A flexible statistics web processing service - Added value for information systems for experiment data*

Dennis Heimann[1], Jens Nieschulze[1], Birgitta König-Ries[2]

[1]Max-Planck-Institute for Biogeochemistry, Hans-Knoell-Straße 10, D-07745 Jena, Germany
{dheimann, jniesch}@bgc-jena.mpg.de

[2]Heinz-Nixdorf Endowed Chair of Practical Computer Science, Friedrich Schiller University, Ernst-Abbe-Platz 1-4, D-07743 Jena, Germany
birgitta.koenig-ries@uni-jena.de

Summary

Data management in the life sciences has evolved from simple storage of data to complex information systems providing additional functionalities like analysis and visualization capabilities, demanding the integration of statistical tools.
In many cases the used statistical tools are hard-coded within the system. That leads to an expensive integration, substitution, or extension of tools because all changes have to be done in program code.
Other systems are using generic solutions for tool integration but adapting them to another system is mostly rather extensive work.
This paper shows a way to provide statistical functionality over a statistics web service, which can be easily integrated in any information system and set up using XML configuration files. The statistical functionality is extendable by simply adding the description of a new application to a configuration file. The service architecture as well as the data exchange process between client and service and the adding of analysis applications to the underlying service provider are described. Furthermore a practical example demonstrates the functionality of the service.

1 Introduction

Data management in the life sciences has been a very active area of research for a number of years ([2], [3], [4], [5]). Over time, it has become obvious, that it does not suffice for tools to offer ”just” integrated (centralized or virtual) data storage capabilities. Rather, users demand direct access to a diverse set of tools to access and analyze data. Of particular importance to many users are statistical analysis tools. What needed are therefore platforms that combine integrated data storage with seamless access to data analysis tools.

The Biodiversity Exploratories[1], a large-scale and long-term biodiversity research project in Germany, are an adequate example in need of such a system. The project aims to examine the relationship between land-use intensity, biodiversity change, and ecosystem functioning for selected taxa. Within this umbrella project, a large number (currently 32, more will be added over time) of individual, independent projects from a diverse set of communities investigate different

*This article is an extended version of [1]
[1]http://www.biodiversity-exploratories.de

aspects of the overall problem, comprising research on botany, vertebrates, invertebrates, soil sciences, and biogeochemical processes. One of the expectations towards the umbrella project is to make data available beyond individual projects to allow for analysis of data across disciplines and over time, e.g., to be able to explore changes in biodiversity over a decade and relate this to changes in the soil brought on by the use of certain fertilizers.

As a technical basis for this task, we are developing the web-based Biodiversity Exploratories Information System[2](BExIS) which offers central storage and management of all project data. To overcome existing semantic heterogeneity among different subprojects' data, a knowledge-based framework for the data management is currently being developed. Along with the upload and download of project data, BExIS will then be able to provide facilities to merge data sets from different subprojects. Relating data sets then allows analyses of data across different subprojects to find relationships between the diverse research areas.

One of the main non-technical challenges faced by BExIS is to get the individual researchers involved actually to use the system. While data uploading can be enforced to a certain degree by central policies, the uploaded data will only be complete and described in sufficient detail, if the researcher sees a direct benefit from using BExIS[3]. We believe that seamless access to analysis tools and the ability to plug in new tools as needed is one way to provide such a direct benefit. In the long run, the seamless integration of statistical methods will also enable to do common analyses across projects directly within the system. The statistical methods needed range from simple calculations and summaries of data sets to complex analyses models. Statistical methods can also be used for visualization purposes embedded in applications without any statistical background. An example for the latter is the visualization of the segmentation of an experimental plot with all its subplots.

In the remainder of the paper, we are going to discuss our approach of seamless integration of such tools into data management platform. We will use the integration of the R statistics package into BExIS as our running example.

The remainder of this paper is organized as follows: Section 2 discusses related approaches and their drawbacks. After that, Section 3 introduces the design of the statistical web service we have developed and its architecture. Section 4 explains the data exchange process between client and service, Section 5 illustrates the query process starting at the client request and ending at the web service response. Section 6 demonstrates the functionality of the introduced web service with the exemplary integration of the statistics package R. Section 7 shows some performance issues of the web service in comparison to direct method calls within R, before section 8 summarizes this web service approach for tool integration.

