

Server-Side Image Processing using the World Wide Web

Daniel Tretter and Andrew Patti
Hewlett-Packard Laboratories
Palo Alto, CA 94304

Abstract

In many image management and image processing scenarios, it is advantageous to have a central repository for storing and processing the images, while users can interact with the images from multiple remote locations. This client-server architecture allows the system resources and image data to be concentrated in a single location, which simplifies system maintenance and dissemination of the data. In this paper, we describe a system in which a web browser is used to access and manipulate image data on a remote server. The images are stored in multiple databases on the server. Clients can navigate and alter these databases as well as select and process individual images. Client interaction is managed through a Java applet, while CGI programs on the server perform the necessary calculations and data manipulations. The client and server communicate through CGI protocols and a secondary socket connection. The architecture supports multiple clients, with changes initiated by any individual client immediately propagating to all affected clients. This paper will describe the system in detail and discuss its advantages and disadvantages.

Introduction

In the past few years, the world wide web has gained great popularity as a vehicle for distributing information, including both text and images. As a result, client browsers are widely available for most computing platforms, and many computer users already have significant experience working with them. We would like to leverage this existing codebase and the corresponding well-defined communications protocols to construct a distributed image processing system, where multiple clients interact with images that are stored and manipulated on a server. We limit ourselves to a system architecture that does not require any special client software (other than a commercially available web browser) or any specific client platform. In this way the system is easily accessible and more likely to attract users.

Our system concentrates the processing on the server rather than on the client platforms. This approach does not require any assumptions about the processing capabilities of the client machine, so the system will work for so-called "thin clients", which have very little processing power. Concentrating the processing on the server also gives us full control over the processing software. Only a single version of the code is in use, so it can be easily updated or modified without having to be redistributed to all the users. This ensures that all clients are using the same version of the code at all times since no code is stored locally on the client.

Unfortunately, server-side processing also has several drawbacks. Since the processing is concentrated on a single machine, the system is not scalable. Even a very powerful server can be overwhelmed if the number of active clients grows too large. Also, for security and privacy reasons the server is not able to access a client's file system. If a client wants to process a local images file, that file must first be uploaded to the server. The server then processes the file, and the output image is downloaded by the client. The uploading and subsequent downloading can add significant overhead to the processing task, particularly if multiple images or large images are being processed.

Distributed imaging systems using the world wide web have been developed by several researchers. Meng, Chang, and their associates at Columbia University have developed a distributed system for searching and editing video databases[3, 5]. Their system, like ours, uses Java on the client side for maximum portability, but also requires a Netscape plugin to be installed on the client. Bamberger proposes a system for distance learning that allows image processing to be specified by a client via the world wide web. His approach uses CGI scripts on the server and HTML forms on the client side[1]. Berman et al. propose a distributed system to allow clients to access a database of X-ray images[2], and Hasegawa et al. devise a multimedia database that allows users to request image processing via the web and view the results[4].

We will next discuss the architecture of our system,

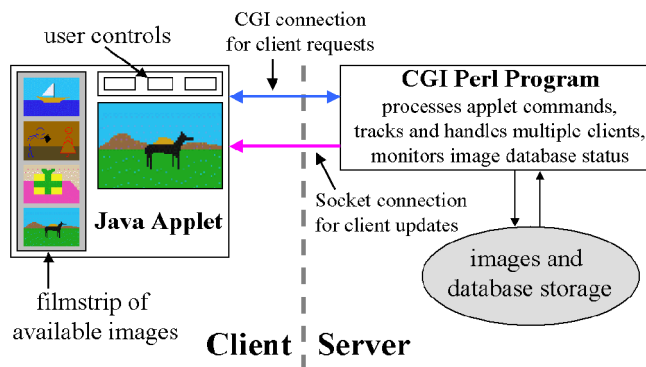


Figure 1: CGI scripts on the server communicate with a Java applet on each client. The data and processing reside on the server, while the user controls the processing through the applet.

and then we go through a brief example interaction to illustrate how the system operates. A more detailed description of our system can be found in [7].

