BiBiWS
WebService Workshop

Jan Krüger jkrueger@techfak.uni-bielefeld.de
Henning Mersch hmersch@techfak.uni-bielefeld.de

8. Oktober 2004

1 BiBiWS Development Guide

This *Development Guide* will show developers the steps to develop a WebService on BiBiServ using the BiBiWS framework. First, basic ideas and functions of a BiBiWS WebService are presented briefly, which are neccessary for understanding the concept. Afterwards to the development environment will be introduced. For getting into the idea of BiBiWS WebServices, a very brief introduction will clarify the workflow of development and how to get it working. They are followed by a detailed description of the source code of the default WebService for doing complex things.

Just for clarification:
A *developer* in this document is a person, who develops a bioinformatic tool and would like to provide a world-wide accessible WebService for this tool. A *user* is a person, who uses such a WebService.

1.1 Basics of a WebService

A WebService contains two sides: Client and server. In general a WebService is a synchronous data exchange service of two points connected via a network. This means after a short time (less than the http timeout, which normally is 300 seconds) an answer of the WebService server is required.

Bioinformatic programs normally run much longer than five minutes. To avoid problems with timeout and changes of IP-Address client, BiBiWS uses a technic called *Request and reply with polling*: First the client side submits a job with the required data (progra parameters and data etc.) and immediately gets an `id` after the job was started which normally takes some seconds (name convention `request()`).
Afterwards the client side is able to get the result using this `id`. If the computation of the program is not finished, the client side gets back a code with enhanced information. Otherwise the result of the call is returned (name convention `response()`).

This polling technique completely avoids problems with timeouts. The user can even request the results hours or days later - or from another host, just with knowledge of the *id*. Although there are asynchronous WebService projects, they have hard restrictions.

```
server/
  <PN>.wsdd          for generating the WSDL of the WebService and for deploying on the tomcat
  WEB-INF/           for including to WAR File
  build.xml          ant build file for building the web service
  config/            properties files
  dist/              distribution WAR and WSDL files
  doc/               documentation of project server side
  doc/api            automatical generated Java API
  lib/               additional libraries
  src/               source code
```

Figure 1.1: Overview of `server/` directory

1.2 Description of the project directory

Here are described the two basic subdirectories of the created default project.
Also the WebService global installation is described to get the default WebService to run. Understanding this is important to get the basic ideas at the following brief introduction.

1.2.1 Project directory

The BiBiServ Administrator Team will set up a project on `/vol/bibidev/<PN>/`, which already includes a running example of a BiBiWS WebService.
It produces a simple string of given length, so a pretty simple example, which the developer can modify to his or her own requirements.
All files and information have to be located in this directory and will be installed from this directory.

There are two subdirectories:

`server/` - WebService server side implementation

Subdirectory `server/` contains all required files for compiling and installing the server side of a WebService. Figure 1.1 gives an overview of the `server/` directory. We will discuss the individual elements later in Chapter 1.4.1

`client/` - command line WebService client side

Subdirectory `client/` represents a simple command line client written in Java for the WebService server side. Figure 1.2 shows the contents of this directory. Discussion in detail will follow in Chapter 1.4.3

```
client/
  Request.java       source of Request part of client
  Response.java      source of Response part of client
  example.fas        sample sequence data in Fasta for testing purpose
  setclasspath.sh    shell script for setting the correct CLASSPATH in current shell
```

Figure 1.2: Overview of `client/` directory

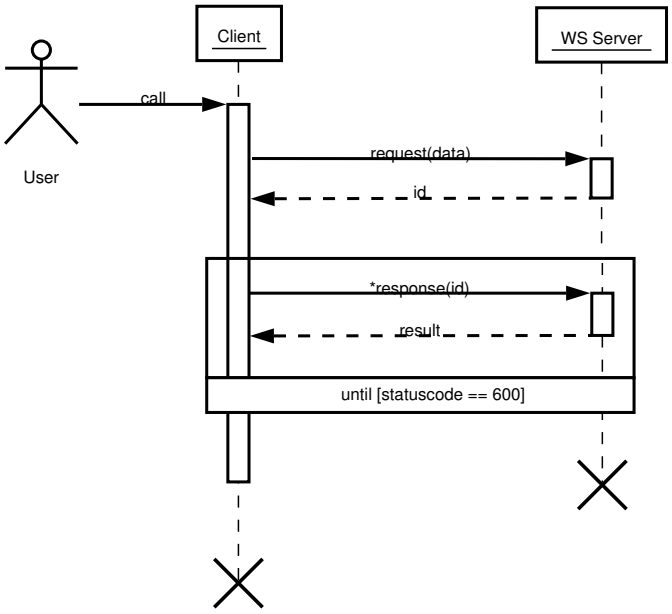

Figure 1.3: Basic flow of an WebService call of BiBiWS.