2 Related work

Over the last few years, a number of attempts to solve at least similar requirements have been proposed. These use different approaches to integrate their tools. One approach is to declare the input and output directory of a tool, thus enabling the host application to access the raw

[2]http://www.exploratories.bgc-jena.mpg.de

[3]The ability to run cross-project analysis is no such immediate benefit, in particular not for the current set of researchers who are the first to use BExIS. We believe that it will be such a benefit for the next generation of users who will then be able to access data provided by their predecessors.

files. An example for such a system is B-Fabric, a system integrating biological analysis tools. B-Fabric's main focus is on the integrated management of all data generated from different analysis tools. Direct tool access is beyond the current scope of B-Fabric. Accordingly, it provides no such facilities [5].

Another approach is to hard-code the access to a tool within the host application code. An example project relying on this approach is the MIGenAS integrated bioinformatics toolkit that allows the analysis of biological sequences over the internet [6]. Here, the access to all supported bioinformatics tools is hard-coded within Java [7]. This leads to an expensive integration, substitution, or extension of tools, because all changes have to be done in the program code of the host application.

Other solutions use a more generic approach to integrate external tools, for example SWAMI[4]. SWAMI tries to provide an integrated environment where biological tools, user data, and public data resources can be easily accessed [4]. The used architecture is highly scalable in terms of adding databases and new tools because of using configuration files for providing definitions for tools and data types, and physical descriptions of resource locations.

While this approach is very promising, the adaption of such a complex generic workflow system to the specific needs of a concrete application is very challenging and involves a lot of effort.

That is why we tried to use advantages of generic integration but in a less complex way for our own system. We have developed a web service for accessing diverse statistical analysis methods. A similar solution was explored inside the Bioconductor project[5] where an R package, namely RWebServices, was developed. The RWebServices approach is to convert R functions to Java objects and methods for exposure as Java-based web services [8]. Obviously, every implemented R function needs its own web service.

Our approach is to combine all statistical methods within one web service. It provides only three operations to list all available statistical methods, to describe a method more specifically, and to invoke a method. By abstraction from the underlying applications, the web service will be easy to integrate in basically any information system.

3 Web service design

3.1 Requirements

In order to achieve seamless integration of external tools, in our running example statistical analysis tools, into BExIS the whole system needs to meet a number of requirements, most importantly abstraction and scalability.

The latter one can be realized by making addition and substitution of methods possible with as little effort as possible. Abstracting from integrated methods is really important to allow a transparent view to all tools by the users. Today, the most common approach – and the one chosen by us, too – to fulfill these requirements is via the use of a service-oriented architecture (SOA). SOA allows for the seamless, platform independent integration of different tools. It

[4]http://www.ngbw.org/
[5]http://www.bioconductor.org

makes it possible to dynamically change the set of offered tools and enables integration of tools without the need of their modification. All that is needed is the realization and description of an appropriate interface. Such an interface can be added to a tool without the need to alter anything in the tool itself - even without the ”knowledge” of the tool.

If one takes a closer look at tools like R, it becomes obvious that they offer a large number of different functions that a user may want to use. The naive approach would be to offer each of these functions as a separate web service (cf. [8]) or at least as a separate operation within a common web service. It should be obvious, that such an approach would result in a lot of description and implementation effort. Thus, a more lightweight is needed to make the development of a flexible solution feasible. In our search for such a lightweight approach we chose the Open GIS Web Processing Service (WPS). This service, which is described in more detail in the next subsection, offers the kind of solution we need - albeit for a different domain. We thus used it as inspiration for our solution and adapted it as needed as described in the remainder of this section.

