:max-exchange-rate($a as xs:string, $b as xs:string) as xs:double
{
  let $firstResultElement := ws1:GetExchangeRate(<GetExchangeRate xmlns="http://www.serviceobjects.com/">
                                                   <ConvertFromCurrency>{$a}</ConvertFromCurrency>
                                                   <ConvertToCurrency>{$b}</ConvertToCurrency>
                                                   <LicenseKey>WS30-TRY1-OHS2</LicenseKey>
                                                 </GetExchangeRate>)
  let $secondResultElement := ws2:ConversionRate(<ConversionRate xmlns="http://www.webserviceX.NET/">
                                                   <FromCurrency>{$a}</FromCurrency>
                                                   <ToCurrency>{$b}</ToCurrency>
                                                 </ConversionRate>)
  return max(($firstResultElement//ws1:ExchangeRate/text(),
    $secondResultElement//ws2:ConversionRateResult/text()))
};

local:max-exchange-rate("USD","CHF")
```

Expected Result: (for 12-06-2008)

1.04883

3.4 Low level SOAP-call

3.4.1 Description

In these cases, the programmer is in charge of both preparing the required payload (SOAP envelope) and resolving the returned result (SOAP envelope).

3.4.2 The Remote Web Service

Here we use the spelling-suggestion facility of the Google's web service. The relevant excerpt from the corresponding WSDL file[4] is as follow (note: all of the provided operations in this api need a GOOGLE_KEY which is a key generated for registered users):

```
<definitions name="GoogleSearch"
             targetNamespace="urn:GoogleSearch"
             xmlns:typens="urn:GoogleSearch"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    ....
  </types>
  ....
```

[4] http://api.google.com/GoogleSearch.wsdl

```
  <message name="doSpellingSuggestion">
    <part name="key"            type="xsd:string"/>
    <part name="phrase"         type="xsd:string"/>
  </message>

  <message name="doSpellingSuggestionResponse">
    <part name="return"         type="xsd:string"/>
  </message>
  ....
  <portType name="GoogleSearchPort">
    ....
    <operation name="doSpellingSuggestion">
      <input message="typens:doSpellingSuggestion"/>
      <output message="typens:doSpellingSuggestionResponse"/>
    </operation>
    ....
  </portType>

  <binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    ....
    <operation name="doSpellingSuggestion">
      <soap:operation soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
    .....
  </binding>

  <service name="GoogleSearchService">
    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
      <soap:address location="http://api.google.com/search/beta2"/>
    </port>
  </service>

</definitions>
```

3.4.3 Q1

using the the spell-checking facility of the Google's web service for getting the correct spelling of "atributte".

```
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'>GOOGLE_KEY</key>
        <phrase xsi:type='xsd:string'>atributte</phrase>
      </tns:doSpellingSuggestion>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

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

4 Built-in functions

We propose to extend the XQuery 1.0 and XPath 2.0 Functions and Operators document with two built-in functions for SOAP support. Those functions provide basic access to Web services data without the need to access and interpret WSDL descriptions. Those can also be used as part of an implementation of the Web Services import and export features described in the next sections. Those functions enable custom interaction with Web services and extensibility.

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

or with a more general signature

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

The proposed methods allows to specify the location, the HTTP method (GET/POST), custom soap headers and XML content (document or element node) and returns a document containing either the SOAP reply message or the SOAP error message.

5 XQuery Web Services

This section proposes extensions to XQuery that can be used to import Web services in queries, or export queries as Web services. The approach is similar to that of [4, 5]. The syntax has been chosen in such a way that web service import is treated as much as possible as regular XQuery module import. We follow two important design goals: 1) Existing XQuery modules can be used with little or no modification. 2) Importing and exporting a Web service should be as symmetric as possible. The syntax extensions are hints, so that an XQuery environment can override them when needed.

5.1 Importing an XQuery web service

This section describes changes to module imports used to import a Web service as an XQuery module.

One may decide to extend the XQuery grammar productions (i.e ModuleImport) with specific new productions in order to achieve this goal but other approaches with less modification to the language, are preffered. A nice solution is using *options* which are one of the extension mechanisms in XQuery.