Architecture

System Overview

Figure 1 illustrates our overall system architecture. We use CGI (Common Gateway Interface) Perl scripts on the server to perform the necessary server-side processing. CGI allows the web server to communicate with other programs on the server.

In typical CGI systems, a user enters information into an HTML form and submits it to the server through a web browser. The web server passes the form data on to CGI programs on the server that process the data and generate a result, often an HTML page, that gets sent back to the client's web browser. Our system uses a Java applet on the client in place of HTML forms to handle user input and interpret server responses. Java allows the user to view and interact with the images in ways not possible through HTML. In addition, Java allows the use of a secondary sockets connection, which allows the server to initiate contact and update each client when server status changes. This second connection will typically fail if the client is connecting through a proxy, so our system will usually not operate properly across a firewall.

Our system stores images in simple databases, which we represent as filmstrips, with each frame of the filmstrip a different image. Images and databases are stored and maintained on the server. Users can access and modify existing databases, create new databases, or process individual images in a database. Images can be included in multiple databases simultaneously, and databases can contain any number of images. Users can only access images through a database, so all images must be in at

least one database. When a user creates a new image either by processing an existing image or by uploading an image from the client, the new image must be added to a database immediately if it is to be kept on the server for future user access.

Since we allow multiple simultaneous users, we encounter many of the same problems seen in any distributed database application. We need to track each client's activity and current state, keep clients up to date when other clients make database changes, and prevent conflicts from occurring when multiple clients access the same database simultaneously. These tasks are the responsibility of the server, which handles all client interactions and updates the clients using a second sockets connection outside the CGI protocol. These issues are covered in more detail below, where we discuss the server responsibilities.

One issue we do not address in our system is the problem of security. The normal web protocols and Java provide some basic security (the Java applet cannot access the client filesystem, users cannot modify CGI scripts, etc.), but a number of security problems still exist. From an individual user's point of view, we would like to implement protections to prevent other users from modifying or even accessing user-created databases and images without permission. This would probably require some sort of password protection for both individual images and entire databases, perhaps in conjunction with Netscape cookies to allow automatic recognition of users. Similarly, the server may want to maintain some read-only databases, and each user should be limited to a pre-defined amount of disk space on the server. Otherwise, a user could upload many very large images and overload the server. These issues, although not insurmountable, have not been addressed in our existing system.

Server Side

Server-side responsibilities fall into three major categories, and the software code structure reflects this division. The three categories for which the server is responsible are client handling, image and database handling, and image processing. The server software defines separate Perl classes for clients, databases, and processing algorithms. Each client or image database is a separate instance of the corresponding class, while the processing parameters and algorithm choice are carried out using the processing class.

Client handling includes storing and tracking multiple client states, communicating with the clients, and preventing client collisions from occurring. When a new client initiates a session, the server assigns a unique identifier and communicates this number to the client. Each subsequent request from the client will be prefaced by the client with the identifier, so the server can automatically identify the request with the client. This identification mechanism is necessary because client requests use the

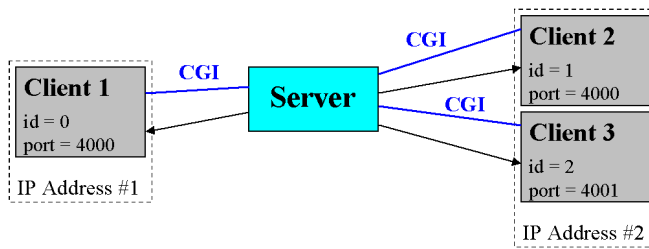


Figure 2: The server communicates with each client through both a CGI connection and a separate sockets connection. The client identification numbers are used to associate CGI requests with clients, while the IP addresses and port numbers are used by the server to specify update targets.

CGI connection with the standard http protocol, which is a stateless protocol. The server maintains no information between transactions, so each request must stand alone, and any tracking required of the server needs to be implemented explicitly.