3.2 Architecture

The design of the statistics web service follows the WPS specification by the Open Geospatial Consortium (OGC)[9]. WPS is designed to make arbitrary GIS functionality available over the internet. Examples of such functionalities are access to simple or complex calculations on spatial data or to computational models. To enable such flexibility, OGC specifies the interface of a general purpose web service that can be used to encode the offering of any desired functionality. Any calculation including its input and output are described as a web service. The specific calculations offered by a WPS implementation are defined by the owner of that implementation. WPS defines three operations each implementation needs to support:

- *GetCapabilities* returns service-level metadata, in particular a description of the capabilities of the offered service
- *DescribeProcess* returns a detailed description of a specific process offered by an implementation including its inputs and outputs,
- *Execute* triggers the execution of a process and returns the outputs of this process.

Quite obviously, the general concept behind WPS is applicable not only for geographic information systems, but in other domains, too. We have, therefore, adapted the WPS approach to our system. However, in order to keep our statistics web service as simple as possible, we did not want a full-fledged implementation of a WPS but adopted only those parts of its concept that we needed. This has led to the design shown in Figure 1.

There is a *StatisticsServer* providing the *StatisticsService*. Apache Axis2/Java [6] is used as web service engine and Apache Tomcat [7] serves as servlet container. The *StatisticsService* contains a central *Controller* handling all service requests and routing them over an *ApplicationConnector* to the corresponding underlying applications. The routing is based on two configuration files, *methods.xml* and *serverConfig.xml*. All available methods provided by the web service as well

[6] http://ws.apache.org/axis2/index.html
[7] http://tomcat.apache.org/

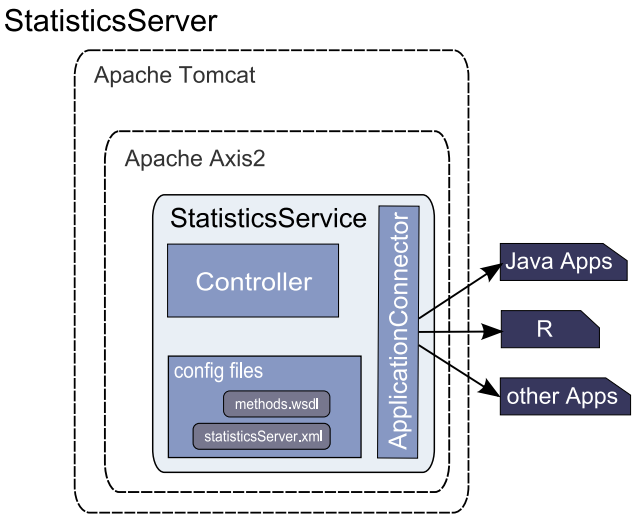

Figure 1: Statistical Web Service Architecture

as their access information are described within the *methods.xml* file. For description, the Web Service Description Language (WSDL) is used. The underlying applications are not necessarily web services, but WSDL affords all needed constructs to describe the provided methods [10], including their interface and data type descriptions.

As an example, Listing 1 shows an excerpt of a Java application computing the summary of a given data set.

Listing 1: Java application computing the summary of a data set.

```
package Methods;
public class Statistics {
      public String Summary(
            String inData, String maxsum, String digits)
      {...}
}
```

A mapping subset of the *Summary* method from Listing 1 to a WSDL description is demonstrated in Listing 2

Listing 2: Mapping of the *Summary* method from Listing 1 to a WSDL description.

```
<wsdl2:description ...
      xmlns:bgcs="Methods.Statistics"
      targetNamespace="Methods.Statistics">
<wsdl2:types> ...
<xs:element name="Summary">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="inFile" type="xs:string"/>
                  <xs:element name="maxsum" type="xs:string"/>
                  <xs:element name="digits" type="xs:string"/>
            </xs:sequence>
```

```
      </xs:complexType>
</xs:element>
<xs:element name="SummaryResponse">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="return" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
 ...
</wsdl2:types>
<wsdl2:interface name="ServiceInterface">
      <wsdl2:operation name="Summary"... >
            <wsdl2:input
                  element="bgcs:Summary".../>
            <wsdl2:output
                  element="bgcs:SummaryResponse".../>
      </wsdl2:operation>
</wsdl2:interface>
...
```

The package information important for method access by the controller is described by the *bgcs* namespace. Within the *ServiceInterface* definition the *Summary* operation together with its input and output is declared. The element types of all input and output parameters are specified within the types definition block.

