DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

JavaStat: A Distributed Statistical Computing Environment

Hengyi Xue[*] **E. James Harner**[†]

Abstract

JavaStat is a distributed application written in Java for do statistical analyses. It is designed to run on multiple computers connected by the Internet. The data model and the associated statistical logic of JavaStat reside on the server while plots and reports and their controllers are located on the clients. The client communicates with the server using Java RMI (Remote Method Invocation). JavaStat can be used as a standalone application to perform traditional data analyses or it can be used for collaborative research among a group of users. For instructional purposes, it is also possible to use Java applets as the user interface. Statistical models or plots are initiated by the user from a dataset (viewed as a table or a variable list), which is obtained from the server. JavaStat allows the user to do standard analyses and advanced plotting, but it was not designed for complex modeling tasks. Instead, R will be used as a backend computing engine and an Omegahat-type interface will be developed between R and JavaStat. JavaStat is a component of a more comprehensive system being developed called JEMS (Java Environment for Mathematics and Statistics).

1 Introduction

JavaStat is a distributed Java application for statistical computing. It is designed to run on multiple computers connected via the Internet. The data model and the associated statistical logic of JavaStat reside on the server whereas plots and plot controllers are located on the clients. JavaStat can be used as a standalone tool to

[*] `hxue@stat.wvu.edu`, Department of Statistics, West Virginia University
[†] `jharner@stat.wvu.edu`, Department of Statistics, West Virginia University

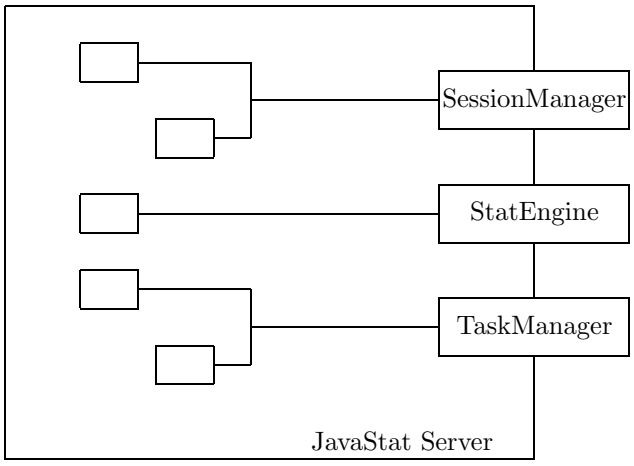
do traditional data analyses or it can be used for collaborative research among a group of users.

The software has a client-server structure. It is organized around the concept of a session. A session stores information about a client, including its name and the data it uses. The data of a session is stored in a DataSet object. A DataSet is a collection of variables with state information. The server module of the software runs on a server machine and its function is to manage sessions for the clients and perform computing tasks. The clients construct reports and graphical views of the datasets. They are lightweight in the sense that they rely on the server module for complex computing algorithms and operations.

There are two types of clients. One is implemented as a Java application. This client gives the user flexibility as to which dataset to choose and which graphical views of the data to construct. It also allows the user to modify the data and permanently save the changes. This type of client is usually used by a more advanced user—like an instructor or researcher. Another type of client is implemented as a Java applet. This client is usually a single view of the dataset such as histogram or spreadsheet. Applet clients dont give their users a choice as to which dataset to analyze or the module to display the data. The parameters are pre-determined for the applet in its initialization and cannot be changed dynamically. The advantage of this type of client is that its code can be downloaded dynamically and it can be embedded easily within static Web content.

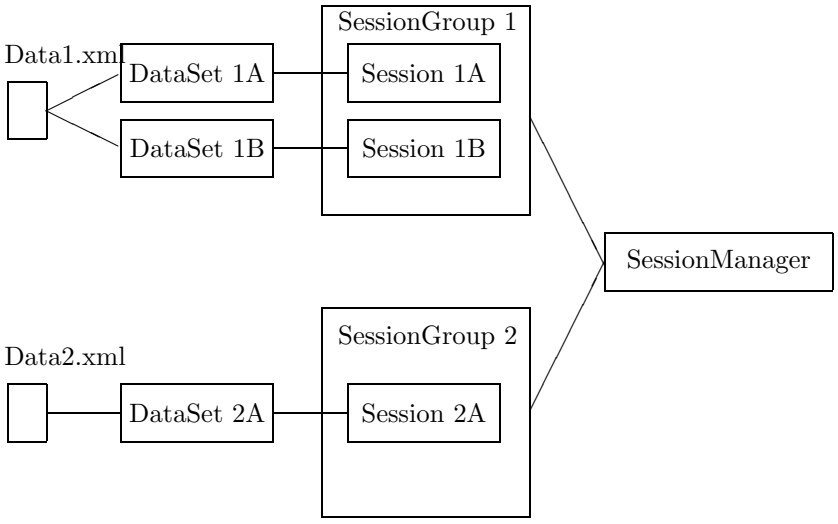
2 The Server Module

The server module manages sessions, performs complex computing tasks on behalf of the clients, and coordinates interaction among collaborative objects.



