Client/Server based Statistical Computing

Torsten Kleinow[1], and Heiko Lehmann[1]

[1] Humboldt-Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät, Institut für Statistik und Ökonometrie, Spandauer Strasse 1, 10178 Berlin, Germany

Summary

We propose a client server architecture for statistical computing. The main feature of our approach is the possibility to connect various client programs via a TCP/IP connection to a powerful statistical engine. This offers the opportunity to include the statistical engine into a number of software packages and to empower the user of these packages to access a modern statistical programming environment. It also allows for the development of specialized client programs for particular tasks. TCP/IP permits a client/server connection with the client and server running on different hosts (remote host) as well as running both applications on the same computer (local host). To have a large flexibility we suggest adding a middleware program managing the communication between Server and Client. This avoids the need to implement TCP/IP communication methods on the server side. The paper provides an overview of the desired environment and illustrates the general structure by the implementation of the XploRe Quantlet Client and XploRe Quantlet Server.

Keywords: Client/server, Java, Statistical computing, XploRe

1 Introduction

An enormous number of statistical methods have been developed during the last decades, e.g. nonparametric methods, bootstrapping time series, wavelets, estimation of diffusion coefficients. To implement these new methods the method developer usually uses a programming environment he is familiar with. Thus, such methods are only available for a certain software package, but not for widely used standard software packages like MS Excel. To apply these methods to empirical analyses a potential user may be facing a number of problems or it may even be impossible for him to use the methods without rewriting them in a different programming language. Even if one wants to apply the new method to simulated data in order to understand the methodology he is confronted with the same drawbacks. A similar problem occurs in teaching statistics. Since students usually do not have the same statistical software packages as their teacher, illustrating examples have to be executable with standard tools.

In general, two kinds of statisticians are involved in the distribution process of newly implemented methods, the provider and the user. The aim of our work is to close the gap between them. To merge their interests we introduce a *statistical middleware*, a software environment that provides the opportunity to connect user and provider with reasonable effort. For this reason such an environment should have the following features:

1. a sophisticated statistical software including a high level programming language, a developing environment to implement and test methods and a large set of numerical methods
2. distribution techniques for new methods, i.e. a mechanism that makes the new method available to many users in a short period of time avoiding any extra effort
3. an interface for the user, that can be integrated in standard software packages in an easy way
4. documentation tools

A lot of statistical programming environments provide features 1 and 4, e.g. Gauss, XploRe.

If we formulate the second feature for data instead of methods, database servers are supporting it. Data stored in the database are immediately available to all clients of the database server without any distribution effort. This feature is based on the client/server architecture of database applications. We propose to apply the same technology to statistical methods. This leads us to a client/server architecture, where the new methods are stored in a methodbase on the server. In addition we propose to extend the server by the ability to execute the methods, which shifts the computational burden from the client side to a powerful server. Both aspects guide us to a combination of a methodbase and a powerful statistical engine and to use this combination as a statistical computing server.

The user interface is the client part of the client/server architecture. To offer a method to a large community of users Java applets, which allow for integration into web browsers, can be used. They are platform independent and modern browsers already support them without the need for installing additional software. It is the flexibility of Java that makes it useful for teaching statistics. A lot of Java applets are already available to illustrate statistical content. But most of these applets are primarily developed for particular tasks, e.g. visualization. To extend their features new Java programs have to be implemented. To combine the advantages of Java with the client/server concept we propose to implement clients in Java. Our Java client supports interactive teaching of statistics as illustrated by Rönz (2001).

Besides the client described in this paper the client/server technology of XploRe was used in the GraphFitI software developed at Ludwig-Maximilians-Universität München. GraphFitI (`http://www.stat.uni-muenchen.de/`) is a Java program that provides model selection methods in graphical chain models. It uses the XploRe Quantlet Server for statistical computation.

n addition to Java our general concept allows the implementation of other clients that integrate the methods stored at the server into standard software. An example of a non-Java client is the ReX project described by Aydinli et al. (2001) that integrates the statistical engine into MS Excel.