In addition to *methods.xml* the *serverConfig.xml* file contains some general information about the server, for example the input and output paths of the server applications.

3.3 Class model

Figure 2 shows the class model of the statistics web service corresponding to its described architecture. The *StatisticsController* class is the central access point to the service. It uses

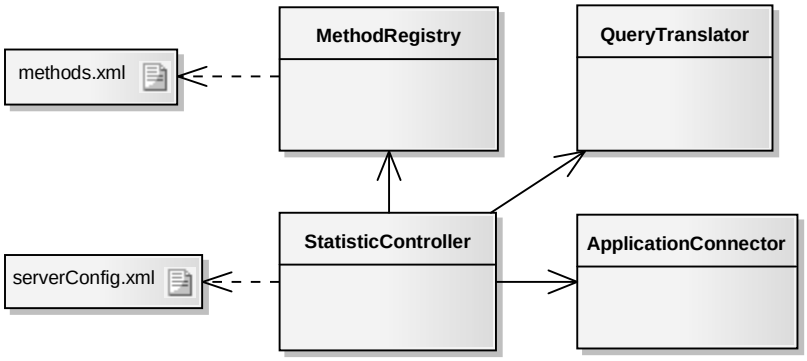

Figure 2: Class model of statistics web service

the *QueryTranslator* class to extract the query information from all incoming requests. The

MethodRegistry class contains functions for reading the *methods.xml* file. Server applications providing the statistical functionality are accessed over the *ApplicationConnector* class.

To abstract from the underlying applications the web service provides only the three operations analogous to the WPS:

- *getMethods* returns a list of all statistical methods provided by the service.
- *getMethod* returns a description of a specific method including its inputs and outputs
- *runMethod* runs a method and returns its outputs.

The transformation to the specific method calls is made by the *StatisticsController* and the *ApplicationConnector* using Java introspection together with the configuration files. Currently the applications are limited to Java but others can be easily integrated by wrapping them in a Java application. By abstracting from the underlying applications the statistics web service is easy to extend, for Java applications only the configuration files have to be changed.

4 Data exchange

An important issue concerns the data exchange between the statistics web service client and the statistics server. Typically data located at the client system have to be transferred to the server. The statistics web service provides different ways for this purpose. Data can be send within the query, as a URI, or as a JDBC database connection string. The type of data exchange is defined within the query.
The query is implemented as a simple message schema based on xml. It provides the three mentioned kinds of service operations.

1. *getMethods* to get a list of all provided methods:

```
<query>
      <type>getMethods<type>
</query>
```

2. *getMethod* to get a description of a specific method:

```
<query>
      <type>getMethod<type>
      <method>{Methods.Statistics}Summary<method>
</query>
```

   Here the fully-qualified method name has to be specified.

3. *runMethod* to execute a method

```
<query>
      <type>runMethod<type>
      <method>{Methods.Statistics}Summary</method>
```

```
            <parameters>
                  <parameter>
                        <name>inData</name>
                        <type>xsd:string</type>
                        <position>0<position>
                        <value>test.txt</value>
                  <parameter>
...
```

Within a *runMethod* query the method name as well as all parameters have to be specified, and at the end of a query message the access to the data has to be declared (Listing 3).

Listing 3: Data access description within query.

```
<dataAccess>
      <format>inQuery</format>
      <parameterPosition>0</parameterPosition>
      <columnDelimiter>;</columnDelimiter>
      <decimalSign>.</decimalSign>
      <data>a;b;c
      1;0.5;7.3
      4.1;2.1;4
      </data>
...
```

It includes the format, *inQuery*, *Uri*, or *JDBC*, the position of the data parameter, delimiter and decimal signs as well as the data if *type* is *inQuery*.
After sending the query to the server the *QueryTranslator* class extracts the data if transferred within the query, gets them from the location specified by a URI or loads them via a specified database connection. A unique temporary file will then be created in the input folder of the target application declared by the *serverConfig.xml* file. Hence, the temporary file stores all data and the file name is returned to the *StatisticsController*.

