How to access 500,000 spectra via XML Web Services

Version 1.0

IVOA Working Draft 2003-10-01

This version:
http://www.ivoa.net/internal/IVOA/WebgridTutorial/SpectrumWSTutorial-v1.0.doc

Latest version:
http://www.ivoa.net/internal/IVOA/WebgridTutorial/SpectrumWSTutorial-v1.0.doc

Previous version:
http://www.ivoa.net/internal/IVOA/WebgridTutorial/SpectrumWSTutorial-v0.9.doc

Editor:
Alex Szalay

Authors:
Tamás Budavári,
László Dobos,
Wil O'Mullane

Interoperability and WebServices Working Groups

Abstract

We present a brief tutorial on how to consume XML Web Services. This step-by-step introduction uses VO services that publish close to 500,000 spectra (SDSS DR1, 2dFGRS). The examples are written in C# using the .NET Framework as well as in Java using the Axis library to demonstrate the interoperability.

Status of this document

This is a Working Draft. The first release of this document was 11 Oct 2003.

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress." A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/docs/.

Acknowledgments

Please give credit to all major contributors to the document.

Contents

1 Introduction

XML Web Services [1] are becoming an industry standard and provide the first truly interoperable solution for distributed computing. Many of the problems in the Virtual Observatory [2,3] map very well to this new emerging paradigm [Szalay et al., 2002]. One example is the astronomical archives providing easy access to scientific data not only on a human readable web page but also directly from programming languages. Web Services are essentially programming library modules living on the Web.

As of the writing of this paper, there are two fully functional implementations of the Web Services specifications, namely Microsoft's .NET Framework [4] and Java Axis [5]. The authors have tested both toolkits, freely downloadable and successfully used services written in C# from Java and vice versa. The tutorials presented in this paper show how to consume services from both C# and Java. For details on how to set up Java Axis or the .NET Framework please go the developer's web site or visit us at http://skyservice.pha.jhu.edu/develop.

2 Using Web Services

Even though all Web Services are different, consuming them always follows the exact same pattern. Here we demonstrate the basic features by consuming the spectrum web services [6] developed at the Johns Hopkins University.

2.1 The proxy class

Once we decided to use a particular service, its formal description needs to be located. This is called the WSDL file, which is an XML document (not meant to be human readable) and usually published on the web via http. In our case this is

http://skyservice.pha.jhu.edu/devel/wave/wave.asmx?WSDL

From this formal description that fully describes the services in a machine-readable format our toolkit can generate the so-called *proxy* class or *stub*. This may be achieved by executing a single command at the prompt or inside the developing environment. The command in the .Net Framework is

```
C:\> wsdl.exe http://…
```

and in Java

```
C:\> java org.apache.axis.wsdl.WSDL2Java http://…
```

The above programs will automatically create the necessary source files in C# and Java that may be used to have direct access to the published methods as if they were actually part of a local library.

The developer environments, e.g. Visual Studio .NET, may offer the possibility of adding a Web Reference to the project. This in fact does the same thing as the above command but even more convenient to use.

In java you need to set your class path properly such as:

```
set ALIB=C:\java\axis-1_1RC2\lib\
set CLASSPATH=.;c:\wil\java;%ALIB%\axis-
ant.jar;%ALIB%\axis.jar;%ALIB%\commons-
discovery.jar;%ALIB%\commonslogging.jar;%ALIB%\saaj.jar;%ALIB%\axis
.jar;%ALIB%\jaxrpc.jar;%ALIB%\wsdl4j.jar;%ALIB%\log4j-1.2.4.jar;
```

2.2 Calling the services

Now, we need to add the auto-generated code to our project by including the necessary namespace. In C# we can simply use the `using` keyword, in Java we use `import` as in the following examples.

```
using myapp.edu.jhu.pha.skyservice;     // C#
```

```
import edu.jhu.pha.skyservice.wave.*;   // Java
import edu.jhu.pha.skyservice.*;
```

First, we instantiate the proxy class that has been auto-generated from the WSDL file and is found in the above namespace, then simply call its methods. Every time one calls a method on the (local) proxy class, posts a SOAP message to the Web Service, retrieves the reply message and de-serialize the XML document into the data structure previously derived from the WSDL file. It is all very simple in practice. The codes below make several calls to the spectrum service at the Johns Hopkins University.