Typically, a particular option will be recognized by some implementations and not by others. The syntax is designed so that option declarations can be successfully parsed by all implementations. There is no default namespace for options and each implementation recognizes an implementation-defined set of namespace URIs used to denote option declarations.

Since the module import production of XQuery 1.0 does not allow to specify options, it needs to be extended to accept an option list:

```
[23] ModuleImport ::= "import" "module"
                      ("namespace" NCName "=")? URILiteral
                      ("at" URILiteral ("," URILiteral)*)?
                      ("options" QName StringLiteral ("," QName StringLiteral) *)?
```

The following option are recognized:

Option name	Values	Meaning
fn:webservice	`true`	Indicate that this module import should be performed as web service import
fn:endpoint	xs:NMTOKEN	name of endpoint/port to use if multiple endpoints are available
fn:servicename	xs:NMTOKEN	name of service to use if multiple service are available

For instance, importing and invoking the maximum bid service from the use cases (see Section 3.1.4) works like this:

```
import module namespace exm="http://example.net" options fn:webservice "true";

exm:highest-bid("U04", 1002)
```

Since XML Schema declarations used in an XQuery module are not automatically imported, the user may have to also import the corresponding XML Schema used in the WSDL description for the Web services. Note that the corresponding physical schema may be located within the WSDL description itself.

The information which the additional parameters *service* and *endpoint* provide is needed since a WSDL file might contain multiple services and multiple bindings on these services. In other words, the URI to the WSDL itself may not be sufficient for importing a given service. If a WSDL contains such multiple definitions, and neither the parameters are given nor equivalent information inside the implementation, an error should be raised. Note that if a service has more than one *endpoint*, they must be imported separately.

The imported functions are considered as *non-deterministic*. This is consistent with the latest resolution for external functions in the current XQuery 1.1 Working Draft. Also, in the case of XQuery 1.1, only non side-effecting functions can be imported. It is the responsibility of the implementation to ensure that all services do not have any effects. In case the proposed feature is also included in the XQuery scripting facility working draft, this restriction can be lifted.

The following lists the errors that may occur during importing a web service (The list of all error codes are provided in Appendix A):

- No available WSDL (err:XQST0094)
- Invalid WSDL (err:XQST0095)
- Conflicting namespaces (err:XQST0047)
- Unspecified service name (err:XQST0096)
- Unspecified endpoint (err:XQST0097)

The following lists the errors that may occur when we calling one of the imported operators:

- Invalid SOAP server URL (err:XQST0098)
- Invalid SOAP response (err:XQST0099)
- Mismatch in number of parameters (err:XPST0017)
- Type error (err:FORG0006)
- Invocation Error (for all other errors arisen from distributed processing) (err:XQDY0101)

SOAP fault messages are being mapped to XQuery errors, as described in Section 6.1.1

5.2 Exporting an XQuery Module as a Web Service

This section describes changes to module declarations used to export an XQuery module as a Web service.

Like in the case of Web service import, we use XQuery *options* to declare the parameters necessary to export a module as web service, so no changes to any grammar production are necessary. The following example shows the how to use option declarations to set the parameters necessary to export an XQuery module.

```
module namespace exm="http://example.net";
declare option fn:webservice "true";
declare option fn:servicename "RelationalDataAccessService"
declare function exm:highest-bid($userid as xs:string, $itemno as xs:integer) as xs:double
{
max(doc("bids.xml")//bid_tuple[userid = $userid and itemno = $itemno]/bid)
}
```

A module can be optionally declared as a web service. Several (distinct) options can be provided to customize the export. In our proposed mapping, all functions of an XQuery module are provided through a single Web service, and through a single WSDL endpoint.

In all cases, the environment must provide all required information other than those which have been explicitly provided by user. An approach uses in many application runtimes is using a deployment descriptor which allows the user to expose ordinary modules as well.