5 Query process

The query process is illustrated in Figure 3. A client sends a request to the web service using the SOAP message protocol [11]. Within the SOAP message body the query is defined as described in section 4. On the server-side the *StatisticsController* class receives the query and checks its type using the *QueryTranslator* class.
Depending on the query type either the *MethodRegistry* class or the *ApplicationConnector* class are used to answer the request or rather to invoke an application. The result of a query is send back to the client by the *StatisticsController* class also via a SOAP message.

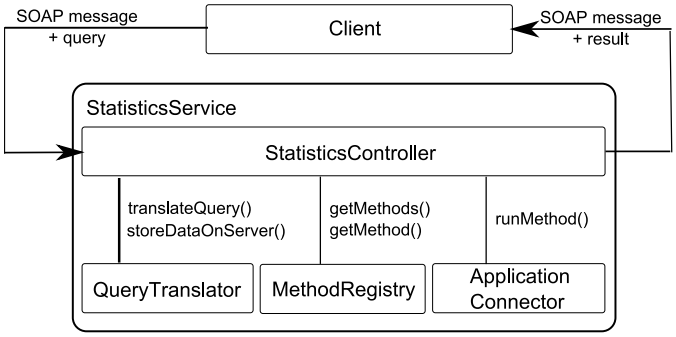

Figure 3: Query process of statistics web service

6 Example

As explained in Section 1 BExIS wants to provide statistical tools to the user. As an example, in this section, the integration of methods based upon the statistics package GNU R[8] is demonstrated. We have chosen R because it is widely used by our user community. Its integration thus offers a considerable benefit to our users.

6.1 Method integration

A Java application was developed which uses Rserve[9] and its Java client to implement the different methods. Rserve is a TCP/IP server which allows other programs to use facilities of R from various languages without the need to initialize R [12]. Within Java a connection to R can simple be made by

```
Rconnection c = new Rconnection();
```

if there is a running Rserve on the local machine. An R command can then be evaluated as a String expression; for example by

```
REXP x = c.eval("R.version.string");
System.out.println(x.asString());
```

The result is stored within an object of class REXP. The REXP class encapsulates any objects received from Rserve and provides accessor methods to obtain the Java object corresponding to the R value. Here the corresponding Java object is a String which is printed out.

The application that is integrated in the statistical web service is based on a simple design. Currently, there is only one class which implements all methods and connects to Rserve, but there is no restriction to modularize or extend it. As a simple example the implementation of the computation of a scatter plot is illustrated in Listing 4, but also calculations of complex models are possible.

Listing 4: Implementation of the ScatterPlot method in Java.

```
package Methods;
```

[8] http://www.r-project.org/
[9] http://www.rforge.net/Rserve/

```
import org.rosuda.REngine.*;
public class Statistics{
      RConnection rCon = new RConnection;
      public byte[] ScatterPlot(String inData,
                  String variable1, String variable2)
      {     String rScript =
                  "bw<-read.table(" + inData + ",header=T)"
                  "data<-bw[-1,]" +
                  "f<-tempfile()" +
                  "png(file=f)" +
                  "plot(" +
                        "data[,"+ variable1+ "]," +
                        "data[,"+ variable2 + "])" +
                  "dev.off()" +
                  "con=file(f,open=rb)" +
                  "g<-readBin(con,raw(),n=50000)" +
                  "g";
      REXP rexp = rCon.eval(rScript);
      return rexp.asBytes();
      }
}
```

The code shows that the ScatterPlot function takes three parameters, the file containing the data and two variables specifying the columns to correlate. Within the function a batch of R commands are concatenated to a String and subsequently evaluated. First, data are read and stored in an object. Then, a temporary file is created to store the graph. After this, the graph is plotted using the data described by the two variables. Finally, the binary graph data is read in again from the temporary file and returned to the REXP object.