Here is the pseudo code of the C# and Java programs to follow:

1. Create proxy class and print the URL of the service
2. Get a single spectrum given by its ID
3. Get a spectrum in VOTable format – Note that VOTable is also a class!
4. Fetch all spectra in a given pointing, a.k.a. ConeSearch
5. Fetch spectra in a given redshift range (z>5 yields SDSS QSOs)
6. Build the high-z QSO composite spectrum

2.2.1 Listings in C#

```
// Proxy class and the URL of the service
Wave p = new Wave();
Console.WriteLine(p.Url);

// Retrieve a spectrum by its ID
Spectrum s = p.GetSpectrum(200000,true);
Console.WriteLine(s.Name);
for (int i=0; i<10; i++)
   Console.WriteLine
      ("\t"+s.Points[i].Wavelength+" "+s.Points[i].Value);

// Same in VOTable format
VOTABLE v = p.GetSpectrumVoTable(200000,false);
Console.Write(v.DESCRIPTION+": ");
Console.WriteLine(v.RESOURCE[0].TABLE[0].DATA.TABLEDATA[0][1].Value);

// ConeSearch returns spectra in a virtual observational field
Spectrum[] sa = p.FindSpectraCone(180,0,5,false);
Console.WriteLine("# of objects found: "+sa.Length);
foreach (Spectrum c in sa)
   Console.WriteLine("\t"+c.Ra+"\t"+c.Dec);

// High redshift search yields QSOs
Spectrum[] qa = p.FindSpectraRedshift(5,10,false);
Console.WriteLine("Z>5 (qso)");
foreach (Spectrum q in qa)
   Console.WriteLine("\t"+q.Name+"\t"+q.Z);

// Derive QSO composite by calling another web method
s = p.ComposeAverageSpectrum(sa, SpectrumWavelengthScale.Linear,1000,
           new float[]{1430},new float[]{1480},1.4f,1e-4f);
Console.WriteLine(s.Name);
```

```
for (int i=0; i<s.Points.Length; i++)
   if (s.Points[i].Wavelength>1200 && s.Points[i].Wavelength<1230)
      Console.WriteLine
         ("\t"+s.Points[i].Wavelength+" "+s.Points[i].Value);
```

2.2.2 Listings in Java

```
// Proxy class and the URL of the service
WaveLocator wl = new WaveLocator();
System.out.println(wl.getWaveSoapAddress());
WaveSoap p = wl.getWaveSoap();

// Retrieve a spectrum by its ID
Spectrum s = p.getSpectrum(200000,true);
System.out.println(s.getName());
for (int i=0; i<10; i++)
   System.out.println
      ("\t"+s.getPoints().getPoint()[i].getWavelength()+"
"+s.getPoints().getPoint()[i].getValue());

// Same in VOTable format
VOTABLE v = p.getSpectrumVoTable(200000,false);
System.out.print(v.getDESCRIPTION()+": ");
System.out.println(v.getRESOURCE()[0].getTABLE()[0].getDATA().getTA
BLEDATA().getTR()[0].getTD()[1].getValue());

// System.out returns spectra in a virtual observational field
ArrayOfSpectrum aos = p.findSpectraCone(180,0,5,false);
Spectrum[] sa = aos.getSpectrum();
System.out.println("# of objects found: "+sa.length);
Spectrum c = null;
for( int si =0; si < sa.length; si++ ){
   c=sa[si];
   System.out.println("\t"+c.getRa()+"\t"+c.getDec());
}

// High redshift search yields QSOs
Spectrum[] qa = p.findSpectraRedshift(5,10,false).getSpectrum();
System.out.println("Z>5 (qso)");
Spectrum q = null;
for( int si =0; si < qa.length; si++ ) {
   q=sa[si];
   System.out.println("\t"+q.getName()+"\t"+q.getZ());
}

// Derive QSO composite by calling another web method
ArrayOfFloat limStart = new ArrayOfFloat();
limStart.set_float(new float[]{1430});
ArrayOfFloat limEnd = new ArrayOfFloat();
limStart.set_float(new float[]{1480});
s = p.composeAverageSpectrum(aos,
SpectrumWavelengthScale.Linear,1000,
         limStart,limEnd,1.4f,1e-4f);
System.out.println(s.getName());
for (int i=0; i<s.getPoints().getPoint().length; i++)
   if (s.getPoints().getPoint()[i].getWavelength()>1200 &&
s.getPoints().getPoint()[i].getWavelength()<1230)
      System.out.println
```

```
            ("\t"+s.getPoints().getPoint()[i].getWavelength()+"
  "+s.getPoints().getPoint()[i].getValue());
```

3 Concluding remarks

XML Web Services are really easy to use and hide all the complexity of the remote procedure invocation. They can be used everywhere because of their inherent interoperability and method calls go through even firewalls just as any web request. Web Services would certainly make building the Virtual Observatory easier and has already made possible developing prototypes such as SkyQuery [7].

The spectrum services presented here are the first attempt to publish large amount of spectra in the Virtual Observatory via web services. Its success depends on you, the user. We encourage you to register and publish your spectra (and filter profiles) in this VO facility that gives you full access to data you own. If you have comments or suggestions concerning the web site and services, please do not hesitate to send an email to us!

References and Links

Szalay, A. S., Budavári, T., Malik, T., Gray, J., and Thakar, A. R., 2002, '*Web Services for the Virtual Observatory',* SPIE, Astronomy Telescopes and Instruments [here]

[1] http://www.w3.org/
[2] http://www.ivoa.net/
[3] http://www.us-vo.org
[4] http://msdn.microsoft.com/netframework/
[5] http://ws.apache.org/axis/
[6] http://skyservice.pha.jhu.edu/devel/wave/
[7] http://www.skyquery.net