When the server generates an identification number for a new client, it also creates a state file for that client. In this file the server stores information identifying the database being used by the client, the current knowledge of the client, and the last time this client contacted the server. The active database and current knowledge of the client are needed so the server knows when the client needs to receive an update (client knowledge is stale) and so the server can correctly interpret client requests regarding individual database images (i.e. add new image to the active database, etc.). The contact time is used by the server to determine when a client is no longer active. If no requests have been received for a certain period of time, the client is declared inactive and its identification number is cleared for use by a new client. This approach is similar to the one used in [2].

Figure 2 illustrates how the server identifies and communicates with multiple clients. The unique client identification number allows the server to keep track of each client, but this mechanism only works if the client initiates the contact. Occasionally, the state of the server itself will change. For instance, when a client modifies an image database, other clients need to be informed of this change. The server cannot initiate a connection through the standard web protocols (this sort of connection requires what is referred to as a push channel). We therefore open a separate sockets connection between the server and the applet to allow this. Each client's IP address is stored by the server, and the clients are each assigned a port on which to "listen" for updates. The port number must be assigned dynamically by the server in case multiple clients share an IP address, as illustrated in Figure 2. If too many clients share the same IP address, the probability of a port conflict may become

unacceptably high.

Client collisions are prevented using a simple "first come, first served" strategy. When a client is in the process of modifying a database, the database is locked to prevent other clients from accessing it. Other clients using that database are then updated through the sockets connection and the database is unlocked for access. In this way, only one client at a time can be modifying any given database.

The server is also responsible for handling the images and databases. Whenever a new image is generated or acquired from a client, two scaled copies are automatically generated and stored on the server. A thumbnail version is used to represent the image in the database filmstrip, and a somewhat larger view image is displayed by the applet when the user requests the image from the database. The original image, which may be of higher resolution than either of the others, is used for processing and downloading to the client if desired. In addition to the multiple image resolutions, the server also keeps track of how many databases are using each image. If an image is ever removed from the only database that currently includes it, it is automatically deleted from the server since clients will no longer be able to access it.

Our system allows a user to specify a processing algorithm to be performed on the chosen image. The processing occurs on the server, which stores the output image in a temporary location until the user decides what to do with it. The user can choose to replace the original image with the processed image, either in the active database only or in all databases that contain the image. The user can also choose to download the processed image to the client, add it to the active database, delete it, or process it further.

The processing module on the server has been designed to allow new algorithms to be added easily without significantly changing the code. All processing algorithms are assumed to take a single input image and produce a single output image. The client is allowed to specify a variety of processing parameters, including integer, floating point, and binary parameters as well as image regions.

Client Side

The client side is responsible for allowing the user to interact with the database and providing a reasonable means of specifying image processing operations that will be carried out on the server. These tasks necessitate a reasonably functional GUI to be implemented at the client side. In addition, the client must be able to accept updates regarding the server state that may occur at any time. Lastly, the client must provide a mechanism for uploading to the server images that reside on the client. This task is especially challenging because Java does not allow direct access to the client local storage.

The client architecture that provides required func-

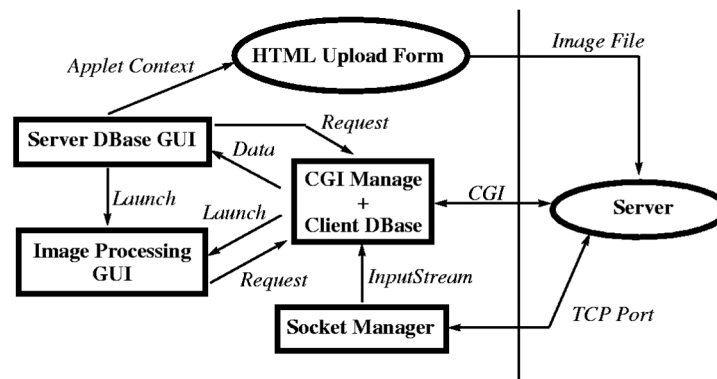


Figure 3: Client Java applet classes (boxes) in relation to the server and HTML forms (ellipses).

tionality is depicted in Figure 3. In the figure, Java classes are depicted by rectangular boxes, while non-Java entities such as HTML forms and server CGI scripts are depicted with ellipses. The client functionality is divided into classes in the following manner. A local database is maintained at the client via the “CGI Manage + Client DBase” class. This class manages all CGI-based connections to the server and can be thought of as a local server, since the GUI classes act as clients that request the local data.

