Yoo-Hoo!
Building a Presence Service with XQuery and WSDL

Mary Fernández
AT&T Labs - Research
180 Park Ave
Florham Park, NJ 07932, USA
mff@research.att.com

Nicola Onose
Ensimag
BP 72
38402 Saint Martin d'Hères
Cedex, France
nicola.onose@ensimag.imag.fr

Richard Hull
Jérôme Siméon
Bell Labs, Lucent
Technologies
600 Mountain Avenue
Murray Hill, 07974, NJ, USA
{hull,simeon}@lucent.com

1. INTRODUCTION

XML is at the heart of Web services: It is used as the format for the messages exchanged between services and applications (using SOAP [8]), to describe the structure of those messages (using XML Schema [9, 6]), and to describe Web services interfaces (using WSDL [10, 11]). Yet current approaches [4] to Web-services development hide the XML layer behind Java or C# APIs, preventing the application to get direct access to the original Web service data and interface. The goal of the XButler[1] project is to support faster and more reliable Web service development by enabling the use of XQuery as an integral part of the Web service infrastructure. We propose to demonstrate XButler by installing a Web service which integrates presence information from multiple service providers.

The main technical contribution behind the XButler approach is a binding between WSDL [10], the Web Services Description Language, and XQuery [12], the W3C XML query language. In particular, we extends XQuery with an `import service` statement, which provides transparent access to Web services from within an XQuery program. Conversely, XButler provides a command-line tool (called `xquery2soap`) which takes an XQuery module and deploys it as a SOAP service. The XButler tool-set and the corresponding XQuery extensions are implemented on top of the Galax[2] [1] XQuery processor. Because of the parallel structure of WSDL definitions and XQuery modules, implementing XButler required only minimal changes to Galax. More details about the underlying WSDL-XQuery binding, and a description of how to implement the corresponding extensions can be found in [7].

To demonstrate the effectiveness of the XButler approach, we use it to deploy a user-presence service, called Yoo-Hoo![3]. Yoo-Hoo! integrates user-presence information from multiple service providers and can provide Yoo-Hoo! clients with information such as "the requested subscriber is on-line at Jabber" or "the subscriber's cell phone is busy". As we will see, XButler dramatically simplifies the implementation and deployment of the Yoo-Hoo! service – the client and server can manipulate XML messages directly in XQuery and all low-level handling of SOAP messages is entirely transparent.

[1] `http://db.bell-labs.com/xbutler`
[2] `http://db.bell-labs.com/galax`
[3] An American colloquialism used to attract attention or as a call to persons.

2. THE YOO-HOO! PRESENCE SERVICE

2.1 Architecture

The architecture of the Yoo-Hoo! Web service is shown on Figure 1. On the right-hand side, instant messaging (IM) services (such as Jabber, AOL-IM, and MSN) and wireless service providers make presence information available to subscribed users. Typical IM services permit their subscribers to announce their availability (e.g., ready to chat, temporarily away, online but do not disturb) and check other users' presence, while telephony providers might indicate whether a subscriber is on their home, cell, or work phone. On the left-hand side, various instant-messenger clients, cell-phones or hand-held devices need access to user presence information from those various providers.

Our goal is to deploy the integrated Yoo-Hoo! service shown in the middle of Figure 1. The service relies upon a traditional data integration architecture, but operating here over Web services. On the one hand, Yoo-Hoo! acts as a service accessible to the instant-messenger clients or devices, and is described using WSDL **(1a)**. On the other hand, Yoo-Hoo! acts as a client to access the corresponding provider-specific presence informations, available as Web services and described using WSDL **(1b)**.

Concretely, all client and services applications must implement a messaging layer that handles the generation and manipulation of SOAP messages transfered over the network. In addition, the Yoo-Hoo! service must implement the application logic that handles the XML values sent to him by its client, dispatch those requests to the underlying provider services, compute the results for the requests and send them back to the clients. In our scenario, XQuery is ideal for both dealing with the messaging layer and accessing the various services. We implement part of the GAIM client in XQuery (**2a**) by using the XButler extension to import the Yoo-Hoo! service as though it were an XQuery module (i.e., a set of functions and global variables). From the `import statement`, XButler generates all the "stub" code for handling the low-level SOAP mes-

