Using Java to Front SAS® Software: A Detailed Design for Internet Information Delivery

Jonathan Stokes, JJT™ Inc., Austin, Texas, USA

ABSTRACT

As the Java language evolves from its rudimentary stages into the next generation for software development, it proves to be a perfect tool for creating robust front-end interfaces to database reporting. The analysis capability of the SAS System, along with the flexibility of its programming language, makes SAS the ideal solution for server-side reporting. This paper outlines a specific design of a Java Internet reporting application using SAS/GRAPH®, and explains the strengths and weaknesses of the implementation.

INTRODUCTION

The Java language and environment, developed by Sun Microsystems®, has brought on a new era in software applications. Network applications — small applications loaded entirely from a server and executed — are replacing traditional client-server applications.

To keep up with technology, developers at JJT set out to design a front-end to SAS reports implemented in Java. Programming features such as multi-threading and network communication make Java an excellent environment for both user interfaces and console applications.

This paper discusses in detail a specific design created by JJT Inc. for a thin-client reporting interface. First, the application objectives will be covered, followed by the general design, an overview of the components, and an explanation of the communication between components. The next section will address using SAS on the server, followed by the overall advantages and disadvantages of this design. Lastly, future enhancements will be considered for this application design.

OBJECTIVES

IMPROVED INFORMATION DELIVERY

The overall objective of Internet reporting is to efficiently deliver information as requested to a vast number of users. In this design, several intelligent front-end interfaces allow simple pick-and-view reports (as well as complex reports) to be requested. The following objectives result in response to several flaws in common web interfaces.

Provide Smart User Interfaces

HTML Forms provide fill-in-the-blank questionnaires but have no logic behind selections. For example, a form's date selector will usually allow users to select the 31st of February, relying on the server to inform them that the selection was invalid after *submitting* the form.

Java offers a more intuitive UI through a not-quite-as-thin client. Modular components can provide anything from intelligent mm/dd/yy fields to pop-up calendar dialogs.

Avoid Static Wait-Time

One of the important advantages of multi-threaded technology is the ability to track progress on the front-end.

Reports requiring large volumes of data can take some time to process. Using an HTML Form interface and CGI scripts, the browser merely *pauses* after the user "submits" the report request. There is no indication of when (or even if) the report will return its output.

In contrast, multi-threaded Java front-ends allow progress bars to show what the server is doing or to load partial output while the report is still running.

Allow Interaction with Viewed Output

With data warehouses becoming more prominent, there is increasing demand for interactive drill-down reports. Using Java DataBase Connectivity (JDBC) and the *SAS JDBC driver* provided by SAS Institute, Java applets and applications can directly read SAS datasets hosted by SAS/Share servers. Server reports can partially summarize data and allow the client to create graphs and charts by further summarizing the subset. This allows the user to view summary information and then drill-down into more detailed output.

MINIMIZE COSTS

Java reporting can reduce long- and short-term costs in several ways:

Reduce Software Costs

Because the client only needs a web browser, the cost of licensing, installing, and maintaining software on each client machine is minimized.

Reduce Hardware Costs

Portable code on both clients and servers provides flexibility with hardware systems. (For example, the Report Request Server and Web Server do not have to be on the machine that contains the SAS System or the data.)
Consequently, hardware can be upgraded or exchanged without the concern for compatibility.

Reduce Maintenance Costs

All the application files are hosted by the web server and sent to each client on demand, therefore, maintaining the code is very efficient. Defect fixes and enhancements can be added without having to re-install application code on each client.

DESIGN

OVERVIEW

Figure A shows a simple overview of the design:

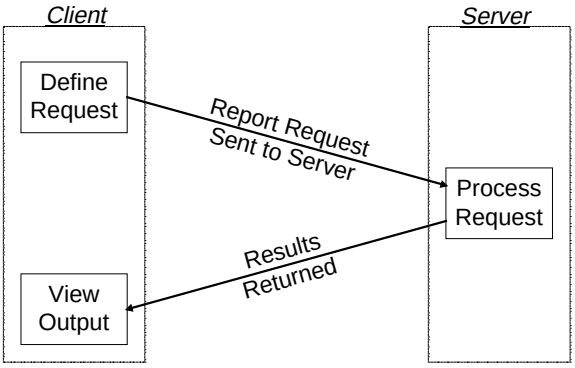

Figure A

The user defines a report and makes a request, which is sent to the server. The server processes the report and returns output which can be viewed by the user.