The following lists the errors that may occur when running the query as a Web service:

- Invalid SOAP request (err:XQDY0100)
- Invocation Error (for all other errors arisen from distributed processing)(err:XQDY0101)

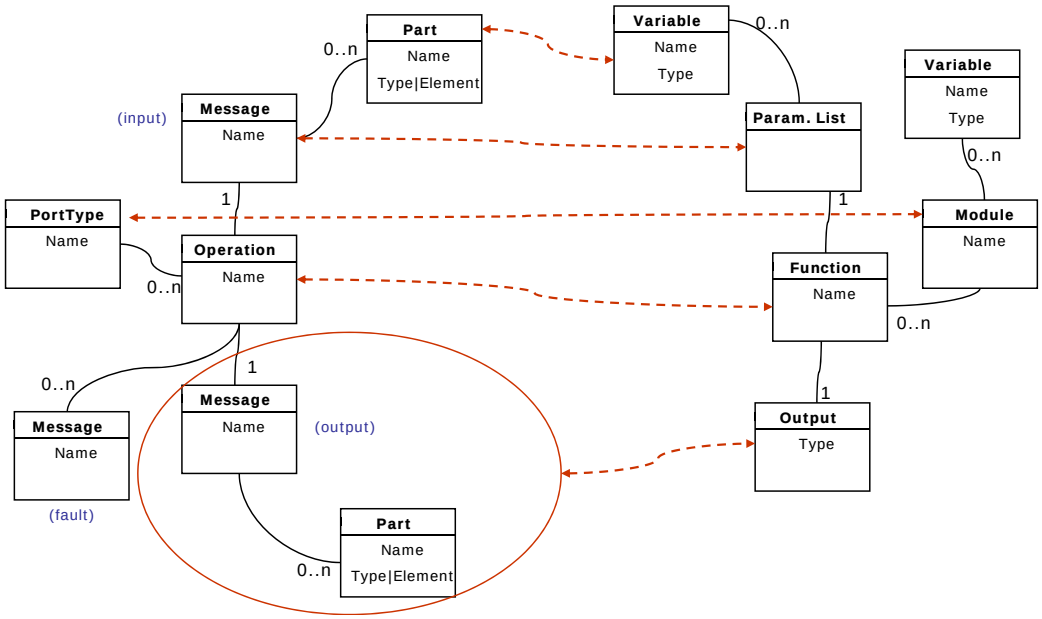

Figure 1: Binding between WSDL 1.1 portTypes and XQuery modules

Additionally, the validation of the SOAP messages/derived XDM instances (ingoing/outgoing) is implementation-dependent.

Type errors or dynamic errors identified during runtime are caught and serialized as SOAP fault messages, as described in Section 6.1.2.

The following option are recognized:

Option name	Values	Meaning
fn:webservice	`true`	Indicate that this module should be performed exported as web service
fn:endpoint	xs:NMTOKEN	name of endpoint/port to put into WSDL/contract
fn:servicename	xs:NMTOKEN	name of service to put into WSDL/contract
fn:uri	xs:anyURI	URI on which the service becomes available

6 WSDL-XQuery Binding

One of the most important steps toward providing web service facility in XQuery language is specifying a set of mapping rules between them. A binding between WSDL and XQuery is proposed in [4] which is based on correspondence of WSDL 1.1 [2] *portTypes* and XQuery 1.1 [1] *modules*. The key insight is that an *operation* input/output messages in the WSDL *portType* are in essence similar to the *function* signature in the XQuery *module*. Each *operation* behaves like a *function*, and each *input part* for the operation corresponds to a *parameter* of that function. Also, both are using XML Schema types to describe their input parameters and their output. There is no equivalent in WSDL for the function's body in XQuery. Instead, WSDL allows defining a binding, which gives some information about how to contact the service (here a SOAP server). Of course, the corresponding service must effectively accept calls according to the WSDL description and implement those operations.

Since the WSDL 1.1 [2] is the version currently most used and supported by existing tools, here we target this version and extend the Galax binding by adding a few rules. On the other hand, due to the importance of WSDL 2.0 [12] (it has reached the W3C recommendation status), as a part of our future work, we are planning to find a binding for that as well.