The “Socket Manager” class maintains a thread which monitors a TCP port for updates from the server. When the server contacts the client to provide an update, this class forwards the information to the “CGI Manage + Client DBase” class in the form of a Java *InputStream*. This enables the client-side data to be updated and maintained in a single place.

The “Server DBase GUI” class presents the user with the capability to select an image strip for viewing, manage the strip display, initiate an image upload to the strip, and select an image for viewing and manipulation at full resolution. The image strip is composed of thumbnails loaded from the server into the client database. To display the strip, a separate thread is managed that continuously scrolls the strip of images. The user is also provided a scrollbar to directly control strip scrolling. To allow client image upload, this class initiates a CGI request to the server, but uses the *Applet Context* to redirect the server response to an HTML form that resides in a separate HTML frame on the page containing the applet. Because the form does have access to the local storage this Java-based system therefore allows for client upload. One last note is that this class is the Java *Applet* derived class that the browser launches.

The last client class, “Image Processing GUI”, allows for viewing and manipulation of the full sized image. This class is launched by the “Server DBase GUI” in response to a request to view an image in the strip, or when the server has finished applying an image processing task and the image is to be presented to the user. The

GUI allows the user to select a processing algorithm and provide pixel-oriented input such as selecting specific regions of the image to apply the processing to. The actual request for processing on the server is directed through the “CGI Manage + Client DBase” class. Upon completion of the processing, the client is alerted via the TCP socket connection and a new viewer containing the processed output image is launched by the “CGI Manage + Client DBase” class.

Case Study

Now that we have described the architecture of our system, we will go through an example interaction to illustrate its operation. Figure 4 illustrates a typical session. We will follow this general outline in our case study.

Suppose a new client accesses the system by entering the appropriate URL in a web browser. The browser encounters and downloads the Java applet automatically. At this point, the browser has instantiated the “Server DBase GUI” class. The applet begins to run on the client machine, establishing a CGI-based connection with the server. This is carried out by the “CGI Manage + Client DBase” class. The server will generate a unique identification number and a port number for this client. This information, along with a list of image database choices and an initial default database, will be sent back to the client. The server also creates a state file for this client, storing the same information just sent to the client plus a time stamp to indicate the last time of contact with this client. Before generating an identification number for the client, the server checks the time stamps for all existing clients and removes those clients that haven’t made contact recently enough.

The client receives the information from the server through the CGI connection and stores it locally. The applet opens a TCP socket connection on the specified port and starts up a thread to “listen” for server updates. This is accomplished by instantiating the “Socket Man-