COMPONENT DESIGN

Of course, nothing is quite that simple. Figure B shows the same design with more details about the processes on the server and the communication between them.

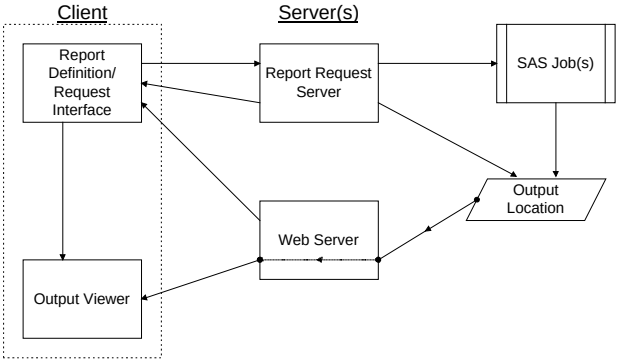

Figure B

The communication between components is detailed in a later section. Descriptions of the components follow.

COMPONENT OVERVIEW

OVERVIEW OF THE CLIENT COMPONENTS

Report Definition/Requester

The main applet provides an interface for the user to pick various reports and fill-in-the-blanks to define the request. Once the request is defined, the user can *process* the report.

Output Viewer

The Output Viewer displays the output returned from a report. It provides scrollbars to navigate through several pages of report output. The viewer is responsible for polling the server until all the output has been received.

OVERVIEW OF THE SERVER COMPONENTS

Report Request Server

The Report Request Server is a stand-alone Java application that uses TCP/IP sockets to receive report requests from the end-users. The requests are handled by spawning SAS processes and returning the output locations to the front-end.

Web Server

An HTTP web server is used to read the main HTML page. In addition, images (charts and graphs) and text output files created by reports are read by the client via the web server.
Because the Java security manager does not allow applets to read or write anywhere except the server from which they came, the Web Server and the Report Request Server must be on the same machine.
This design could be modified if the client applets ran as *trusted* applets or in a stand-alone application (outside of the browser).

SAS Process(es)

The Report Request Server spawns a SAS process to fulfill each report request. These SAS jobs are given a location to find the request definition and a location for the resulting output.
These processes do not need to execute on the same machine as the Web and Report Request servers, and may access data on any server.

COMMUNICATION

The various components involved must effectively communicate with each other. In this design, the main Report Definition Applet sends requests to the Report Request Server via TCP/IP sockets using a predefined protocol. The server creates a temporary storage area, writes the report definition to a file in that area, spawns a SAS job to deal with the request, and returns to the Applet a URL (web address) for the temporary location.

SEQUENCE

The following diagram (Figure C) labels the sequence of communication between components.

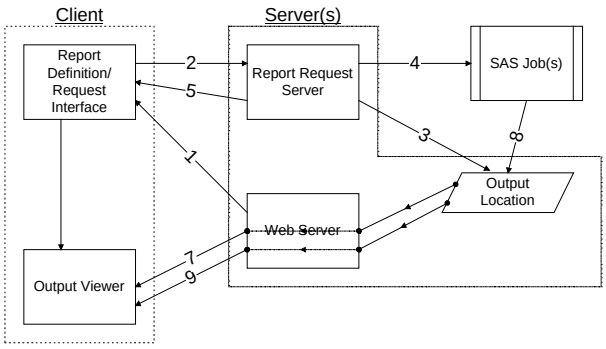

Figure C

1. The client browser loads HTML page and applet code from the web server. The applet executes and the user defines a report.
2. The report definition is sent to the server.
3. The Report Request Server writes the report definition to a newly created temporary output location.
4. The Report Request Server dispatches a SAS process to handle the report.
5. The Report server returns the URL of the new output location.
6. The main report applet creates an Output Viewer if needed and sends the URL to the viewer.
7. The Output Viewer polls and reads the table-of-contents file.
8. The SAS process writes the output and table-of-contents to the output location.
9. The Output Viewer loads the output files for viewing.

PROTOCOLS

Components that communicate must use established methods to transfer information. General protocols used between components are described below.

Report Definition Applet To Report Request Server

The Report Definition Applet uses a simple, pre-defined protocol for communicating to the Report Request Server via TCP/IP sockets.

Report Request Server to SAS Process

The details of a report request are written to a file in the temporary directory. When the SAS process is spawned, a `-sasuser` option sets the SASUSER library for the session to the temporary directory. The `.sas` program knows to look in its SASUSER directory for the report details and to create its table-of-contents and dynamic output in this directory.
Other variations could include setting the SYSPARM macro variable to the URL of the temporary location.