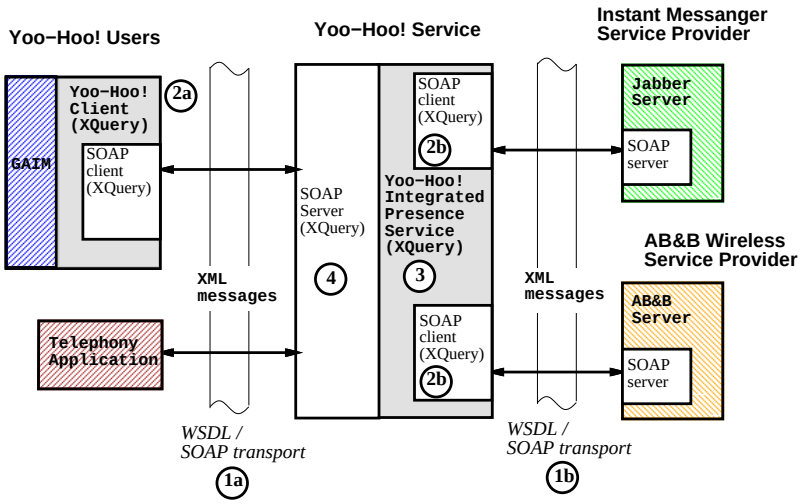

Figure 1: Yoo-Hoo! Integrated Presence Web Service

sages required by the Yoo-Hoo! server using XQuery itself. Then, the client application can invoke the Yoo-Hoo! service by simply calling XQuery functions – all of the low-level messaging code is entirely transparent.

As for the client, using XQuery can greatly simplify the server implementation. The Yoo-Hoo! service itself is implemented as an XQuery module (**3**). From this module definition, XButler generates the "stub" code for handling the low-level SOAP messages and dispatching incoming messages to the appropriate XQuery functions (**4**). For this application, XQuery is powerful enough to implement all of the Yoo-Hoo! service logic as well, although a more complex service might require the additional support of a general-purpose programming language.

Finally, the Yoo-Hoo! service can use XButler's import-service feature to transparently access the operations of multiple service providers (**2b** by simply calling the corresponding XQuery functions.

We now illustrate in detail how to build the Yoo-Hoo! presence service using XQuery and WSDL. We assume some knowledge of XML, XQuery and Web services and refer the reader to introductions on these subjects [3, 5, 10].

2.2 Describing the Yoo-Hoo! service

Figure 2 contains (a small part of) the WSDL 1.1 definition for the Yoo-Hoo! presence service. For our purpose, the most important component in the WSDL description is the `portType`, which describes `operations` supported by the service. Each operation is described by an input and and output `message`, whose parameters can be specified using XML Schema `types`.

In our example, the *Yoo-Hoo!* service provides one operation, *presence*, which takes as input a user's Yoo-Hoo! identifier (a string) and returns their presence information in an `yh:presence` element, which contains the user's status and location.

2.3 Importing WSDL Services in XQuery

XButler extends XQuery with an `import service` statement which allows to access Web services from within an XQuery program. Importing a service is very simple: one just needs to identify the WSDL resource, the name of the service, and the port which must be accessed. The WSDL target namespace may be bound to a prefix for use within the rest of the query. For instance, the following statement imports the Yoo-Hoo! service, and can be used to implement part of the GAIM client (`(2a)` on Figure 1).

```
import service namespace yh = "http://YooHoo.net/"
  name "YooHoo"

let $b := //buddies[name = "Elena Buchsbaum"]
for $s in yh:presence(b/@yhid)/service return
return
  let $presence :=
    if ($s[online|chat]) then "Available :)"
    else "Unavailable :("
  return
    fn:concat($presence, " on: ",$s/location)
```

Once the service is imported, the operations from that service are available to the query *as standard XQuery functions*. In our example, the imported Web service operation is used to retrieve the presence information for the user `"Elena Buchsbaum"`. Under the hood, calling the XQuery function `yh:presence` triggers a SOAP call to the Yoo-Hoo! service through the SOAP client stub. The input parameters from the function call are passed to the corresponding Web service operation. Once the service returns a result for that call, this result is passed back to the XQuery function and the query evaluation may proceed.

Because the values manipulated by XQuery and the SOAP messages are both in XML, the user does not need to perform any conversion of the input parameters (resp. of the result) before (resp. after) calling the service. Note also that the approach also imports the type for the operations as XQuery function signature, enabling the use of XQuery's static typing feature. See [7] for more details.