1.3 Brief introduction to the BiBiWS WebServices

This is a very basic and rudimentary description of how a BiBiWS WebService works. It is neither complete nor describes the whole source code, which is described later (Chapter 1.4). It is just a short description of the most important ideas of a BiBiWS WebService.
The example WebService gets an `Integer` „length“ as input and creates a `String` of `x`'s of the given length, which is returned to the user.

Figure 1.3 shows the basic flow of a call at the created WebService server side. After „starting“ the program by calling `request()` with the data to progress and getting an `id`, the client can *poll* for the result by calling the `response()` method with this `id`. Statuscodes (see Appendix 2) with reason phrases are used to represent the current status.[1]

[1] statuscode 600 indicates *successfully finished*

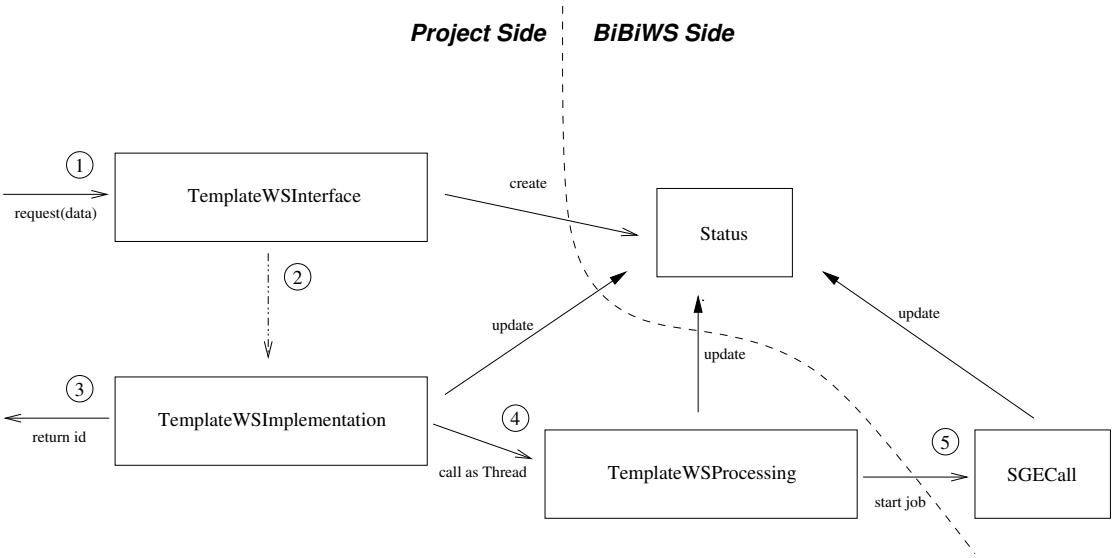

Figure 1.4: Basic flow of `request` call on server side.

1.3.1 Installing and running the default WebService of a new project

Very litte work has to be done for getting the default BiBiWS WebService of a new created project to run:

Step 1: Creating a new project `<LOGIN>WS` in your homedirectory is done by calling `/vol/bibiadm/bin/makeProjectHOBIT`

Step 2: Go to the project server directory in `~/<LOGIN>WS/server` and type `ant deploy_ws`. The Server side will be installed. Your default WebService is ready !

Step 3a: Go to the project client directory in `~/<LOGIN>WS/client`, set the `CLASSPATH` by typing `. ./setclasspath.sh`.

Step 3b: Compile the Java sources calling `javac *.java`

Step 3c: Type `java Request <Integer>` for calling the sample webservice and get an `id`.

Step 3d: Type `java Response <id of request>` and get the string of requested length (or get a status, so you have to try again later).

Step 4: As you see - everything works fine. You are welcome to start modifying this default project to support your own bioinformatic tool

1.3.2 Server Side

Figure 1.4 shows the basic flow of objects, if a `request()` call comes in.
The „project side“ represents classes, which have to be modified by the developer

where the BiBiWS side is the connection to the BiBiWS library.
The Status object is representing the current status while processing, so every class can modify this.