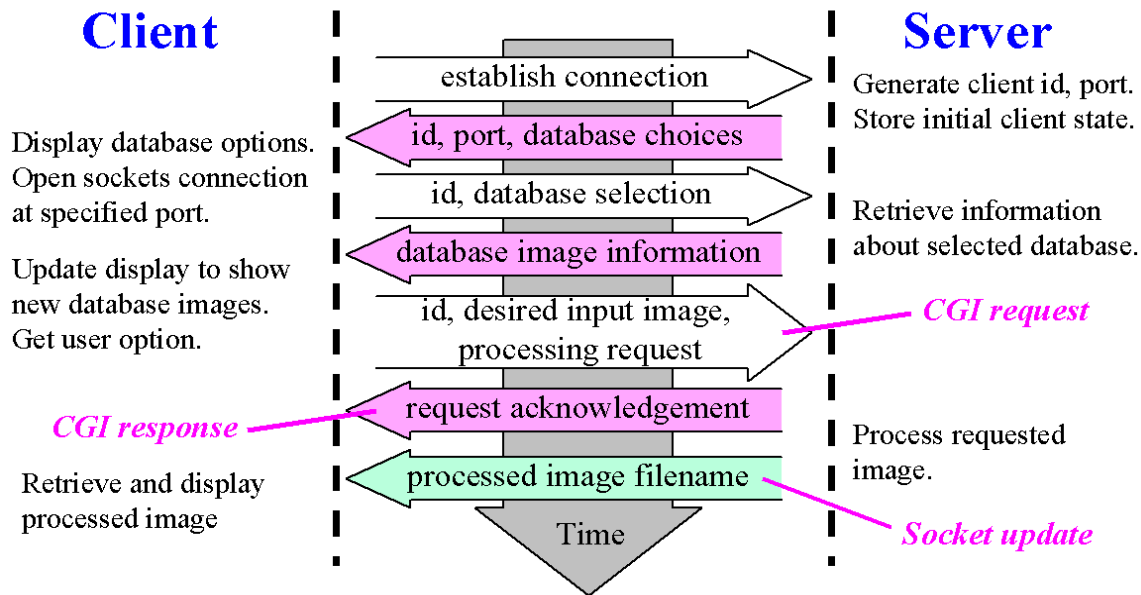


Figure 4: This diagram indicates the communications for a typical client-server session. Note that the client sends its identifier with each request.

ager” class. It also gets the thumbnail images for the initial database from the server and generates a screen for the user showing the database selections and the initial database. Figure 5 shows a screen shot of what the user sees at this point.

Suppose the user selects a different database from the list. The “Server DBase GUI” class passes this request to the “CGI Manage + Client DBase” class, which contacts the server and updates the local database. Upon receiving the request from the applet, the server responds with a list of images contained in this database. The applet then retrieves thumbnails of the indicated images from the server and updates the user display. The server updates its state file for this client to reflect the database change.

Now suppose the user selects an image from the database by clicking on one of the thumbnails in the filmstrip. The applet retrieves the higher resolution version of the image from the server and launches the “Image Processing GUI” to display it in a separate window. The user might then elect to use the mouse to select an image region for processing. Figure 6 shows a screen shot of what the user might see at this point. In this example, we are using a redeye reduction processing algorithm[6], so we have selected an eye region. For some processing algorithms, the user might select parameters other than an image region. After selecting a processing algorithm and parameter values, the user inputs a “process image” command.

The processing request will be passed to the server through the “CGI Manage + Client DBase” class, and the request will include the input image name, the cho-

sen algorithm, parameter values, and its client identifier. The server responds first by acknowledging the request. The acknowledgement is sent back to the applet, freeing it up so the user can continue to interact with the image database while the server is performing the desired processing. The server then forks off a child process to perform the requested image processing. After the processing is completed and an output image is available, the server sends the name of this processed image to the applet through the secondary TCP socket. The applet then retrieves this information via the “Socket Manager” class, which ultimately causes the processed image to be displayed via the “Image Processing GUI” class. The display window then allows the user to specify further actions on this image (ie. saving, further processing, etc.).

Conclusion

We have proposed and implemented an architecture for performing image processing and archiving on the web. The proposed system uses server-based processing, and no special software needs to be installed on the client. By using a combination of Java, CGI, and Perl programs, users are allowed pixelwise interaction with the images, and the server software can be easily upgraded without requiring a change in client software. Future versions of such a system would benefit from the use of Flashpix format images, which incorporate the desired multiple resolutions. This would eliminate the need to generate and store multiple versions of each image.

