Java ME Clients for XML Web Services

Paul POCATILU
Academy of Economic Studies, Bucharest

Using Web services in developing applications has many advantages like the existence of standards, multiple software platforms that support them, and many areas of usage. These advantages derive from the XML and Web technologies. This paper describes the stages in the development of a Web service client for Java ME platform and presents examples based on kSOAP and JSR 172.

Keywords: *Web services, mobile applications, Java ME, SOAP, XML, m-learning.*

1 Introduction

Web services are programmable units that provide specific functionalities and are accessible to a large number of consumers using Internet protocols. XML Web services are methods for stored objects on the server to accept requests from the clients using XML, usually over the HTTP protocol. The requests from the clients are received using a transport protocol (HTTP, TCP/IP, SMTP etc). The response from the server is received by the client using the same encoding and transport protocol, figure 1.

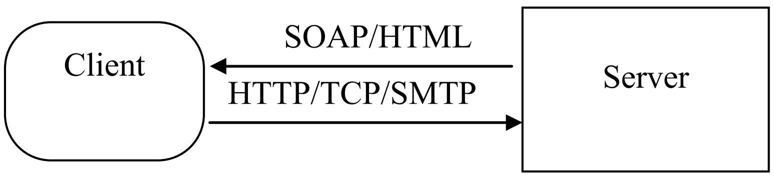


Fig.1. Web Services

The supported requests types for the Web services are (but not limited to) HTTP GET, HTTP POST and SOAP. The Web services consumers are the client applications that access the Web services interface methods initiated through TCP/IP or UDP protocols.

Developing and using XML Web services can be done using many programming languages and for various clients (mobile, desktop, Web etc). Mobile applications are used on various domains and their use is growing each day and these can be easily used as Web services clients. Using the description of the Web service from a server (based on WDSL and UDDI), a proxy server is generated for an easier development.

In this paper the Java ME technologies for XML Web services are briefly presented and examples are provided for Java ME clients for a XML Web service that performs English to Romanian and Romanian to English translations and is implemented using .NET technologies.

This Dictionary Web Service is seen as a starting point for an e-learning and m-learning platform using desktop and mobile clients.

Sun's Java 2 Micro Edition platform runs on many mobile devices, which have installed a Java Virtual Machine. The biggest benefit of using the Java platform for mobile device development is that is possible to produce portable code that can run on multiple platforms. It is almost impossible to port the complete functionalities of an application to all mobile devices because wireless devices have a vast range of capabilities in terms of memory, processing power, battery life, display size, and network bandwidth.

Java ME is divided into several different configurations and profiles. Configurations contain Java language core libraries for a category of devices. In this moment there are two configurations:

▪ Connected Limited Device Configuration (CLDC) – designed for small, resourceconstrained devices (cell phones, low-end PDAs);

▪ Connected Device Configuration (CDC) – designed for relatively big and powerful devices (high-end PDAs, set-top boxes, network appliances); CDC has more capabilities than CLDC in terms of security, mathematical, and I/O functions.

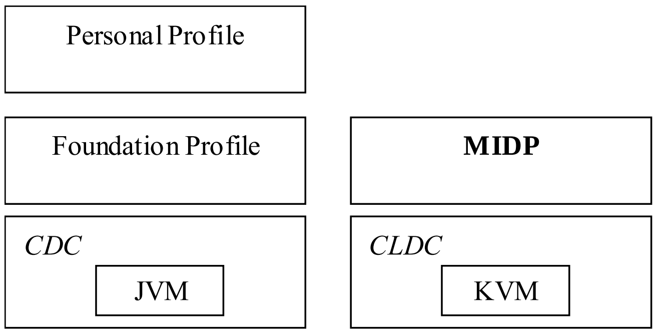


Fig.2. Java 2 Micro Edition platform

On top of each configuration are several profiles. A profile defines device-specific API libraries, like GUI, networking, and persistent-storage APIs. Each profile has its own runtime environment and is suited for a range of similar devices.

The two major profiles for the CLDC are Mobile Information Device Profile (MIDP) and PDA Profile. For the CDC there are two important profiles: the Foundation Profile and the Personal Profile (figure 2).

In [POCA02] is presented a Web service *Dictionary* developed using .NET and C# language. This Web service has two public methods: *Tradu* (translate a word from Romanian to English) and *Translate* (translate a word from English to Romanian). The service is published at the http://pocatilu.ase.ro/dictionary/dservice.asmx. DService supports all protocols for communication (HTTP GET and POST and SOAP). The supported protocols are obtained through SDL (Service Descriptor Language) that is based on XML.

Consuming Web Services from Java ME applications can be done using at least two approaches: WSA (JSR 172) or kSOAP library.

JSR 172 defines *the Web Services API* (WSA) and extends the web services platform to include Java ME. J2ME Web Services APIs enable J2ME devices to be web services clients, providing a programming model that is very similar with the standard Web services platform.