Let us now take a detailed look at how this method can be integrated into the statistics web service. As it turns out, only two steps are required to achieve this goal. Firstly, a WSDL description of the ScatterPlot function has to be created and put into the *methods.xml* file. Secondly, the corresponding class files have to be put into the statistical web service classes folder. After restarting the Axis engine, the ScatterPlot method will be accessible.

6.2 Statistics web service used by BExIS

Now, we will describe how the statistics web service and its provided methods can be used. We will use an example from BExIS to illustrate this. The web service is accessible through BExIS by a separate statistics application.

To analyze a data set using a method from the statistics web service the procedure is as follows:

Firstly, the user has to select a data set stored in the system. An example of this step is shown in Table 1. It shows an observation of trees including the measurement of their diameter at breast height (dbh) and their age in years.

Secondly, the user has to choose a statistical method he/she wants to apply to the data set. The list of methods is obtained by the use of the *getMethods* operation of the statistics web service.

Table 1: Example data set: Measurement of dbh.

observationId	species	dbh	treeage
384301	beech	31.31	75
384302	beech	33.88	140
384303	beech	28.93	110
384304	beech	20.38	84
384305	beech	40.41	103
384306	beech	41.31	85
384307	beech	37.88	121
384308	beech	23.93	90
384309	beech	22.38	84
384310	beech	39.41	131

For the sample data set it could be interesting to find out, if there is a correlation between a tree's age and its dbh. To visualize such a possible correlation, the *ScatterPlot* method is applicable. After the user has chosen the method, he uses the *getMethod* operation from the web service to obtain detailed information about this function, in particular about the required inputs and expected outputs. The description of the ScatterPlot example is shown in Table 2.

Table 2: Description of the ScatterPlot method.

Name	Type	Description	Direction
inData	xs:string	...	IN
variable1	xs:string	...	IN
variable2	xs:string	...	IN
return	xs:base64Binary	...	OUT

Thirdly, the user has to assign a value to each IN parameter in the description over a web form. Depending on the parameter description, the value can be a text or a number, or a name of a column of the selected data set. For the sample data an assignment could look like illustrated in Table 3. The *inData* parameter is assigned an arbitrary text. This text specifies the name of the

Table 3: Example of a parameter assignment for a data set.

Name	Type	Assignment
inData	xs:string	data.txt
variable1	xs:string	column(treeage)
variable2	xs:string	column(dbh)

data file to be stored on the server by the *QueryTranslator* class. The assignments for *variable1* and *variable2* are column names of the sample data set, namely *treeage* and *dbh*. They specify the values of the columns of the data set to be used by the ScatterPlot method.

The data access specification has to be determined by the statistics web service client, that is BExIS. BExIS has to specify the format (inQuery, URI or JDBC), the position of the Parameter containing the value for the data description, the column delimiter, and the decimal sign. Additionally BExIS has to prepare the data depending on the transfer format. For the sample a data access specification could look like the one shown in Listing 5.

Listing 5: Query data access specification example.

```
<dataAccess>
      <format>inQuery</format>
      <parameterPosition>0</parameterPosition>
      <columnDelimiter>;</columnDelimiter>
      <decimalSign>.</decimalSign>
      <data>observationId;species;dbh;treeage
      384301;beech;31.31;75
      384302;beech;33.88;121
      384303;beech;28.93;110
      384304;beech;20.38;84
      384305;beech;40.41;143
      384306;beech;41.31;85
      384307;beech;37.88;121
      384308;beech;23.93;90
      384309;beech;22.38;84
      384310;beech;39.41;131
      </data>
</dataAccess>
```

Fourthly, the runMethod operation can be invoked and the result will be displayed on the page. Depending on the return type of a statistical method the displayed result can vary from text to image data. The ScatterPlot example returns an image as a byte array, so BExIS has to transform the byte array to an image and displays it on the page.