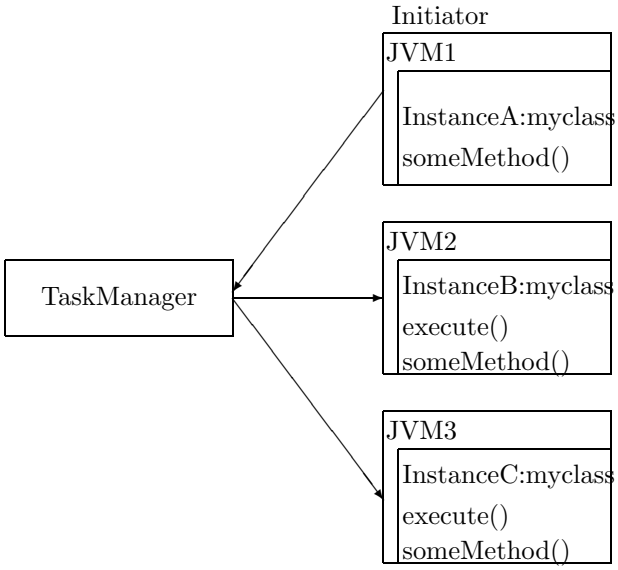
It has three major components:

- **SessionManager:** This component manages sessions. There is one dataset associated with each session object. The dataset is read and parsed from an XML file saved on the server. Session objects are organized into SessionGroups based on the associated datasets. Sessions that use the same original dataset belong in the same group. In normal running mode, there is one session for each client. On the other Hand, in collaborative running mode, multiple clients can share a session.



- **StatEngine:** The StatEngine component implements mathematical and statistical procedures used in JavaStat. All computationally intensive operations are passed by the client to StatEngine, which completes the task and returns it to the client. StatEngine users MathUtils, for doing mathematical computations, and Distribution, for manipulating probability distributions.

- **TaskManager:** This component coordinates interaction among collaborative objects to achieve a high level of collaboration among a group of clients. A collaborative object is an object whose methods can be invoked remotely by an object of the same class on another Java Virtual Machine (JVM). The basic idea is that when a method is invoked on a collaborative object, it broadcasts the method invocation to objects of the same class on other JVMs so that they behave as a whole. Because client-to-client communication in Java is forbidden for security reasons, the TaskManager acts as the middle man for the broadcasting. It receives the request from one client and sends it to the rest.



3 The Client Module

The principal responsibility of the client is to access and display data views. The classes in the client are organized into 'modules' according to their functionality. For example, a histogram view of a dataset derives from the HistogramModule. The modules can be used in either a client Java application or an applet.

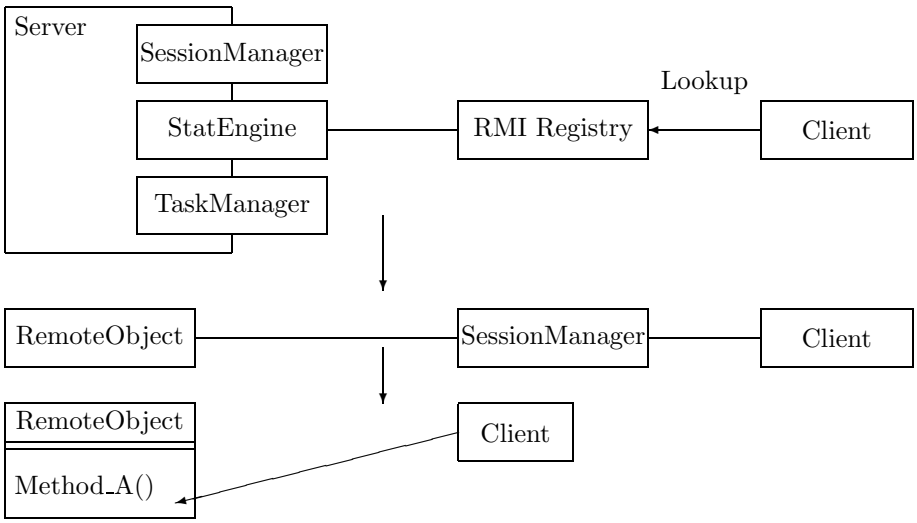
The client communicates with the server using Sun's Java RMI (Remote Method Invocation) protocol. This protocol allows complex objects to be passed between different JVMs. The binding of the client with the server is a three staged process:

1. The client looks up registered remote objects through a facility called the RMI Registry. The registered remote objects serve as entry points for the server.