If the mobile device does not implement this optional standard (WSA), kSOAP library is another solution to develop Web services clients for Java ME platform.

In order to develop Java ME Web services clients, the Dictionary Web Service is used for test, published at http://pocatilu.ase.ro/dictionary/dservice.asmx. NetBeans IDE with Mobility Pack (includes Sun’s Java Wireless Toolkit) was used to develop the applications.

2. Consuming Web services using kSOAP

In order to develop Web services clients for Java ME applications using kSOAP 2, the libraries need to be added to the project (downloadable from the project’s site http://ksoap2.sourceforge.net/). This library is based on the SOAP architecture and there is no need to generate a proxy/stub to call Web services methods.

A SOAP envelope is created using the **`SoapSerializationEnvelope`** class (specifying the SOAP version) and the request details are added to the envelope body (using **`SoapObject`** class). The **`HttpTransport`** class is used to make the actual call of the Web service method, the envelope being passed as parameter. The result is retrieving from the response part of the envelope.

Figure 3 shows the code that initializes the SOAP envelope, make the call and retrieve the result.

```
try {
      //creare request, transport and envelope
      SoapObject request = new SoapObject(namespace, metoda);
      HttpTransport ht = new HttpTransport(url);
      SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
      //adaugare parametru + valoare
      request.addProperty(parametru, parent.getTfCuvint().getString());
      envelope.setOutputSoapObject(request);
      envelope.dotNet = true;
      ht.debug = true;

      //apel metoda serviciu Web
      ht.call(namespace + metoda, envelope);
      //preluare rezultat
      SoapPrimitive result = (SoapPrimitive)envelope.getResponse();
      //afisare rezultat
      parent.getTfTraducere().setString(result.toString());

     }
     catch(Exception ex)
     {
      parent.getTfTraducere().setString("Exceptie: " + ex.getMessage() + "
" + ex.getClass().toString());
      }
```

Fig.3. *Translate* method call code of Web Dictionary service using kSOAP 2

It is very important to send the right parameters to the methods. In this example, the parameters used in these calls are initialized as follows:

• `String namespace = "http://tempuri.org/"`; – Web service namespace; in this example the default namespace is used

• `String url = "http://pocatilu.ase.ro/dictionary/DService.asmx"`; – Web service URL

• `String metoda = "Translate"`; – Web service method name

`String parametru = "word"`; – the name of the Translate method parameter. Using wrong values for these parameters will lead to Web service method call to fail.

Using wrong values for these parameters will lead to Web service method call to fail.

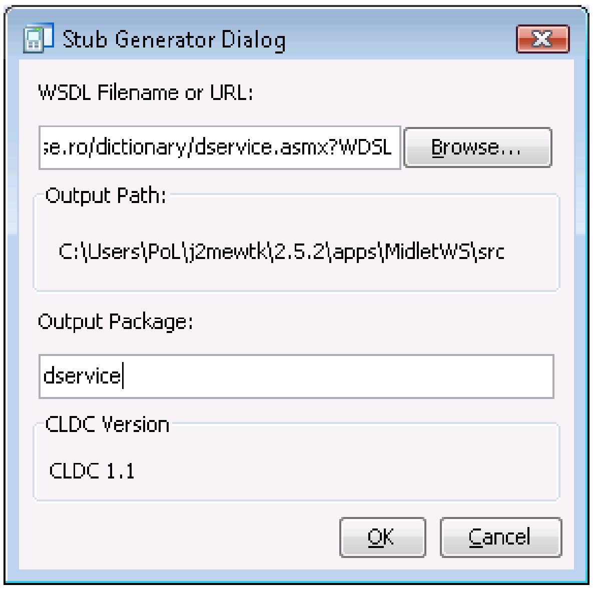
kSOAP library includes kXML package, a XML parser, also provided as an open project (http://kxml.sourceforge.net).

```
try
{
      //creare request, transport and envelope

      DServiceSoap_Stub client = new DServiceSoap_Stub();
      //apel metoda serviciu Web si preluare rezultat
      String result = client.translate(parent.getTfCuvint().getString());
      //afisare rezultat
      parent.getTfTraducere().setString(result);

}
catch(Exception ex)
{
     parent.getTfTraducere().setString("Exceptie: " + ex.getMessage() + " "
+ ex.getClass().toString());
}
```

Fig.5. *Translate* method call code of Web Dictionary service using WSA

3. JSR 172 standard for Web services

If the Java ME platform implements the JSR 172 standard on the mobile device, then the WSA can be used to develop Web services clients.