The overall scheme of this binding is shown in figure 1. Since most of this binding rules are straightforward, just the proposed binding between WSDL Parts and XQuery sequence types are described in more detail here. A WSDL Part could use its either *type* or *element* attribute to indentify its type. For the type mappings, there are two possible cases:

6.1 Schema-ful Mapping

In this case, we assume that the full-schema support is available. Here, depending on applied method we can map the Parts as follow [4]:

- $type = "AtomicType" == AtomicType$ i.e for 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>
```

  The resulting XQuery module would be something like:

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

- $type = "ListType" == AtomicType*$ [Where AtomicType is the atomic type from which the list type is derived] i.e this xquery module:

```
module namespace eth="http://www.ethz.ch/";
declare function eth:simpleInputSeqOutputFunc($a as xs:integer) as xs:integer* {($a - 1,$a,$a+1)};
```

  will be mapped to following description:

```
<definitions targetNamespace="http://www.ethz.ch/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ethz.ch/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/" >
    <types>
        <xs:schema  targetNamespace="http://www.ethz.ch/">
            <xs:simpleType name="newType">
                <xs:list itemType="xs:integer" />
            </xs:simpleType>
        </xs:schema>
    </types>

    <message name="simpleInputSeqOutputFunc">
```

```
        <part name="a" type="xs:integer"/>
    </message>
    <message name="simpleInputSeqOutputFuncResponse">
        <part name="return" type="tns:newType"/>
    </message>

    <portType name="port">
        <operation name="simpleInputSeqOutputFunc">
            <input name="input1" message="tns:simpleInputSeqOutputFunc"/>
            <output name="output1" message="tns:simpleInputSeqOutputFuncResponse"/>
        </operation>
    </portType>

    <binding name="binding" type="tns:port">
    ....
    </binding>
    <service name="SecondExample">
        <port name="SecondExamplePort" binding="tns:binding">
            .....
        </port>
    </service>
</definitions>
```

- $element = "QName" == element(QName)$ i.e for below XQuery module:

```
module namespace eth="http://www.ethz.ch/";
import schema namespace ipo = "http://www.example.com/IPO" at "ipo.xsd";
declare function eth:elementMappingFunc($a as element(ipo:purchaseOrder)) as xs:boolean{...};
```

generated WSDL would look like:

```
<definitions targetNamespace="http://www.ethz.ch/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ethz.ch/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ipo="http://www.example.com/IPO">
    <types>
        <xs:schema  targetNamespace="http://www.ethz.ch/" xmlns:ipo="http://www.example.com/IPO">
            <xs:import namespace="http://www.example.com/IPO" schemaLocation="..."/>
        </xs:schema>
    </types>
    <message name="elementMappingFunc">
        <part name="a" element="ipo:purchaseOrder"/>
    </message>
    <message name="elementMappingFuncResponse">
        <part name="return" type="xs:boolean"/>
    </message>
    <portType name="port">
        <operation name="elementMappingFunc">
            <input name="input1" message="tns:elementMappingFunc"/>
            <output name="output1" message="tns:elementMappingFuncResponse"/>
        </operation>
    </portType>
    <binding name="binding" type="tns:port">
        ....
    </binding>
    <service name="ThirdExample">
        <port name="ThirdExamplePort" binding="tns:binding">
            ....
        </port>
    </service>
</definitions>
```

It should be noted that there are some limitations in this binding that must be taken into account. In addition, since XQuery modules and WSDL portTypes do not correspond to each other completely, there are some parts in each side remained unmapped. Here, we present some ideas to cope with these limitations.

6.1.1 From WSDL to XQuery

- *Union types*: There is no equivalent for them in XQuery's sequence types. A possible solution for union of atomic types could be using the nearest common supertype of the involved types union (or in some situations, promoting the types).
  for example, following WSDL:

```
<definitions .... >
<types>
  <xsd:schema xmlns="http://www.w3.org/
  2001/XMLSchema"
  targetNamespace="http://www.ethz.ch/
  xmlns:tns="http://www.ethz.ch/">

    <xsd:simpleType name="Output">
      <xsd:union memberTypes="xs:integer xs:double"/>
    </xsd:simpleType>
</types>

<message name = "add">
  <part name="param1" type="xs:integer"/>
  <part name="param2" type="xs:integer"/>
</message>
<message name="addResponse">
  <part name="result" type= "tns:Output"/>
</message>

<portType name = "addPortType">
  <operation name="add">
    <input message="tns:add"/>
    <output message= "tns:addResponse"/>
  </operatiaon>
</portType>
...
<service name="FourthExample">
  <port name="FourthExamplePort" ... />
</service>