To summarize our approach, the fundamental concept of client/server computing is the separation of a large piece of software into its constituent parts. It thus is creating the opportunity for easier development – for our purposes to integrate new statistical methods – and better maintainability, to have clients and servers running on the appropriate hardware and software platforms for their functions. The client/server architecture is intended to improve flexibility, interoperability, usability and scalability as compared to common software packages.

2 XploRe Quantlet Client/Server Model

The general XploRe Quantlet Client/Server (XQC/XQS) architecture is based on a common three level client/server model as shown in Figure 1. It consists of the main components server, middleware and client (Kleinow & Thomas 2000, Härdle et al. 2001).

A **server** is offering services to one or more client(s). The server of the XQC/XQS architecture consists of the XploRe Quantlet Server (XQS) representing the powerful statistical engine. For server side communication purposes the middleware MD*Serv is attached to the XQS. Both components are described below.

The server offers access to a **data-** and **methodbase**, which contains a variety of methods and data. This easy extendible database ensures the possibility to add newly developed statistical methods and to use them via the client without any changes on the client side.

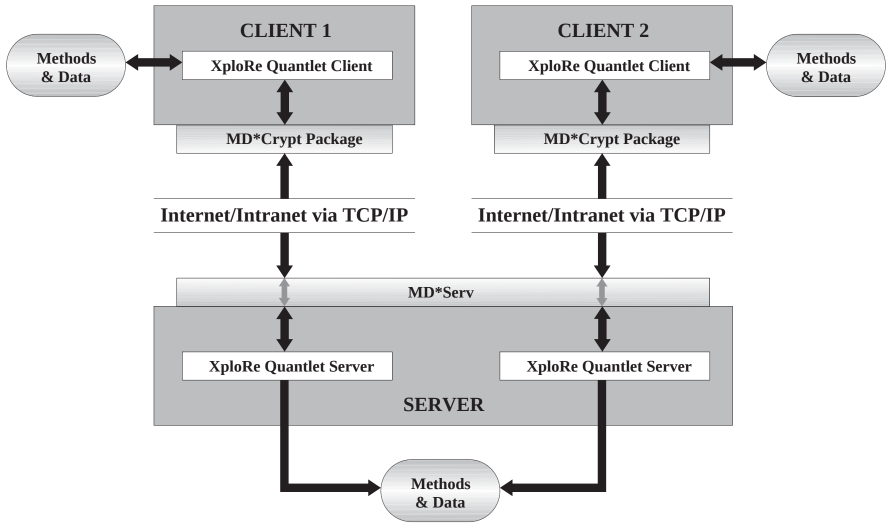


Figure 1: XploRe Quantlet Client/Server architecture

The **client** is the part of the architecture requesting a service. Using the client the user is able to access the statistical methods, data and computing power offered by the server. The XploRe Quantlet Client (XQC) responsible for presenting the statistical results represents the client of the XQC/XQS architecture. For client side communication purposes the MD*Crypt package is attached to the client. This package implements the MD*Crypt protocol which is used as the basis for communication between XQC and XQS. All components are described in the following sections.

2.1 XploRe Quantlet Server

The heart of the client/server architecture is the XploRe Quantlet Server. It represents the back end of the system. The XQS is a powerful computing engine written in C++ that provides a sophisticated statistical programming language. It is based on the statistical computing environment XploRe, which is available for Windows workstations as well as for UNIX platforms (Härdle et al. 1999). Because the XQS is written in native code it enables a fast computation on both platforms. Running on a remote computer the XQS can offer a magnitude of computer power, which many users would not be able to access in other ways. Having access to the method- and database the XQS and the method- and database respectively is easily extendible by new statistical methods via XploRe programs (Quantlets) as well as native code methods, e.g. dll and so. The Communication with MD*Serv is realized via standard I/O streams - the XQS reads from the standard input and writes to the standard output.