The source code is located at the subdirectory `server/src/`.

Here is briefly described, what has to be changed for getting your own tool running.

① The Interface class defines the types of incoming and outgoing data. Change the parameters of the `request()` method to your requirements.

② The Implementation class of the Interface. Again the parameters of `request()` have to be changed. Also parameters have to be checked.

③ `request()` returns the `id` of the WebService call.

④ Before returning the `id` the processing class is started as thread. Change the constructor for matching your parameters. Within the `run()` method of the processing class prepare the call of your tool by writing data to spooldirectory and convert parameters to command line parameters.

⑤ Also at the Processing class is the useage of `SGECall`, which executes your tool. This has to be modified to your personal tool.

The only modification at the `response()` method within Interface and Implementation classes are: Change the returned data type and read the result from spooldirectory for returning.

1.3.3 Client Side

Modify arguments used for your own WebService in `client/Request.java`. Also WebService parameters and processing data has to be changed.

1.3.4 Install your WebService

After modifying the default WebService application, it has to be installed.

step 1: Server side: call `ant` within the `server/` directory. This will compile and build WebService and build the WSDL Specification.

step 2: Client side: recompile the Java sources located at the `client` directory.

1.3.5 Testing your WebService

Change to the project client directory located in `~/<LOGIN>WS/client`. Now call your request program with your arguments to get an ID. Call the `java Response <id get by request>`.

1.4 Detailed introduction to the BiBiWS WebServices

This section explains all files and directories of the default project. It gives a developer a detailed overview of the source code of the default project.
After the brief introduction at Chapter 1.3, this gives the possibility to do more complex processing for development of a BiBiWS webservice.
The location of server side BiBiWS API is [BIBIWSAPI-WSS].

1.4.1 Server side of a BiBiWS tool

The `server/` directory contains several files and subdirectories, which are explained here. (see Figure 1.1 for overview)

`<PN>.wsdd`

The *Web Service Deployment descriptor* (short WSDD) is required to generate the WebService WAR File. Basically it just describes the class Axis has to bind to and methods to export.[2] Normally developers don't have to change anything here, except one would like to add methods or change names of methods.

`WEB-INF/`

This directory will be part of the WAR file to release. If specific jars or other files are neccessary for the tool - place them here in the `lib/` subdirectory.
Two libraries are already included in every WebService, because of tomcat problems otherwise.
Normally developer don't have to change anything here.

`build.xml`

On the server side development `ant` is used for processing. There are some `ant` targets for developers to use, described in Table 1.1.

[2] replace methods to * for exporting all methodes

redeploy_ws	(default) Will uninstall and install WebService
deploy_ws	Will install and start WebService (use this at 1st time installation)
undeploy_wst	Will remove and uninstall WebService
reload_ws	Will reload WebService on BiBiTest WebService Server
clean_dist	Will tidy up the project directory
api	Will generate the Java-doc api to `doc/api/`

Table 1.1: BiBiServ Developer commands of `server/` subdirectory

```
toolname=TEMPLATEWS
max.submit=100
statuscode.701=Input Error - Tool spec explanation
statuscode.701.internal=Input Error - Tool spec internal information
```

Figure 1.5: default `tool.properties` file

`config/`

`config/` includes two properties files.
First there is `tool.properties`. (see Figure 1.5) there are some properties configured on creation of the default project.

`toolname` is your <PN>

`max.submit` is maximum submission characters over all parameters accepted by WebService

`statuscode.701` is an example of how to set a client side statuscode. This will override the description of the default BiBiStatuscodes if defined there. (see Table 2.2 for complete listing of predefined BiBiStatuscodes)

`statuscode.701.internal` is the corresponding internal description, which will go to the log files additionally, but never occur outside.

Second there is the `log4j.properties`, which configures logging to `/vol/log/<PN>/wss.log`. See log4j manual for details [LOG4J]

`src/`

All classes belonging to a WebService are included at the package `de/unibi/techfak/bibiserv/<PN>`.
Here are the basic constraints of the classes:
AXIS requires an interface to the implementation of the methods called by WebService. The implementation class creates an `id` and calls the processing class as a thread with checking for maximum submitting characters before. After starting the thread, it

```
1   abstract String request(Hashtable params) throws TEMPLATEWSException;
2   abstract String response(String id) throws TEMPLATEWSException;
```