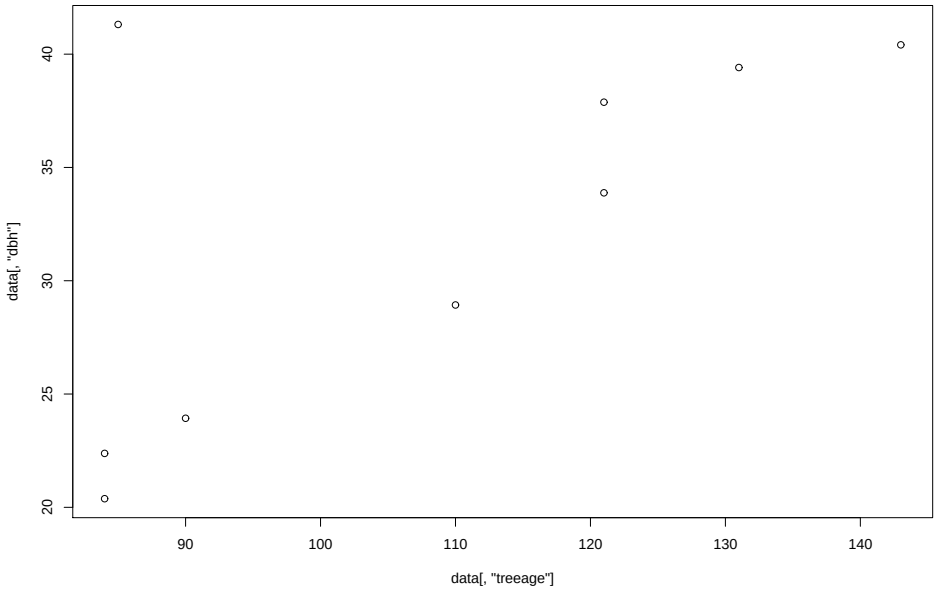

Figure 4: Result of ScatterPlot method.

Figure 4 shows the result for our example; evidently, the dbh of one tree correlates with its age; bar the outlier.

7 Performance

To check the performance of the StatisticsService, we created some test queries and measured the response time when calling the ScatterPlot method. Therefore two test data sets each with three columns (id of integer, x and y of real) were generated, one containing 10.000 rows and the other containing 50.000 rows. Then the two data sets were passed to the StatisticsService per *inQuery*, *Url*, and *JDBC* respectively. Additionally we measured the response time for accessing the ScatterPlot method directly within R over a script. Here data passed as a text file and subsequently as a JDBC connection.
The results can be seen in figure 4.
The JDBC access within R takes longer than the JDBC access over the Service. The reason is

Table 4: Measurement of response time of test queries.

transfer type	response in sec (10.000 rows)	response in sec (50.000 rows)
inQuery	0.51	1.78
JDBC	0.62	1.85
Url	0.53	1.91
direct pass to R (txt file)	0.2	0.95
direct pass to R (JDBC)	3.6	33.01

that when passing a JDBC connection over the service the data is retrieved from the database and then it is written to a text file on the server. So R accesses all data from text files independent of the transfer type of the service (cf. section 4).
Looking at the other results it can be seen that the StatisticsService takes little longer to respond when using the *inQuery* or the *Url* transfer type than accessing the ScatterPlot method directly within R where data are passed as text file. In most instances the response times should be well within acceptable ranges except the transfer of a larger amount of data comprising well more than 50000 rows as well as in the case of more complex calculations.

8 Conclusion

In this paper, we described a web service-based approach for integrating statistical analysis methods in an information system for experiment data.

We started with the introduction of the Biodiversity Exploratories Information System (BExIS) to demonstrate the emerging demands on data management platforms within the life sciences. In particular, there is a growing need for the seamless integration of external tools, e.g., statistical analysis tools, into such systems, for at least two reasons: first, such seamless integration makes the usage of the information systems more attractive to researchers thus helping to convince them to add their data to these centralized repositories. Second, integrated tools allow for easy and convenient data analysis across different projects, thereby supporting the discovery of new and possibly important pattern in data which may lead to new insights into the underlying application domains.

With respect to the need for the integration of external tools, we explained the design of our statistics web service. This service allows the usage of any methods provided by the R statistics tool set from within our information system. We have described the architecture as well as the encapsulation of statistical methods by configuration files. The use of configuration files enables the simple and low effort extension or substitution of methods without the need to touch the underlying statistics program or to code anything within our information system. To abstract from the underlying applications providing the statistical methods we have followed the example of OGC's WPS and introduced a simple query message providing three operations to access information about offered methods, details about the individual methods' interfaces and the possibility to execute any of these methods.