2.2 Middleware MD*Serv

MD*Serv works as the "/" (slash) in the XQC/XQS architecture. Being the middleware it provides the communication between the XQS and possible clients, offering the services of the XQS to the clients. MD*Serv is implemented in Java and relies on the MD*Crypt protocol which is necessary for the user to access the network services. This protocol dictates the manner in which the client requests services from the server and how the server replies to that requests.

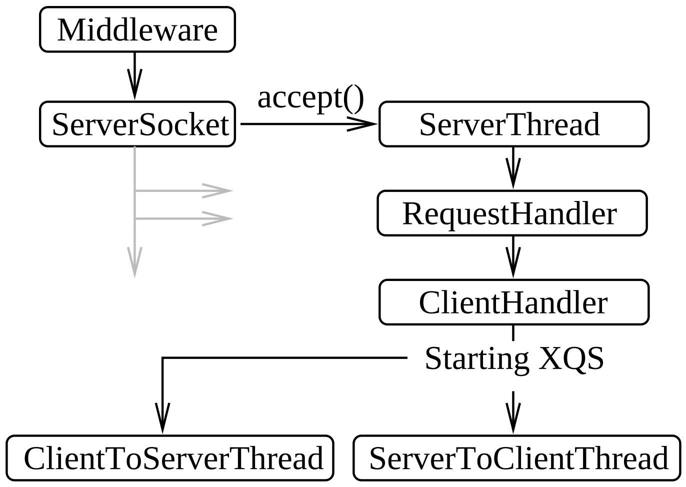


Figure 2: MD*Serv structure

Due to the design of the client/server model XQS and MD*Serv can be run as part of a wide area network (WAN) or/and a local area network (LAN) –

remote host – as well as on the same computer as the client - local host.

Figure 2 visualizes the MD*Serv structure. Right after MD*Serv has been started it binds a socket to a specific port stated in the configuration file on the server. MD*Serv is listening to the socket for client requests. If a client knows the host name of the machine on which the server is running and the port number to which the server is connected it tries to rendezvous with the server. MD*Serv will start a new *ServerThread* to handle the request. MD*Serv works as a parallel application and is ready for another client request. After successfully identifying the client via the arranged protocol a new XQS process is started. This process works exclusively for the requesting client. A *ClientToServerThread* transmits data coming from the client via TCP/IP to the XQS using its standard input stream. The other way around, a *ServerToClientThread* transmits data read from the standard output of the XQS via TCP/IP to the client. The current version of MD*Serv is able to handle up to 50 clients at the same time.

2.3 MD*Crypt Package

The MD*Crypt package supports the client side communication between client and server in the XQC/XQS model. It is implemented in a single Java package to make it available to different clients, e.g. XQC and GraphFitI, without double programming effort. Besides the Java package there also exists a MS Windows dynamic link library (dll) for the use in native Windows applications like Excel (Aydinli et al. 2001). The MD*Crypt package gets in and keeps contact to MD*Serv using the MD*Crypt protocol. Via TCP/IP incoming data are prepared to be available to the client in an easy accessible way.