Figure 1.6: client to call `request()` on WebService

```
1   public String request(Hashtable params) throws TEMPLATEWSException {
2     WSSTools wsstools = new WSSTools(this.getClass().getResourceAsStream("/tool.properties"));
3     Status status = new Status(wsstools);
4     TEMPLATEWSProcessing proc = new TEMPLATEWSProcessing(wsstools,status,
     (new Integer(params.get("length").toString())));
5     Thread t = new Thread(proc);
6     t.start();
7     return status.getId();
8   }
```

Figure 1.7: Lines from `<PN>Implementation.java`

returns the `id` to the client side. The processing class itself calls the tool after preprocessing and before postprocessing by developer's implementation.
Following is the detailed description of classes:

`<PN>Interface.java` The interface definition in Figure 1.6 shows, that every method to provide for a WebService call has to be defined here.
The `request` method will return an `id` and get data for starting the WebService call. This example does not get any data, but only a length as parameter.
The `response` method gets an `id` and return results.

`<PN>Implementation.java` Figure 1.7 shows the `request()` method for submitting a job to WebService. After creating a new `status` object (line 3) the processing class is called as a thread. (line 4-6) `request` returns an `id` (line 7) for calling `response()` after a while by the client side.
The `response` method just requests the current status of the submitted id (line 3 at Figure 1.8). If statuscode is 600 the job has finished and the result can be read (line 5).

```
1  public String response(String id) throws TEMPLATEWSException {
2    WSSTools wsstools = new WSSTools(this.getClass().getResourceAsStream("/tool.properties"));
3    Status status = new Status(wsstools,id);
4    if(status.getStatuscode() == 600) { // ready
5
6      //read data and warp output to HOBIT Standards (see "Converting...")
7
8    } else {
9      throw new TEMPLATEWSException(status);
10   }
11 }
```

Figure 1.8: Lines from `<PN>Implementation.java`

```
1  public void run() {
2    status.setStatuscode(602);
3    status.setStatuscode(603);
4    SGECall call = new SGECall(wsstools, status);
5  if(!call.call("java -cp /vol/bibiwssv/webapps/templatehobit/WEB-INF/classes
       de.unibi.techfak.bibiserv.templatehobit.PerformanceTest "+length))
6      wsstools.log("error","unsuccessful SGECall...Processing aborted");
7      return;
8
9    status.setStatuscode(605);
10   wsstools.writeSpoolFile("stdout-log.txt",call.getStdOutStream());
11   wsstools.writeSpoolFile("stderr-log.txt",call.getStdErrStream());
12   status.setStatuscode(600);
13 }
```

Figure 1.9: Lines from <PN>Processing.java

```
1   public TEMPLATEWSException(Status status)
2   public TEMPLATEWSException(int statuscode, String description)
```

Figure 1.10: client to call request on WebService

Afterwards the result has to be converted for returning (line 6/7, see Chapter 1.4.2) .

If statuscode is different, response just informs the client side by returning an Axis fault (throwing a <PN>Exception, which is converted by the AXIS library).

<PN>Processing.java (Figure 1.9) After setting to statuscode 602 the preprocessing can be done, if required (line 2). Statuscode 603 means main processing, so the command execution is done (line 5), which returns after the job has finished or failed. Following line 9 there is the phase of postprocessing. In this example the STDOUT and STDERR streams of the execution call are written to the spooldirectory, because the example application prints the result to STDOUT. After all this is done, the statuscode is set to „finished“ - 600 (line 12), which indicates a possible return of the result, if the client asks for it.

<PN>Exception.java Due to the fact that AXIS does not accept a general BiBiException class, which is mapped to a soap_fault on error, every WebService has to have its own exception to throw.
In Figure 1.10 there are two constructors to create this exception. Either a status object can be given (line 1, will take statuscode and description from status object) or soap_fault and soap_description is set manually (line 2). So, normally, the developer has nothing to change here.

PerformanceTest.java This class is just a standalone script, which is called by the default WebService as an example. So it is not documented here and the developer should replace the execution of this class to their own tool's execution.

```
1       String xmlString = incoming_data;
2       DocumentBuilder docBuilder = factory.newDocumentBuilder();
3       Document doc = docBuilder.parse(new InputSource(new StringReader(xmlString)));
4       String stringOfLength = doc.getElementsByTagName("stringOfLength").item(1).getNodeValue();
```

Figure 1.11: create a DOM Object and access data

1.4.2 Converting result from/to specified XMLSchemas

The HOBIT-Projekt uses different existing XMLSchemas for representing data. This XMLSchemas describe formats to represent biological data in XML. If a program accepts or returns data in one of these standards, another tool can handle this data without converting. This way tools can be combined as chains.
So a BiBiWS WebService supporting HOBIT standards needs to accept and to return data within one of these XMLSchemas. It's up to the developer's decision which are best matching the tool's in- and output.