Report Definition Applet to Output Viewer

The main interface must also communicate with the Output Viewer(s) created when a report is processed. After requesting a report, the Report Server returns a URL for a table-of-contents in the temporary directory. The Report Definition Applet creates an Output Viewer pointed to that URL. The Output Viewer can then poll until the table-of-contents is

complete, loading output files as they are created. Since both the Report Definition Applet and the Output Viewer are Java objects in the same run-time, regular method calls are used to create a viewer or add reports to an existing viewer.

Output Viewer to Report Request Server

When report output is discarded, a simple message is sent from the Output Viewer to the Report Request Server to allow the server to delete the temporary output directory.

THE RESULT

REPORT DEFINITION

The Report Definition/Requester interface is the main applet of the front-end. Users can pick from a list of available reports and then fill-in-the-blanks to define the details for each report.

The outer applet class is a generic viewer that displays a list of report classes defined in the HTML tags of the containing web page.

When a specific report class is chosen, an instance of that class is asked to fill in the appropriate components (list-boxes, check boxes, text fields, etc.) for the details of that report.

Figure D shows an example of a report definition within the main applet.

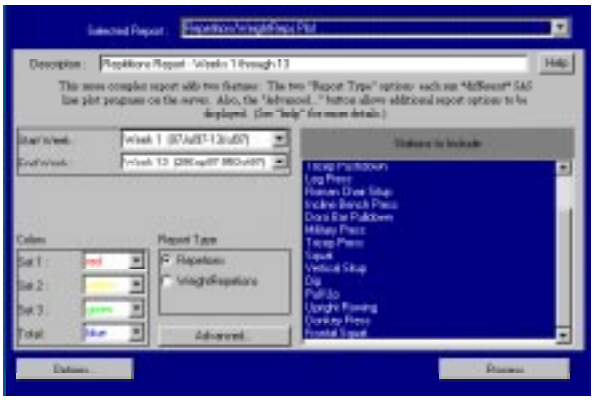

Figure D

The specific implementation of the report definition interface is expected to evolve over time. It can be revised and customized without affecting the overall design.

What is important in this case is that the outer applet "hosts" other *report* classes, each of which is expected to support a certain interface. As long as the report can provide the information needed for the server and the Report Request Server has a report *handler* class that corresponds, the specific format of the report definition data and interface is impertinent.

OUTPUT VIEWER

The Output Viewer is given a URL that points to a table-of-contents file on the server. This file is built (using a predefined format) by the SAS process handling the report request.

The table-of-contents file contains a list of URLs pointing to the output files for the report. This allows the SAS job to create new *dynamic* output or point the viewer to existing *published* output, such as a monthly report.

The table-of-contents approach also allows reports to deliver *any* type of output. For example, a single report could return a text page, an image (chart or graph), an HTML page, *and* a table of data.

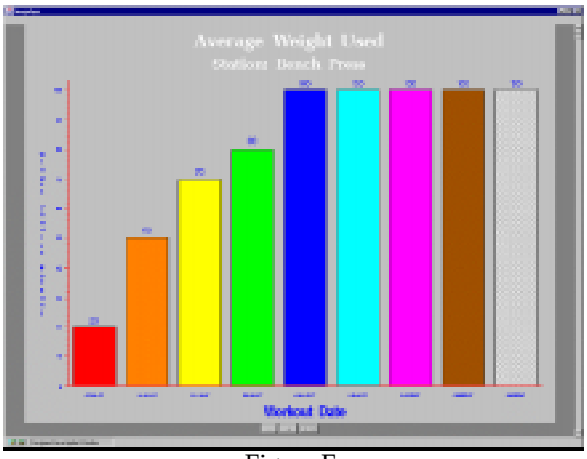

Figure E

Figure E shows a sample image shown in the Output Viewer. Images can be created by SAS/GRAPH procedures using image-format device drivers.

The viewer above provides zoom in/out capabilities and can be scrolled from page to page within the report.

WHY SAS?

This over-all design is not dependent on SAS for reporting. However, there are several reasons SAS proves ideal for server-side reporting.

POWERFUL STATISTICAL GRAPHING

SAS/GRAPH provides the capability to generate detailed and accurate graphs on summarized or raw data. Report writers have the ability to greatly

customize textual and graphical reports. In addition, the ability to direct graphical output into image files is important for web publishing.

FLEXIBLE DATA MANIPULATION

The base SAS language gives report writers the ability to manipulate text files, which is important in building the table-of-contents files used in this design. A reporting application that does not include a capable programming environment would require "wrapper" code to be written around the report in another language (i.e. C or Perl).