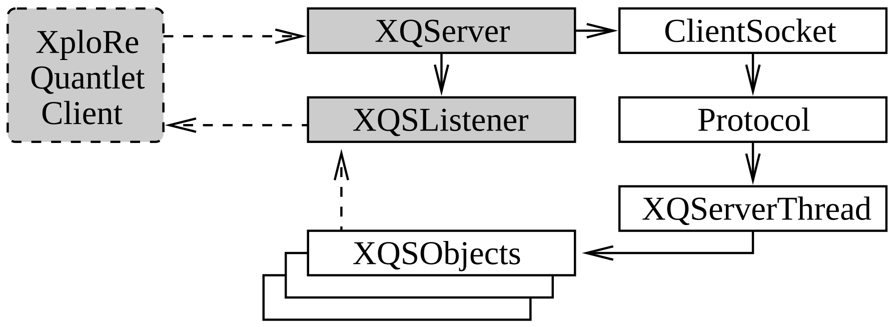


Figure 3: MD*Crypt structure

The MD*Crypt package being set up in between the XQS respective the MD*Serv middleware and clients mimics a server to possible front-end clients (see Figure 3). After the acceptance of a connection request by MD*Serv the MD*Crypt package creates a client side socket that is used for further communication between both applications and starts an *XQServerThread* to handle the connection. The communication to the XQC takes place via a common listener interface implemented by the client. The exchange of data is based on the MD*Crypt protocol that is based on TCP/IP. Depending on the kind of data that arrives a new *XQSObject* is created and passed to the client. This *XQSObject* contains methods to access the content of the object. Each client class that has implemented the *XQServerListener* and is also registered receives the information about the arrived object and can thus process it. For further details regarding the MD*Crypt package (Feuerhake 2001).

2.4 XploRe Quantlet Client

The XploRe Quantlet Client (XQC) represents the front end – the graphical user interface (GUI) of the discussed XQC/XQS architecture. The XQC is fully programmed in Java2 to make use of the advanced graphical features (e.g. line thickness, anti aliasing). Using a pure Java solution the XQC does not depend on a certain computer platform. Running as an application or a certified applet the XQC can access its own local database containing statistical methods and data.

The GUI is based on the MS Windows XploRe version to ensure an easy handling and a familiar look. As shown in Figure 4 the XQC consists of the following main components.

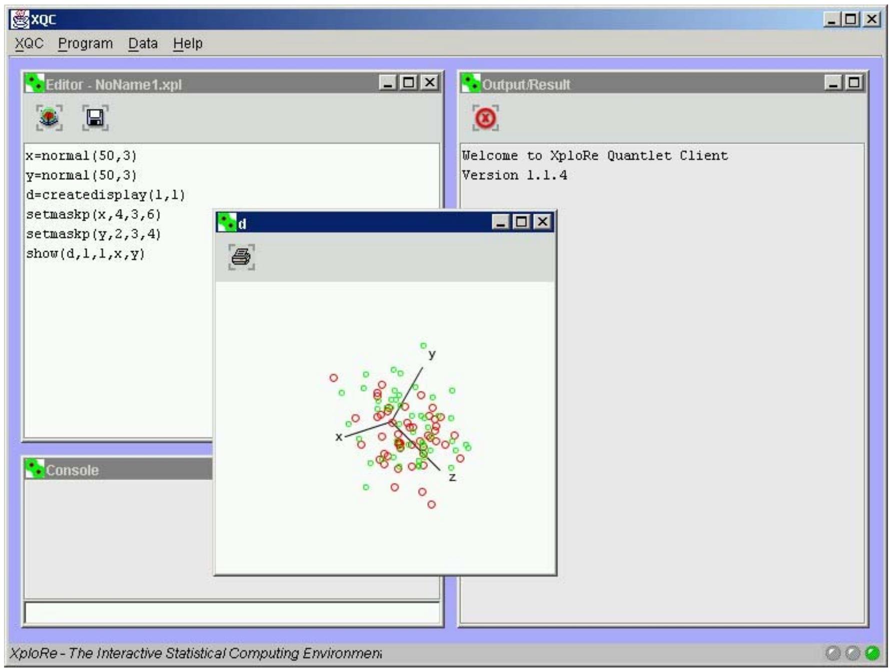
The **Desktop Frame** is contains all graphical components. It offers a menu bar for basic features like connecting and disconnecting to the XploRe Quantlet Server, program exit, opening editor and data window and online help access.

The **Output/Result Frame** displays text output and information transmitted from the server.