</definitions>
```

  would be mapped to:

```
module namespace ws = "http://www.ethz.ch/" ;

declare function ws:add($param1 as xs:integer, $param2 as xs:integer) as xs:double;
```

- *Complex types*: There is no equivalent for them in XQuery's sequence types. A possible solution could be enclosing these types in an element. In other words:
  $type = "ComplexType" == element(*, ComplexType)$

  for example, following WSDL:

```
<definitions .... >

<types>
  <xsd:schema xmlns="http://www.w3.org/
  2001/XMLSchema"
  targetNamespace="http://www.ethz.ch/
```

```
  xmlns:tns="http://www.ethz.ch/">

    <xsd:complexType name="Inputs">
      <xsd:sequence>
        <xsd:element name="param1" type="xs:integer"/>
        <xsd:element name="param2" type ="xs:integer"/>
      <xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="Output">
      <xsd:union memberTypes="xs:integer
      xs:double"/>
    </xsd:simpleType>
</types>

<message name="add">
  <part name="params" type="tns:Inputs"/>
</message>
<message name="addResponse">
  <part name="result" type="tns:Output"/>
</message>

<portType name="addPortType">
  <operation name="add">
    <input message="tns:add"/>
    <output message="tns:addResponse"/>
  </operatiaon>
</portType>
...
<service name="FifthExample">
  <port name="FifthExamplePort" ... />
</service>

</definitions>
```

would be mapped to:

```
module namespace ws = "http://www.ethz.ch/";
import schema namespace sch="http://www.ethz.ch/FifthExample";

declare function ws:add($param as element(*,sch:Inputs)) as element(*,sch:Output);
```

that the module imports the schema *FifthExample.xsd* which contains the schema that corresponds to the *types* element in the WSDL description[5]. Note than one may append the port name to the module's target namespace in order to deal properly with services that contain multiple ports.

- *Multiple-part output messages*: this limitation is imposed due to the fact that XQuery functions must have exactly one return type. To solve this problem, user can include these multiple parts as child nodes of a new XML element (i.e. $< multiple - result >$) in the same order as they are appeared in the WSDL. For example, suppose that we have this WSDL document:

```
<definitions
  targetNameSpace="http://www.ethz.ch/"
  xmlns:tns="http://www.ethz.ch/"
  .... >

<message name = "stat">
  <part name="param1" type="xs:integer"/>
  <part name="param2" type="xs:integer"/>
</message>
<message name = "statResponse">
  <part name="ave" type="xs:double"/>
```

[5]Often the target namespace of the WSDL and its type declaration part are same.

```
  <part name="var" type="xs:dobule"/>
</message>

<portType name="statPortType">
  <operation name="stat">
    <input message="tns:stat"/>
    <output message= "tns:statResponse"/>
  </operatiaon>
</portType>
...
<service name="SixthExample">
  <port name="SixthExamplePort" ... />
</service>

</definitions>
```

Then we should add the following complex type declaration to the schema which would be imported by the corresponding module:

```
<complexType name="MultiParts">
  <sequence>
    <element name="ave" type="xs:double">
    <element name="var" type="xs:double">
  </sequence>
</complexType>
```

and the module would be:

```
module namespace ws = "http://www.ethz.ch/";
import schema namespace sch="http://www.ethz.ch/SixthExample";