2.4 Implementing the service logic

The import service statement makes access to multiple Web services and their composition easy. As a result, we can easily use it to access the various service providers, as required to implement

```
<definitions
  targetNamespace="http://YooHoo.net"
  xmlns:yh="http://YooHoo.net" ...>
  <types>
   <xs:schema targetNamespace="http://YooHoo.net">
    <xs:element name="presence">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="service">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="location" type="xs:string"/>
          <xs:choice>
            <xs:element name="online" type="xs:string"/>
            <xs:element name="offline" type="xs:string"/>
            <xs:element name="chat" type="xs:string"/>
          </xs:choice>
         </xs:sequence>
        </xs:complexType>
       </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:schema>
  </types>

  <message name="User">
   <part name="user" type="xs:string"/>
  </message>

  <message name="Presence">
    <part name="result" element="yh:presence"/>
  </message>

  <portType name="PresencePort">
   <operation name="presence">
    <input message="yh:User"/>
    <output message="yh:Presence"/>
   </operation>
  </portType>

  <binding name="PresenceSOAP" type="yh:PresencePort">
    ...
  </binding>

  <service name="YooHoo">...</service>
</definitions>
```

Figure 2: (A part of) the Yoo-Hoo! WSDL definition

the service logic (**(3)** on Figure 1). The following XQuery program imports operations from both service providers and composes those operations to retrieve the presence information for the requested user.

```
module namespace yh = "http://YooHoo.net/YooHoo"

import service namespace im = "..." port "IM"
import service namespace jabber=".." port "Jabber"

define function yh:presence($user as xs:string)
  as element(yh:presence) {
    <presence>
     <service>
      <location>IM</location>
      { im:status($user) }
     </service>
     <service>
      <location>Jabber</location>
      { jabber:onlineinfo($user) }
     </service>
    </presence>
  }
```

Note that this code implements the function `yh:presence` which was described in the WSDL for the Yoo-Hoo! service on Figure 2.

2.5 Deploying an XQuery module as a service

Finally, we need to install the Yoo-Hoo! Web service. XButler provides a command-line tool called `xquery2soap` which can performs the necessary system installation on a SOAP-Apache server[4]. The Yoo-Hoo! service can be deploying from the previous XQuery module as follows.

```
xquery2soap
 -installdir "/var/www/services"
 -interfacedir "/var/www/services/wsdl"
 -address "http://YooHoo.net/YooHoo.xqs"
 YooHoo.xq
```

As a result of this call, a WSDL file similar to that in Figure 2 is generated. From that WSDL file, an XQuery SOAP server is built and copied along with *YooHoo.xq* to the location of the Apache service directory (here " `/var/www/services`"). In addition, the tool generates the appropriate stub (denoted ' `YooHoo.xqs`') between the Apache server and the XQuery module. Compared to the stub generated in the `import service` case, this one plays the inverse role: it extracts the parameters from the SOAP messages and passes them to the appropriate XQuery function in the XQuery module. The result of the XQuery call is then wrapped back again as a SOAP message, which is sent back as a result to the calling application.

3. THE DEMONSTRATION

We intend to present the following at the conference.

- We will demonstrate a complete running YooHoo! service integrating two different Jabber [2] servers, one representing an IM service, and one simulating a wireless service provider.
- We will demonstrate both the `import service` extension to XQuery and the `xquery2soap` command-line tool. We will also demonstrate the use of static typing over XQuery program including Web service calls.
- We will show all of the user-defined, as well as XButler-generated XQuery code used to implement the service and deploy the service from scratch in real time.
- As a comparison point, we propose also to show a similar service implemented using standard Java and C# Web service development tools.

4. REFERENCES

[1] M. Fernandez, J. Siméon, B. Choi, A. Marian, and G. Sur. Implementing XQuery 1.0: The Galax experience. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 1077–1080, Berlin, Germany, Sept. 2003.

[2] Jabber. `http://www.jabber.org/`.

[3] P. B. James McGovern, K. Cagle, J. Linn, and V. Nagarajan. *XQuery Kick Start*. SAMS, 2003.

[4] Java architecture for XML binding (JAXB). `http://java.sun.com/xml/jaxb/`.

[5] H. Katz, editor. *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison-Wesley, 2003.

[4]We decided to support apache first, but the approach should easily extend to other http servers.

[6] M. Maloney and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, May 2001.

[7] N. Onose and J. Siméon. XQuery at your Web service. Unpublished manuscript, Aug. 2003. `http://db.bell-labs.com/galax/xbutler.ps`.

[8] SOAP version 1.2 part 0: Messaging framework. W3C Recommendation, June 2003.

[9] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. W3C Recommendation, May 2001.

[10] Web services description language (wsdl) 1.1. W3C Note, Mar. 2001.

[11] Web services description language (WSDL) version 2.0 part 1: Core. W3C Working Draft, Mar. 2003.

[12] XQuery 1.0: An XML query language. W3C Working Draft, Nov. 2003.