The **Console** offers direct access to the XploRe Quantlet Server. Users can enter single line XploRe commands. In addition it keeps a history of the last 20 executed commands.

The **Editor Frame** offers a text area for editing and executing more than just a single line command. It can be used to work on and execute a complete XploRe program (Quantlet).

The **Data Frame** is also programmed as a text area for handling datasets and uploading the data to the XploRe Server in order to use them within programs. If run as an application or a certified applet it is possible to open and save programs and data locally. In addition the *copy* and *paste* features



```
x=normal(50,3)
y=normal(50,3)
d=createdisplay(1,1)
setmaskp(x,4,3,6)
setmaskp(y,2,3,4)
show(d,1,1,x,y)
```

```
Welcome to XploRe Quantlet Client
Version 1.1.4
```

Figure 4: Screen shot of the XQC

of the underlying operating system are usable, too.

Display Frames visualize graphical output. Three-dimensional plots can be rotated for recognizing structures within the data. A print routine allows for either printing the whole display or just a single plot.

To ensure user interaction within a running program the XQC also contains dialog frames for functions such as *read value* and *select item* based on the XQS dialog structure.

The technical structure of the XQC is mainly based on the GUI structure shown in Figure 5. Each of the described components is implemented in its own Java class. The main class *XQClient* represents the desktop itself. It holds and manages the other classes and components respectively. Since a display can consist of more than one plot, the *XQDisplay* class manages all plots.

From the XQC's point of view the MD*Crypt package behaves like a server. To send data to the XQS the *XQClient* uses the *XQServer.sendQuantlet* method provided by the MD*Crypt package. *XQConsole*, *XQEditorFrame* and *XQDataFrame* send their content using the *XQServer.sendQuantlet*

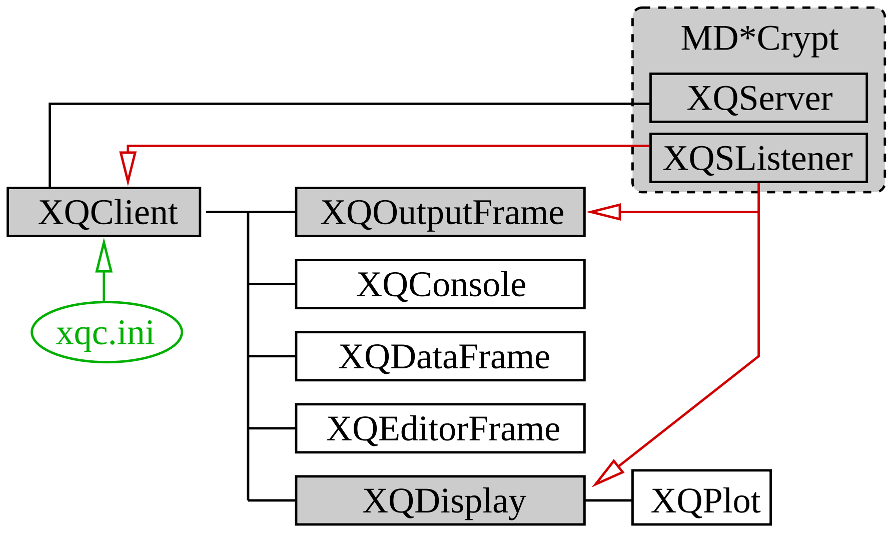
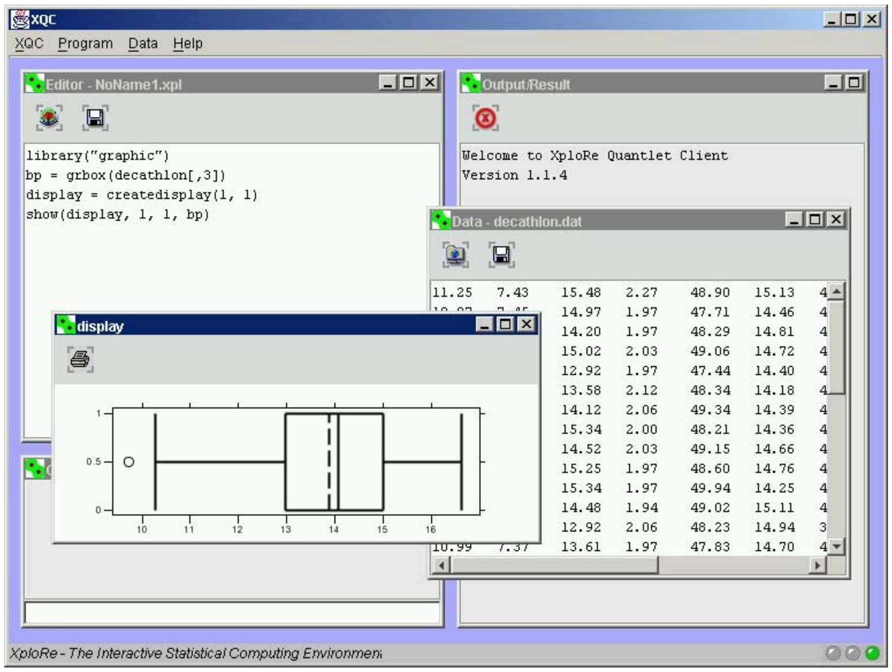