Here is a description how to convert data.

Converting data according to a XMLSchema to tool's input format.

While the example WebService just takes a parameter, but no further input data, it is not included at the example.
Figure 1.11 shows how to retrieving incoming XML. One just has to create a DOM Object from the incomming XML String and access the included data.

See [MCL] for further informations on the DOM library.

Converting tool's result format according to a XMLSchema

Figure 1.4.2 shows a basic conversion of result data to XML according to a XMLSchema.
At this WebService, we have an example XMLSchema
`http://bibiserv.techfak.uni-bielefeld.de/xsd/TemplateExample.xsd` to follow.
Line 1 and 2 gets the data from the spooldirectory.
Developer have to create a DOM object, including namespaces (lines 4-9), which is serializeable simple to return as string.
Lines 10 to 12 appends the returned string and lines 13 to 15 the log message.
Afterwards the serialization is done, which does not have to be modified. The generated string is returned to the client of the WebService call.

See [MCL] for further informations on the DOM library.

```
1     String stringOflength = new String(wsstools.readSpoolFile("stdout-log.txt"));
2     String logmsg = new String(wsstools.readSpoolFile("stderr-log.txt"));
3     //Creating DOM
4     DOMImplementationImpl impl = new DOMImplementationImpl();
5     Document domDocument =
        impl.createDocument("de:unibi:techfak:bibiserv:templateexample", "dataTypeTest", null);
6     Element dataTypeTestElement = domDocument.getDocumentElement();
7     dataTypeTestElement.setAttribute("xmlns", "de:unibi:techfak:bibiserv:templateexample");
8     dataTypeTestElement.setAttribute("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
9     dataTypeTestElement.setAttribute("xsi:schemaLocation",
    "de:unibi:techfak:bibiserv:templateexample
        http://bibiserv.techfak.uni-bielefeld.de/xsd/TemplateExample.xsd");
10    Element stringOfLengthElement = domDocument.createElement("stringOfLength");
11    dataTypeTestElement.appendChild(stringOfLengthElement);
12    stringOfLengthElement.appendChild(domDocument.createTextNode(stringOflength));
13    Element logmsgElement = domDocument.createElement("logmsg");
14    dataTypeTestElement.appendChild(logmsgElement);
15    logmsgElement.appendChild(domDocument.createTextNode(logmsg));
```

Figure 1.12: create a DOM Object and fill with result data

1.4.3 Command line client for testing server side

Source of the client side is mostly simple and (hopefully) self explaining.

2 BiBiWS - Predefined statuscodes

Like the (for now) underlaying http protocol, BiBiWS gives back statuscodes with reason phrases to inform the user if the requested data isn't returned.
For not getting confused with http statuscodes, BiBiWS uses statuscodes beginning with `6` and `7`, while http uses statuscodes beginning with `1` to `4` (see RFC 2616 [RFC2616] for details).

This chapter gives an overview of the predefined statuscodes by BiBiWS.
They can be overridden by the tool specific statuscodes, simply by giving same numbers at the tool specific properties file.
There are some general conventions explained in Table 2.1 to follow, which are required, because BiBiWS decides on these rules whether an error occurred, the WebService finished or user has to wait.
Some statuscodes can be used on several reasons and it might be hard for the developer to decide, where it comes from. So all statuscodes may have a internal description (`.internal` at the `properties` file) , which is never going outside the server. It is just for debugging and development of the tools on server side. It can be set individual to enhance the normal description, which is presented to the user.

Predefined statuscodes occurring on server side are explained in Table 2.2 and the client side ones are described in Table 2.3. They begin in general with a `c`, so the statuscodes theirself can be as equal as possible on both sides.

Statuscode	Description
600	WebService call is finished successfully, result ready.
6xx (beginning with 6)	WebService call is not yet finished. `xx` gives more information.
700	WebService call is in general error state. This fall-back, Should be avoided by developer to give more information.
70x (beginning with 70)	User errors. Submitted data is NOT processed by the WebService call.
72x (beginning with 72)	Errors relaying to BiBiServ Administrator Team or developer of the tool. User can inform and/or retry later.