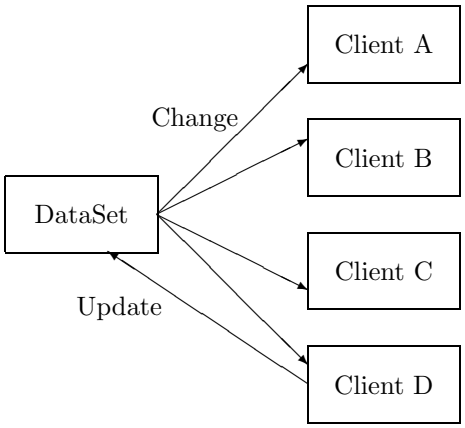
2. Through the remote objects found in stage 1, the client can obtain references to other remote objects.

3. Once the client has a reference to a remote object, it can call its method directly as if the object is local. The object's physical location is transparent to the client.

The communication process is illustrated in the following diagram:

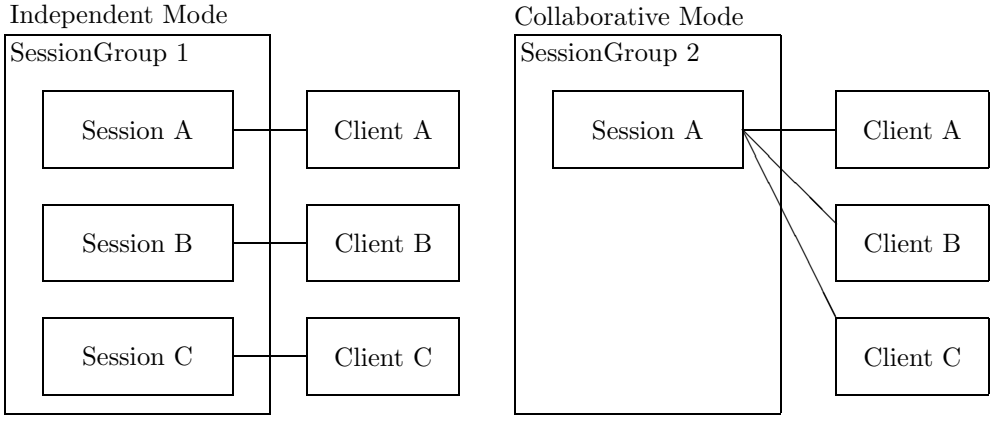


A client acts as a remote observer for a DataSet and a DataSet can have many remote observers. A privileged client can make changes to the DataSet. Any change will cause all remote observers to be updated.



As mentioned before, the JavaStat client can run in standalone or collaborative mode. In independent mode, each client will have its own Session. The changes a client makes to its Session DataSet will not affect the datasets of other clients even if they are sharing the same XML data file. In collaborative mode, clients in the same SessionGroup have the option of sharing a single Session. The Session is

owned by a particular client. The owner of the Session can make changes to the DataSet and all the other clients will reflect the change dynamically. Other clients can not modify the data unless the owner grant them privileges.



To begin a collaborative session, the session initiator (e.g., the instructor) will run JavaStat in collaborative mode. Other users can then login using either Java applet or application clients.

3.1 The User Interface for Data Modeling

After a client acquires a dataset from the server, it can display it in either of two views—as a table or variable list. The table view gives a user a data table that the user can use for viewing, changing or inserting data. The variable-list view is used for data modeling. In this mode, the user can see which variables are categorical and which are numerical. The user can also assign roles to the variables to do data analysis. Each variable can have one of three roles: x, y or z, corresponding to whether it is an explanatory variable, outcome variable or conditioning variable.

An icon with dotted lines indicates the variable is categorical, whereas an icon with solid lines indicates the variable is numerical. The three axis of an icon corresponds to the three roles. For example, if a variable has the 'y' role, then the y axis is painted red. The user can assign roles by clicking on one of the axis or by assigning the role in a dialog.

After the roles are assigned, the user can perform data analyses. This is done through the `Plot` menu. Based on the roles of the variables, the `Plot` menu will have different menu choices for the user to choose. If the data set has y and x variables, the menu will have choices that say $y \mid z$ plot and $y \sim x \mid z$ plot. Each of these menu choices will do different default data analyses based on the characteristic of the variables.