Figure 5: XQC structure

method via the *XQClient*. To receive information and data from the XQS the appropriate classes implement MD*Crypt's *XQSListener*. This listener works just like a common listener in the Java programming language. If any results from the XQS are available, each of the classes gets notified. The MD*Crypt package also sends the result as a certain object (e.g. *XQSOutputObject*, *XQSGraphicsObject*) depending on the information received from the XQS. Each of the XQC classes handles only the objects intended for this class.

Figure 6 shows an example of how to work with the XQC. After uploading the dataset ("`decathlon.dat`") from the data window a simple Xplore program is executed. It generates a BoxPlot for the third column of the dataset by using the Quantlet "grbox" out of the XploRe library "graphic". A help system and introduction to the XploRe language is available online at `http://www.i-xplore.de/help/_Xpl_Start.html`.

A configuration file allows for customizing the XQC to meet special needs and thus to manage the appearance and behavior of the XQC. Using the settings it is possible to easily embed the XQC into multimedia contents. It could be started with executing a certain Quantlet stated in the configuration file without displaying console or output frame. In this case the XQC behaves like a Java applet programmed for a particular task. It could also be started with just opening a certain Quantlet that can be edited and manually executed. Because of these attributes and the platform independence the XQC recommends itself for the integration into HTML and PDF contents for visualizing



```
library("graphic")
bp = grbox(decathlon[,3])
display = createdisplay(1, 1)
show(display, 1, 1, bp)
```

```
Welcome to XploRe Quantlet Client
Version 1.1.4
```

```
11.25   7.43    15.48   2.27    48.90   15.13   4
                14.97   1.97    47.71   14.46   4
                14.20   1.97    48.29   14.81   4
                15.02   2.03    49.06   14.72   4
                12.92   1.97    47.44   14.40   4
                13.58   2.12    48.34   14.18   4
                14.12   2.06    49.34   14.39   4
                15.34   2.00    48.21   14.36   4
                14.52   2.03    49.15   14.66   4
                15.25   1.97    48.60   14.76   4
                15.34   1.97    49.94   14.25   4
                14.48   1.94    49.02   15.11   4
                12.92   2.06    48.23   14.94   3
10.99   7.37    13.61   1.97    47.83   14.70   4
```

Figure 6: XQC working example

statistical and mathematical coherences (Rönz 2001, Klinke 2001).