Table 2.1: General BiBiWS conventions for statuscodes

Statuscode	Description
600	Finished OK
601	Submitted
602	PreProcessing
603	Processing: Pending
604	Processing: Running
605	Postprocessing
700	General Error
701	Input Format Error
702	Input Size Error (submitted data to large)
703	Exec Error (executable gives enhanced errormsg)
704	RAM Size Error
705	CPU Time Error
706	ID unknown (or older than 30 days)
707	ID data deleted (older than 3 days)
708	Mail Check Failed (notification email is not valid)
720	WSS Server Busy
721	Internal Resource Error *internal description:* Internal Resource Error (Grid)
722	Internal Resource Error *internal description:* Internal Resource Error (DB)
723	Internal Resource Error *internal description:* Internal Resource Error (HDDfull)
724	Internal Resource Error *internal description:* Internal Resource Error (WS-Error)
725	Internal Resource Error *internal description:* Internal Resource Error (BiBiWSS-Lib Error)
731	Resource Busy *internal description:* Resource Busy (Grid)
732	Resource Busy *internal description:* Resource Busy (DB)
733	Resource Busy *internal description:* Resource Busy (HDDfull)

Table 2.2: Predefined BiBiWS statuscodes on server side

Statuscode	Description
c600	Ready - got result
c601	Submitted
c700	General unknown error
c701	Input Error
c702	Input size too large
c708	Mail Check Failed (notification email is not valid)
c723	Internal Error - BiBiServ Team is informed, please try again later. *internal description:* Internal Resource Error (HDDfull)
c724	Internal Error - BiBiServ Team is informed, please try again later. *internal description:* Internal Resource Error (WS-Error)

Table 2.3: Predefined BiBiWS statuscodes on client side

Literaturverzeichnis

[ROSE] Rose, Marshall T.: *Beep - The Definition Guide* O'Reilly, 2002

[OEST] Oestereich, B.: *Objektorientierte Softwareentwicklung: Analyse und Design mit der UML* Oldenbourg, 2004

[CHAP] Chappell D. and Jewell, T.: *Java Web Services* O'Reilly, 2002

[RAY] Ray, R. J. and Kulchenko, P.: *Programming Web Services with Perl* O'Reilly, 2002

[TANE] Tanenbaum, A. S.: *Computer Networks* Prentice Hall, 1996

[HERM] Hermjakob, Henning et. al.: *The HUPO PSI Molecular Interaction Format - A community standard for the representation of protein interaction data* Nature Biotechnology, 2004

[MCL] McLaughlim, B.: *Java and XML* O'Reilly, 2001

[CORBA] Sun Microsystems: *Introduction to CORBA*
31.08.2004 <http://java.sun.com/developer/onlineTraining/corba/corba.html >

[XMLRPC] *XML-RPC Home Page*
31.08.2004 <http://www.xmlrpc.com/ >

[JRMI] David Reilly: *Introduction to Java RMI*
31.08.2004 <http://www.javacoffeebreak.com/articles/javarmi/javarmi.html >

[WSDL] W3C: *Web Services Description Language (WSDL) 1.1*
31.08.2004 <http://www.w3.org/TR/wsdl >

[XML] W3C: *Extensible Markup Language (XML)*
31.08.2004 <http://www.w3c.org/XML/ >

[XMLSCHEMA] W3C: *XML Schema*
31.08.2004 <http://www.w3.org/XML/Schema >

[HOBIT] *HOBIT-Projekt*
31.08.2004 <http://mips.gsf.de/proj/hobit/ >

[PSI] HUPO: *Proteomics Standards Initiative*
31.08.2004 <http://psidev.sf.net/ >

[PERL] O'Reilly: *Perl - The Source for Perl*
31.08.2004 <http://www.perl.com/ >

[JAVA] Sun Microsystems: *Java*
31.08.2004 <http://www.java.com/ >

[APACHE] *The Apache Software Foundation*
31.08.2004 <http://www.apache.org >

[APACHE] The Apache Software Foundation: *Apache http Server*
31.08.2004 <http://httpd.apache.org/http/ >

[TOMCAT] The Apache Software Foundation: *Apache Jakarta Tomcat*
31.08.2004 <http://jakarta.apache.org/tomcat >

[AXIS] The Apache Software Foundation: *AXIS*
31.08.2004 <http://ws.apache.org/axis/