For data exchange between client and web service we implemented three opportunities, namely to transfer data within the query, to pass a URI describing the data location, or to pass a JDBC connection string. Performance tests showed that in the majority of cases the response times are well within acceptable ranges. However, transferring a large amount of data as well as complex calculations can be very time-consuming and could result in a connection time-out. To avoid this problem we implemented a mechanism to initiate a statistical analysis and get informed via email when the process is terminated and where the result data can be retrieved. By now, that is done on server-side by an additional proxy service, but we plan to integrate it into the statistics service.

The proposed web service has been implemented and successfully tested within our project.

Currently, we are working on making its usage even simpler: For the integration of R methods we are working on using configuration files providing the R code instead of implementing each method in Java.

We are aware of code injection security risks, because the executable R script is generated within a Java wrapper class by the concatenation of R code with the committed parameters. Since we are currently using the web service behind our firewall and allowing the access through BExIS only, we are able to control all committed parameters. However, for public release this problem has to overcome by security mechanisms. Besides the restriction of user rights and the R working directory, one solution envisages is an input parameter validation, e.g. during the query translation process.

Another extension we are currently investigating is the integration of other tool sets. Up to now, service implementation is in development and we plan to deploy it within the operative BExIS system for use by the different subprojects of the Biodiversity Exploratories in the near future.

References

[1] Dennis Heimann, Jens Nieschulze, and Birgitta König-Ries. A web service based approach for integrating statistics tools into an information system for experiment data. In *Proceedings of Informatik 2009 - The 39th annual conference of the Gesellschaft für Informatik e.V.*, volume P-154 of *Lecture Notes in Informatics*, page 36, Lübeck, 09/10 2009. Stefan Fischer and Erik Maehle and Rüdiger Reischuk.

[2] HV Jagadish and F. Olken. Database management for life sciences research. *ACM SIGMOD Record*, 33(2):15–20, 2004.

[3] Ulf Leser and Felix Naumann. (almost) hands-off information integration for the life sciences. In *Second Biennial Conf. on Innovative Database Research (CIDR)*, pages 131–143, Asilomar, CA, January 2005.

[4] R. Rifaieh, R. Unwin, J. Carver, and M.A. Miller. Swami: Integrating biological databases and analysis tools within user friendly environment. *LECTURE NOTES IN COMPUTER SCIENCE*, 4544:48–58, 2007.

[5] C. Türker, E. Stolte, D. Joho, and R. Schlapbach. B-Fabric: A Data and Application Integration Framework for Life Sciences Research. In *Data Integration in the Life Sciences 4th International Workshop (DILS)*, pages 37–47, Philadelphia, PA, USA, June 2007. Springer.

[6] M. Rampp, T. Soddemann, and H. Lederer. The MIGenAS integrated bioinformatics toolkit for web-based sequence analysis. *Nucleic Acids Research*, 34(Web Server issue):W15–W19, 2006.

[7] M. Rampp and T. Soddemann. A work flow engine for microbial genome research. *Forschung und wissenschaftliches Rechnen*, 68:30–53, 2004.

[8] Nianhua Li, Martin T. Morgan, Seth Falcon, Robert Gentleman, and Duncan Temple Lang. From r to java: the typeinfo and rwebservices paradigm. Technical report, BioConductor, 2006.

[9] P. Schut. OpenGIS Web Processing Services. OGC Publicly Available Standard OGC 05-007r7, Open Geospatial Consortium, Inc., June 2008. Version 1.0. 0.

[10] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. *W3C Working Draft*, 26, 2004.

[11] W3C. SOAP Version 1.2 Part 0: Primer. *W3C Recommendation*, 24, 2003.

[12] S. Urbanek. Rserve–a fast way to provide r functionality to applications. In *Workshop on Distributed Statistical Computing (DSC)*, pages 20–22, Vienna, Austria, March 2003.