3 XQS/XQC Compared to Other Web-based Statistical Solutions

Searching the Internet for net-based statistical solutions leads to three different approaches:

1. CGI techniques
2. “standalone” Java applets
3. java based distributed computing

Using CGI techniques the user enters data or the location of a data file via a CGI interface. A statistical program on the server side calculates and sends back the results to the user. The user will get the results either right away - shown in the browser window or the result will be sent to the user by e-mail. Examples are given by Inoue et al. (2001), the **MMM** project (`http://macke.wiwi.hu-berlin.de/mmm/`) the Rweb project (`http://www.math.montana.edu/Rweb/`). The advantage of the CGI approach is the use of an architecture that is similar to the client/server architecture. With the statistical program running on the server side the user can access resources of a powerful computing system as offered by our XQC/XQS approach. The disadvantage of the CGI is the lack of interactivity. A CGI program works as a new individual process against a HTTP request without any communication taking place between different processes.

Interactivity is an advantage offered by the use of Java applets. Most statistical (standalone) applets available via the World Wide Web have two things in common – they are completely programmed in Java and integrated in one single applet. Therefore, the user has to download the whole program containing the computation algorithm as well as routines for presenting the results. Examples of such statistical applets are the Internet Statistical Computing Center (`http://www.statlets.com`) and the WebStat project of the University of South Carolina (`http://www.stat.sc.edu/webstat/`). Our XQC/XQS architecture on the other hand splits the load between the client and the server. The Java client only has to present the output computed by the server. Therefore it can be a relatively slim application.

For calculating just a simple histogram of a small dataset the use of a single Java applet would be an appropriate way. But the more complex a statistical algorithm gets and the larger the datasets are the more difficult is a pure Java implementation of these algorithms and the more load lies on the client computer. Computing a nonparametric time series process that contains about a thousand observations within a single Java applet is hardly possible. The computational load of the XQC/XQS model lies on the server side that can take advantage of a powerful underlying computer architecture. This speeds up the computational process significantly. Due to the communication process between client and server which takes place the XQC might take a little longer for calculating simple statistical problems compared to a pure Java applet. But with increasing complexity the time saved using the server power exceeds the time needed for the communication process.

Extending an existing Java applet with a new statistical method implies a high effort for reprogramming the method. The new method, usually developed using a statistical software package, would have to be reprogrammed in Java. Extending the XQC/XQS model just means to add the new method programmed in XploRe to the server's or to the client's methodbase without any reprogramming effort on the client or server side.

Besides our XQC/XQS project there exist other projects using client/server approaches for statistical computing via the Internet. One example is the Jasp project (`http://jasp.ism.ac.jp/index-e.html`), see Kobayashi et al. (2001). Jasp is a statistical system whose language is based on Pnuts (`http://javacenter.sun.co.jp/pnuts/`). Like the XQC/XQS model the Jasp approach uses the advantages of Java and Java applets respectively to implement a user interface. The user interface – a mixed user interface consisting of a CUI as well as a GUI – is an advantage of the JASP project over the current version of the XploRe Quantlet Client. It only offers a CUI, where knowledge of the programming language XploRe is required to perform statistical computing. But the XQC is not only meant to work like a "conventional" program - the advantage of the XQC is the possibility to customize its behavior via a configuration file. This characteristic offers a way to extend the features of electronically enhanced books (e-books) towards interactive examples. The Jasp approach allows for distributed computing on several servers whereas in the XQC/XQS model a client chooses a certain server that computes the data of this (and possible other) client(s) during the entire session.

Another example for a java based statistical computing environment is the Omega project (`http://www.omegahat.org/`). The aim of this project is to provide a variety of different modules (GUI, Graphics, CORBA, language, numerical methods, etc.) that can be combined by the user to meet his particular needs. Distributed computing is supported by the integration of the CORBA interface, which provides access to remote servers. Of particular interest for the Java programmer is the org.omegahat.R.Java package, which provides several methods to evaluate R code directly in Java and to get back the results of the evaluation. For an overview of the available features see Duncan (2002). An integration of this package into the Middleware MD*Serv can extend our architecture to handle R requests. In addition to the interaction between R and Java, omegahat offers another way to access R resources via the Web, namely the R-plug-in for Netscape. This plug in allows calling R from JavaScript. Unfortunately it is only available for Unix operating systems.