[AXISUM] The Apache Software Foundation: *AXIS User's Manual* 31.08.2004 <http://ws.apache.org/axis/java/user-guide.html

[CEBITEC] *Center for Biotechnology (CeBiTec)*
31.08.2004 <http://www.cebitec.uni-bielefeld.de/ >

[DFG] *Deutsche Forschungsgemeinschaft*
31.08.2004 <http://www.dfg.de >

[HELMHOLTZ] *Helmholtz Association of National Research Centres*
31.08.2004 <http://www.helmholtz.de/ >

[DRMAA] *Distributed Resource Management Application API Working Group*
31.08.2004 <http://www.drmaa.org/ >

[UDDI] *OASIS UDDI*
31.08.2004 <http://www.uddi.org/ >

[RFC2616] *RFC 2616: Hypertext Transfer Protocol - HTTP/1.1*
31.08.2004 <http://www.w3.org/Protocols/rfc2616/rfc2616.html >

[WSDEF] *Web Services Activity*
31.08.2004 <http://www.w3.org/2002/ws/ >

[JCP] *Java Comunity Process - Homepage* 31.08.2004 <http://jcp.org/en/home/index >

[SOAP] W3C: *SOAP Version 1.2* 31.08.2004 <http://www.w3.org/TR/soap12-part1/ >

[HTTP] W3C: *Hypertext Transfer Protocol – HTTP/1.1* 31.08.2004 <http://www.w3.org/Protocols/rfc2616/rfc2616.html >

[DOM] W3C: *Document Object Model (DOM)* 31.08.2004 <http://www.w3.org/DOM/ >

[ADA1] Adams, H.: *Asynchronous operations and WebService, Part 1: A primer on asynchrnous trasnactions.* 31.08.2004 <http://www-106.ibm.com/developerworks/library/wsasynch1/ >

[ADA2] Adams, H.: *Asynchronous operations and WebService, Part 2: Programming patterns to build asynchronous WebService.* 31.08.2004 <http://www-106.ibm.com/developerworks/library/wsasynch2/ >

[SGEUM] Sun Mircosystems: *Grid Engine* 31.08.2004 <http://gridengine.sunsource.net/ >

[SGEUM] Sun Mircosystems: *Sun grid engine user manual* 31.08.2004 <http://gridengine.sunsource.net/project/gridengine-download/SGE53AdminUserDoc.pdf?content-type=application/pdf>

[XHTML] W3C: *XHTML 1.0 - The Extensible HyperText Markup Language* 31.08.2004 <http://www.w3.org/TR/xhtml1/ >

[LOG4P] *The log4perl project* 31.08.2004 <http://log4perl.sourceforge.net/ >

[LOG4J] Apache Software Foundation: *Logging Services* 31.08.2004 <http://logging.apache.org/log4j/ >

[CPAN] *CPAN - Comprehensive Perl Archive Network* 31.08.2004 <www.cpan.org >

[RPCVSDOC] *RPC vs. Document WSDL encoding* 31.08.2004 <http://www.rassoc.com/gregr/weblog/archive.aspx?post=465 >

[APALIZ] *The Apache Software Foundation - Licenses* 31.08.2004 <http://www.apache.org/licenses/ >

[NETCRAFT] Netcraft: *Web Server Survey* 31.08.2004 <http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html >

[JSP] *JavaServer Pages Technology* 31.08.2004 <http://java.sun.com/products/jsp/ >

[BIBIWSDLS] *WSDL Spezifikationen der auf BiBiServ angeboteten WebServices* 31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/wsdl/>

[BIBISERVSTATS] *Statistiken des BiBiServ* 31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/statistik/>

[BIBISERVPOLICIES] *Policies of BiBiServ*
31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/bibi/Administration_Policies.html >

[BIBISERV] *Technical Faculty of the University Bielefeld*: *The Bielefeld University Bioinformatics Server (BiBiServ)*
31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/ >

[BIBISERV] *Der BiBiServ (production system)*
31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/ >

[BIBITEST] *Der BiBiTest (development system)*
31.08.2004 <http://bibitest.techfak.uni-bielefeld.de/ >

[BIBIWSAPI-WSS] *BiBiWS server side API*
31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/hobit/wss-api/ >

[BIBIWSAPI-WSC] *BiBiWS client side API*
31.08.2004 <http://bibiserv.techfak.uni-bielefeld.de/hobit/wsc-api.html >