Furthermore, many variables in report output are data-driven. For example, the number of pages created by a procedure using BY-group processing is dependent on values in the data. For this reason, having a rich language with both data and file manipulation capabilities makes the SAS System an excellent environment.

CONNECTIVITY TO ANYWHERE

A powerful advantage that SAS provides is the ability to connect to almost any relational Database Management System (DBMS) using SAS/ACCESS products. This allows flexibility and scalability in the hardware system design. The data used by web report interfaces can be housed on many different machines, which can be upgraded as needed to support demand.

DESIGN ADVANTAGES

CROSS-PLATFORM DEVELOPMENT

There are numerous advantages to developing robust software that can be executed on any platform. Client-side components can be executed in any browser on any operating system. Server-side processes can be moved across machines, and servers can be upgraded without the traditional cost of porting code.

INTELLIGENT USER INTERFACES

Graphical Java components can create the intuitive interfaces expected of today's software. For Internet applications, this reaches beyond HTML-forms and server-dependent CGI programs into the well-developed realm of desktop application interfaces.

MULTI-THREADING

Multi-threading allows an application to perform two actions simultaneously. Java provides a simple yet powerful API for using threads. Multiple threads are utilized when Output Viewers load report output while the user defines and processes other reports. Threads can also be used to display progress while performing lengthy tasks.

EASE OF DISTRIBUTION

Maintenance is the main reason thin-client applications are increasingly preferred over traditional client-server designs. The cost of installing, supporting, and maintaining an application on hundreds or thousands of PCs or workstations can be fatal to development projects.

Java applets are maintained entirely on the server, and distributed automatically by the web server to each client browser. Distribution costs are minimized, and users can configure the browser of their choice with the options they prefer.

DISADVANTAGES

Two main disadvantages were known when this design was first considered: the slight complexity of the design and the immaturity of the Java language.

JAVA IS NEW

This application design was first implemented using the first major production release of Java, version 1.0.2. While many improvements have been made in the Java 1.1 API, it is still a new and improving environment. There were no show-stopping problems with the 1.0.2 implementation of this application. However, the myth of *"write-once, run anywhere"* has yet to be fulfilled. Slight compensations were made to accommodate portability problems.

COMPLEXITY

While this design boasts a wealth of advantages, the complexity of the components involved can be intimidating. Modular components help to simplify functions by encapsulating the complexity, but for simple static web publishing, such a design could be overkill.

A point worth noting is that the original implementation of this application was on one single-processor Unix machine, containing all of the web server, Report Request Server, and SAS installation.

FUTURE ENHANCEMENTS

One of the many advantages of object technology is the ability to enhance some components without affecting others. Several future enhancements were considered when designing this application.

DISTRIBUTED PROCESSING

Expanding on the simple implementation of this design, the Report Request Server could spawn the SAS report processes on different server-side machines. This could easily be done by report handlers, which could partition certain reports to run on particular server machines.
More difficult would be to distribute processing to different servers based on processor load.

SERVER LOAD BALANCING

Balancing the load on various machines could be approached in several ways. The simplest might be based on the number of processes spawned. Approaches that are more complex include monitoring CPU usage or number-of-executing-processes for the available servers.
Load balancing logic could be handled by the Report Request Server.

ENHANCED OPERATION IN TRUSTED ENVIRONMENTS

Releasing the Java applet from the restrictions of the Java security manager opens several avenues for enhancements. (Security can be lifted by *trusting* a signed applet or placing the applet in a stand-alone Java environment.)
Profiles could be saved for individual users to store report preferences, user interface options, etc.
The Report Request Server would not be required to reside on the same host as the web server.
Although Intranet applications can easily execute in insecure environments, this design focuses on providing solutions in secure environments to include distribution of regular Internet web applications.

SUMMARY

A specific design for an Internet reporting front-end implemented in Java has been outlined. The various components, and the communication between them, create a robust user interface for reports created in SAS. This design provides an efficient way to deliver a powerful thin-client application. The advantages of implementing Internet applications in Java far outweigh the disadvantages. In addition, the SAS System has proven the most effective environment for server-side reporting and output manipulation.

CONTACT INFORMATION

The author may be contacted at:

Jonathan Stokes
JJT Inc.
1610 West Ave.
Austin, TX 78701
jonathan@jjt.com

A demo of the application described in this document can be found at:

```
http://lonestar.jjt.com/sugi23
```