Stub Generator Dialog

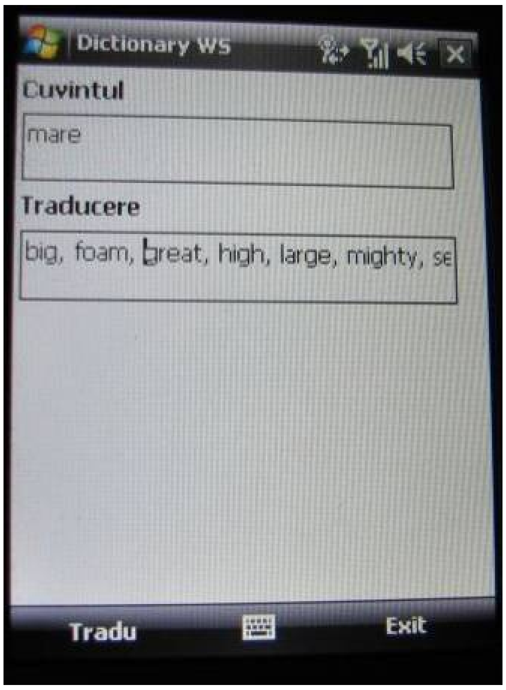
WSDL Filename or URL: se.ro/dictionary/dservice.asmx?WDSL

Output Path: C:\Users\PoL\j2mewtk\2.5.2\apps\MidletWS\src

Output Package: dservice

CLDC Version: CLDC 1.1

Fig.4. WTK's Stub Generator Dialog


Dictionary WS

Cuvintul: mare

Traducere: big, foam, breat, high, large, mighty, se

Fig.6. Dictionary Web Service client running

First step is to generate stub classes for the Web service. This can be done easily using Sun Java WTK. Figure 4 shows the Stub Generator Dialog that generates the Java classes for the Web service proxy. The URL to the WDSL description of the Web service (http://pocatilu.ase.ro/dictionary/dservice.asmx?WDSL) and the name of the output package are passed.

Using this tool, several Java classes are generated by the JAXRPC SI and 172 StubGenerator. In this application `DServiceSoap_Stub` it is used to instantiate the client and to make the call to the Web service method, as an ordinary Java class method. Figure 5 shows the code used in application to call the *Translate* method of the Dictionary Web Service.

All these calls were integrated in Java ME Midlet, and the result is shown in figure 6 (using WTK's emulator).

In order to keep the user interface responsive and not to block it during the network calls, all the Web methods calls runs in a different execution thread.

4. Conclusions and Future Work

Using XML Web Services has many advantages for developers and for users: they use simple protocols and the implementation of the services and clients is easier than other methods. The client application, for example, could be based on Windows Forms, a native application, a Java ME Midlet, Web based etc.

This paper shows how easy Web services client applications can be developed using Sun's NetBeans for Java ME applications. The developer can better focus on the application design and business logic, than to the connections with the Web services.

The interface of these is very light, and it will be improved, adding new functionalities and provide a basis for m-learning applications. Also, the Dictionary Web Service will become more complex, by adding other methods and new functionalities and can be used as a framework for e-learning applications.

References

[FOX02] Dan Fox, Jon Box – *Building Solutions with the Microsoft .NET Compact Framework: Architecture and Best Practices for Mobile Development*, Addison Wesley, ISBN : 0-321-19788-7, 2003

[HASH05] Sayed Y. Hashimi, Scott J. Steffan – *Pro Service-Oriented Smart Clients with .NET 2.0*, Apress, ISBN 1-59059-551-3, 2005

[PLAT01] David Platt – *Building reusable Web Components with SOAP and ASP.NET*, in MSDN Magazine, February 2001, Vol. 16, No 2, pp. 100-114.

[POCA02] Paul Pocatilu – *Dictionary Web Service*, in Economy Informatics, vol. III, nr. 1, 2003, pp. 119-122,

[POCA04] Paul Pocatilu, Cristian Toma - Dezvoltarea aplicaţiilor mobile în Java, în Informatica Economica, vol. VIII, nr. 3(31), 2004

[POCA04a] Paul Pocatilu – *Dezvoltarea clientilor Java pentru servicii Web XML*, în Informatica Economica, vol. VIII, nr. 4(32), 2004, pp. 68-71

[POCA06] Paul POCATILU - *Consuming Web Services From .NET Compact Framework Applications,* Proceedings of the 6th Biennial International Symposium SIMPEC 2006, vol 2, Braşov, 26-27 Mai 2006, pp. 288-292

[RUBI03] Erik Rubin, Ronnie Yates – *Microsoft .NET Compact Framework Kick Start*, Sams Publishing, ISBN 0-672-32570-5, 2003

[WEI03] Lee Wei-Meng – *Developing Mobile Applications Using the Microsoft Mobile Internet Toolkit*, http://www.devx.com/wireless/Article/10148 , 2003

[WIGL03] Andy Wigley, Stephen Wheelwright – *Microsoft .NET Compact Framework* , Microsoft Press, ISBN 0-7356-1725-2, 2003

[YAO04] Paul Yao, David Durant –*.NET Compact Framework Programming with Visual Basic .NET*, Addison Wesley, ISBN 0-321-17404-6, 2004

[*****] http://sun.java.com/javame