declare function ws:stat($param1 as xs:integer,$param2 as xs:integer) as
element(sch:muliple-result,sch:MultiParts)
```

This wrapping should be only applied to non-single results due to preserving the compatibility with importing an regular XQuery module.

- *Fault messages*: these messages are optional and include a single part which could be like other input and output parts. In order to facilitate working with these messages, an XQuery error is raised where the error object contains the contents of the fault message. The semantics are the same as if a call to $fn : error(WSErrorCode, desc, SOAPErrorMessage)$ would have occurred, where there are two ways to bind the values of $WSErrorCode$,*desc* and $SOAPErrorMessage$:
  1. If the format of the SOAP fault messages is the one outlined in Section 6.1.2, i.e., the called service complies to this proposal, the fault message is parsed and the values are filled in accordingly.
  2. In all other cases, $WSErrorCode$ is err:XQDY0101, *desc* is implementation-defined and $SOAPErrorMessage$ contains the contents of the SOAP fault message.

  The try/catch mechanism proposed in XQueryP/XQuery 1.1 provides a convenient way to recover from such errors.

6.1.2 From XQuery to WSDL

- *Sequence type occurrence indicators (?, +, and *)*: there is no any equivalent part in WSDL for these symbols of XQuery typing system. An approach to solve this problem is simulating these indicators in WSDL using *minOccurs* and *maxOccurs* facets. Actually, there has been a relation between these concepts initially[13]:

  > All XQuery values are sequences of length zero, one, or more. This rule arises from the XML Schema 'facets' name *minOccurs* and *maxOccurs*, which can be attached to a part of a schema in order to constrain its number of occurrences.

  For example, the following XQuery module

```
module namespace eth="http://www.ethz.ch/";

declare function eth:nodeSeqResFunc() as element()* {...};
```

will mapped to this WSDL

```
<definitions  targetNamespace="http://www.ethz.ch/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ethz.ch/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <xs:schema  targetNamespace="http://www.ethz.ch/">
            <xs:complexType name="newType">
                <xs:sequence  minOccurs="0" maxOccurs="unbounded" >
                    <xs:element name="baseType" type="xs:anyType" />
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
     </types>
     <message name="nodeSeqResFunc"/>
     <message name="nodeSeqResFuncResponse">
        <part name="return" type="tns:newType"/>
     </message>
     <portType name="port">
        <operation name="nodeSeqResFunc">
            <input name="input" message="tns:nodeSeqResFunc"/>
            <output name="output" message="tns:nodeSeqResFuncResponse"/>
        </operation>
     </portType>
    .....
</definitions>
```

This approach also conforms to the standard mapping of Java arrays to WSDL specified in JAX-RPC specification[14].

- *Item and Node types*: mapping these types to WSDL is still an open problem. For example, it's not clear that how should we map following XQuery function to WSDL:

  ```
  declare function fn:general($param1 as node()) as item()
  ```

- *Global variables*: since there is no equivalent for global variables of XQuery in the WSDL documents, we propose to provide some wrapper (setter and getter) functions for them in the generated WSDL. For example, assume that in our module, there is a variable delcaration like:

  ```
  module namespace eth="http://www.ethz.ch/";

  declare variable $eth:glob as xs:decimal;
  ```

  then there would be two new operations in the corresponding WSDL which look like:

  ```
  <definitions
    targetNameSpace="http://www.ethz.ch/"
    xmlns:tns="http://www.ethz.ch/"
    .... >

  <message name="getglob"/>
  <message name="getglobResponse">
      <part name="glob" type="xs:decimal"/>
  </message>
  <message name="setglob">
      <part name="glob" type="xs:decimal"/>
  </message>
  <message name="setglobResponse"/>

  <portType....>
    ....
    <operation name="getGlob">
  ```

```
    <input message="tns:getglob"/>
    <output message="tns:getglobResponse"/>
  </operation>
  <operation name="setGlob">
    <input message="tns:setglob"/>
    <output message="tns:setglobResponse"/>
  </operation>
  ...
<portType>
...
<definitions>
```

- *XQuery Errors*: they can be caught and serialized into SOAP fault messages. To simplifying accessing this information if the caller is also an XQuery/XPath 2.0/XSLT 2.0 implementation, the following serialization format is being used:

```
<error errNs="error code namespace" code=prefix:code description="...">
error-object, serialized in XML serialization...
</error>
```

6.2 Schema-less Mapping

Since support for schema import and user-defined types is an optional feature for XQuery, (and a significant number of implementations do not provide support for this feature), an alternative, less precise mapping is also provided by this proposal. This is especially relevant for import WSDL into XQuery, since Schema types are widely used in WSDL. When exporting XQuery modules that only use the built-in types, no schema-related aspects need to be considered.

The general idea is to use the closest possible ancestor type among the built-in types to represent a derived type. This approach can be further refined for these individual cases:

- Built-in types are used without any mapping
- User-defined simple types derived by restriction from a built-in type are mapped to this built-in type. (i.e xs:integer instead of myNS:hatsize which has been defined by restricting the xs:integer). This mapping can also be done transitively, so that types derived from type that is in turn derived from a built-in type are mapped to this built-in type.
- Attributes with user-defined type are mapped to attribute with a built-in type or xs:untypedAtomic
- Complex types definitions are mapped to element() without additional typing
- User-defined list and union types are mapped to xs:anySimpleType
- All other cases are mapped to "xs:anyType"

7 Open Issues

- As mentioned before, there are some problems with the mapping between XQuery and WSDL and they need to be addressed. In particular, how to map the XQuery's Items and Nodes to WSDL definitions?
- A more precise list of possible errors needs to be prepared.
- Sometimes, the programmer might want to use the loose-binding intentionally. One way to implement this possibility is introducing a new option as follows:

Option name	Values	Meaning
fn:loose-binding	`true`, `false`(Default)	Even if schema support is available, only use built-in types for parameters and result

- In the current specification of XQuery, one has to declare the meta-data information (like execution type of expression) at the level of module. Once we have some sort of function annotation, it would be possible to declare execution type at the level of functions which is more precise and efficient. Moreover, such annotation can be very useful toward the information hiding purpose (one can expose just a certain subset of all functions in a module). It seems that the concept of *module interface* is a nice way to provide these annotations.
  Another approach could be having some new options:

Option name	Values	Meaning
fn:deterministic	`true`, `false`(Default)	Treat all functions in the module as deterministic (or not)
fn:sequential	`true`, `false`(Default)	Treat all functions in the module as sequential, i.e. (or not)

- Some use cases of Web services seems to specify callback functions. Do we want to cater for this, and if yes, how should this be done?

References

[1] XQuery 1.0: An XML Query Language. [Online]. Available: http://www.w3.org/TR/xquery/

[2] Web Services Description Language (WSDL) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl

[3] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) W3C Recommendation 27 April 2007. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[4] N. Onose and J. Simeon, "XQuery at Your Web Service," in *Proceedings of the 13th International conference on World Wide Web*, New York, NY, USA, 2004, pp. 603 – 611.

[5] D. Graf, "XQueryP: Implementation and Formal Semantics," Master's thesis, ETH Zurich, Zurich, 2007.

[6] BEA Liquid Data. [Online]. Available: http://e-docs.bea.com/liquiddata/docs11/

[7] Stylus Studio. [Online]. Available: http://www.stylusstudio.com/

[8] Galax XQuery Engine. [Online]. Available: http://www.galaxquery.org/

[9] MXQuery XQuery Engine. [Online]. Available: http://www.mxquery.org/

[10] S. Graham, D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Koenig, and C. Zentner, *Building web services with java : making sense of XML, SOAP, WSDL and UDDI (2nd Ed.)*. Reading, MA: Sams, 2004, ch. 4.

[11] D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, and J. Robie, "Xml query use cases," Tech. Rep., 2007, http://www.w3.org/TR/xquery-use-cases/.

[12] Web Services Description Language (WSDL) 2.0. [Online]. Available: http://www.w3.org/TR/wsdl20/

[13] D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy, and P. Wadler, *XQuery from the Experts: A Guide to the W3C XML Query Language*, H. Katz, Ed. Addison-Wesley, 2003.

[14] JSR 101: Java APIs for XML based RPC. [Online]. Available: http://jcp.org/en/jsr/detail?id=101

A Error Codes

List of possible errors for web service scenarios:

Description	Code	Time
No WSDL available	err:XQST0094	Importing
Invalid WSDL	err:XQST0095	Importing
If multiple module imports in the same Prolog specify the same target namespace.	err:XQST0047	Importing
Unspecified serive name	err:XQST0096	Importing
Unspecified endpoint	err:XQST0097	Importing
Invalid SOAP server URL	err:XQDY0098	Runtime
Invalid SOAP respone	err:XQDY0099	Runtime
Invalid SOAP request	err:XQDY0100	Runtime
If the expanded QName and number of arguments in a function call do not match the name and arity	err:XPST0017	Runtime
Invalid argument type	err:FORG0006	Runtime
Invocation Error	err:XQDY0101	Runtime