3.2 Data Level Collaboration and User Interface Level Collaboration

The collaboration implemented in JavaStat can be classified into two categories: data level collaboration and user interface level collaboration.

Data level collaboration means any changes to the data set by one client are visible to all the other clients. This is achieved by implementing the DataSet object as a remote object. It always resides on the server. Changes by a client are actually made to the object on the server. The other clients are notified using an RMI callback mechanism. This is in fact the model-view-controller design pattern implemented in a distributed environment with the DataSet being the model and all the clients being the view.

User interface level collaboration means that changes made by one client, not affecting the dataset, are nonetheless visible to the other clients. For example, if a user changes the number of bins in a histogram, which will not affect the dataset, then the other collaborative users will see the corresponding changes in their histograms as well. We achieved this by implementing some of the UI components as collaborative objects. We define collaborative objects as distributed objects that react to a single method invocation as a whole. Consider the histogram example: 1) a user uses a UI gadget to increase the number of bins in the histogram; 2) the event handler will respond to the user interaction and invoke a method on a UI component to update the histogram; 3) the method executes but also broadcasts a message to the same UI components on all other sharing JVMs to execute the same method. When UI collaborative objects are instantiated, each of the instances will export its stub to the server. The server will keep a registry of these stubs. When one of the collaborative objects method is invoked, it sends a message to the server to tell all the other instances to execute the same method. The server interprets the message and invokes the method on all the stubs, which are delegated to all the distributed instances through the Java RMI protocol.

3.3 The JavaStat Evaluator

Both the table and variable list views of a dataset have a 'Data' menu with an 'Evaluator' menu item. The Evaluator is used for creating new variables (columns in the table view). Generally, this is done by constructing a variable expression (saved as a MathML expression). If a value of an underlying variable is changed, the variable expression is automatically updated. The changes are propagated to all clients.

Variables can also be generated from analyses, including those from remote R analyses. The Evaluator can be used to initiate these remote R function calls.

4 R as a Computing Engine

The StatEngine component in the server module has limited modeling capabilities. Specifically, JavaStat can generate plots and reports from models of the form $y \mid z$

or $y \sim x \mid z$, where x, y, and z are categorical or numerical and z is an optional conditioning variable. For example, if both x and y are categorical, then a mosaic plot is created by default and a contingency report is generated. Alternative plots and reports are often available, e.g., those associated with correspondence analyses in the case of categorical variables.

The use of R within JavaStat in not intended to replicate the full functionality of R. Rather, R will be used to enhance JavaStat—initially by using the advanced modeling functions of R. The R Interface to Java (http://www.omegahoat.org) by Duncan Temple Lang makes this possible. The interest here is to allow R functions to be used to implement Java interfaces. With this interface, we can access R modeling functions from JavaStat.

When we make R function calls from within Java, we need to consider how to call R functions and how to pass arguments from Java to R and get returned results back into Java. The approach here is to do as much work as possible in JavaStat and to do as little as possible in R. This will facilitate the portability of JavaStat.

4.1 Calling R functions

There are two ways to make R function calls from Java. One is to pass an R expression as a string to the R evaluator. The limitation of this method is that it is difficult to pass JavaStat variables as arguments to these calls.

The second way of calling R, which is used by JavaStat, is to call R functions in a Java-like mechanism. We can identify the function by name and pass objects to it using ordered arguments or named arguments in the form
`REvaluator.call(functionName, argArray, namedArgTable)`,
where `REvaluator` is implemented in `org.omegahat.R.Java`.

Standard conversion mechanisms are used when the inter-system interface is used to convert the Java arguments to R objects.

4.2 Returning R objects to Java

The result of invoking an R function is an R object—typically a list. When this is passed back to the Java method that called it, standard mechanisms for converting R objects to Java objects are applied. If no converter is found (either user-level or built-in), a proxy for the R object is returned to Java in the form of an `RForeignClass` object.

5 Future Work

JavaStat is undergoing extensive development and a production versions is expected in August of 2001. JavaStat is designed for doing statistical analyses. A companion component for manipulating mathematical expressions, called JavaMath, is also being developed. JavaStat and Javamath both have the same underlying structure based on a collaborative session. Therefore, the SessionManager and TaskManager

of JavaStat are being moved to a higher level in a system called JEMS (Java Environment for Mathematics and Statistics). Currently, JavaMath is in its initial stage of development and is principally used for manipulating mathematical expressions dynamically as represented in a graph.