Pure web-based client/server approaches suffer from well-known problems of the Internet - the security of data transferred via the Internet, the stability and the speed of the network/modem connection that may represent the bottleneck of the client/server architecture. Encrypting the data could solve the security problem. To take advantage of the server's speed a fast and stable network/modem connection is required for the transport of data and results. The quite fast technical development in this area should help to solve this problem in the future.

4 Conclusion

Our approach combines the possibilities of a powerful statistical software environment with the advantages of distributed applications and the opportunities offered by the Internet. The result is a statistical package usable via the World Wide Web that behaves like a traditional statistical software package without the need of installing the whole software package.

The XQC/XQS architecture offers a sophisticated statistical software including a high level programming language easily accessible via the Internet, a distributed computing environment for balancing the load between client and server and an easy way of distributing new statistical methods using the configuration possibilities of the XQC. Nevertheless it should not be seen as a substitute for “traditional statistical software packages but as an additional tool for statistical computing.

Right now the XQC represents itself mainly as a programming environment. For access to the methodbase the user has to know the name of the needed method - have to be familiar with the XploRe language. One of our next planned steps is an extension of the XQC for an easier access of the existing methods of the server’s methodbase via the XQC GUI.

Further developments will be presented at `http://www.i-XploRe.de`.

Acknowledgements

Financial Support was received by the Deutsche Forschungsgemeinschaft, SFB 373 (“Quantification und Simulation Ökonomischer Prozesse”),Humboldt-Universität zu Berlin.

References

Aydinli, G., Härdle, W., Kleinow, T. & Sofyan, H. (2001), Rex: Modern statistical tools in office applications, *Proceedings of the ISM symposium “Statistical software in the Internet age”*, 101–109, Institute of Statistical Mathematics, Tokyo.

Feuerhake, J. (2001), *MD*CRYPT - the XQS/XQC protocol*, `http://www.md-crypt.com`.

Härdle, W., Kleinow, T. & Tschernig, R. (2001), Web quantlets for time series analysis, *Annals of the Institute of Statistical Mathematics* **53**(1), 179–188.

Härdle, W., Klinke, S. & Müller, M. (1999), *XploRe – Learning Guide*, Berlin: Springer (`http://www.xplore-stat.de/`).

Inoue, T., Asahi, Y., Yamamoto, Y. & Yadohisa, H. (2001), A prototype of Data Representation System, *Proceedings of the ISM symposium "Statistical software in the Internet age"*, 85–90, Institute of Statistical Mathematics, Tokyo.

Kleinow, T. & Thomas, M. (2000), Computational resources for extremes, *Measuring Risk in Complex Stochastic Systems* (Franke, J., Härdle, W. & Stahl, G. eds.) *number 147 in Lecture Notes in Statistics*, New York: Springer-Verlag.

Klinke, S. (2001), MD*book - a tool for creating interactive documents, *Proceedings of the ISM symposium "Statistical software in the Internet age"*, 75–84, Institute of Statistical Mathematics, Tokyo.

Kobayashi, I., Fujiwara, T., Yamamoto, Y. & Nakano, J. (2001), The Language and the Extendibility of the Statistical System Jasp, *Proceedings of the ISM symposium "Statistical software in the Internet age"*, 65–73, Institute of Statistical Mathematics, Tokyo.

Lang, D.T. (2002), *Calling R from Java*, `http://www.omegahat.org/RSJava/RFromJava.pdf`.

Rönz, B. (2001), The multimedia-project mm*stat for teaching statistics, *Proceedings of the ISM symposium "Statistical software in the Internet age"*, 27–31, Institute of Statistical Mathematics, Tokyo.