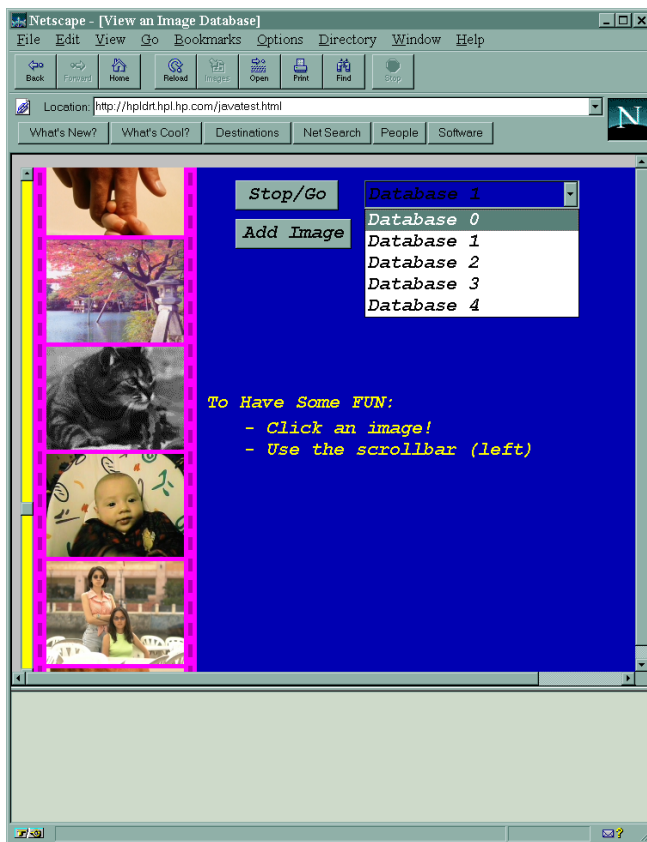


Figure 5: The user initially sees a scrolling filmstrip of image thumbnails and a list of available databases.

References

- [1] R. H. Bamberger. A prototype distance learning laboratory for image processing education. In *Proc. 26th Annual Frontiers in Education (FIE) Conference I*, pages 51–54, November 1996.
- [2] L. E. Berman, R. Long, and G. R. Thoma. Challenges in providing general access to digitized x-rays over the internet. In *Proceedings of the 23rd AIPR Workshop*, pages 183–193, Washington, D.C., October 1994.
- [3] Shih-Fu Chang, John R. Smith, and Horace J. Meng. Exploring image functionalities in www applications – development of image/video search and editing engines. In *Proc. IEEE International Conference on Image Processing III*, pages 1–4, Santa Barbara, CA, October 1997.
- [4] T. Hasegawa, T. Minamihaba, M. Daibo, T. Kumagai, M. Fujisawa, and N. Tayama. Development of interactive multimedia database. In *Proc. SPIE – volume 2915*, pages 183–190, November 1996.
- [5] H. J. Meng, D. Zhong, and S.-F. Chang. Webclip: A www video editing/browsing system. In *Proc. IEEE*

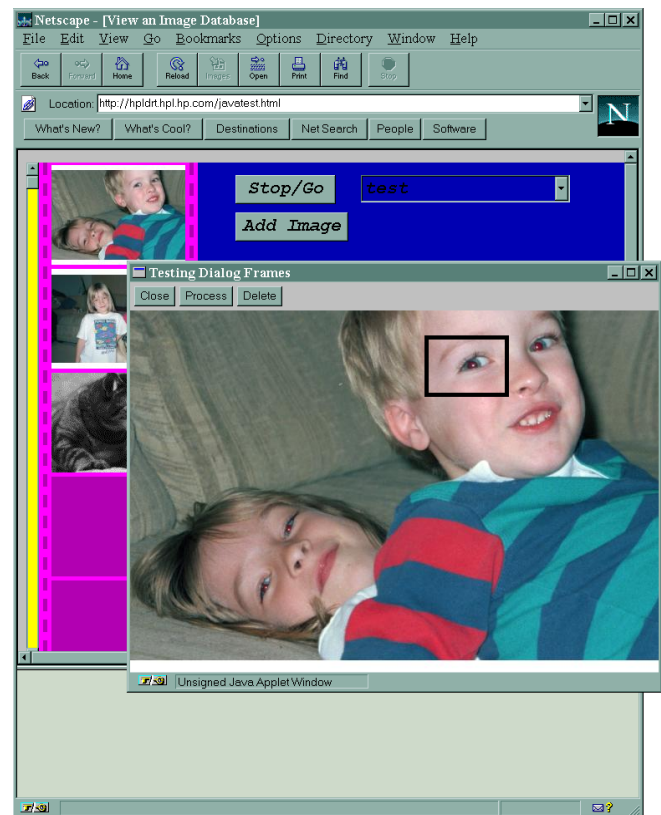


Figure 6: After selecting an image, the user can interact with the chosen image, using the mouse, for example, to select an image region.

1st Multimedia Signal Processing Workshop, Princeton, NJ, June 1997.

- [6] Andrew Patti, Konstantinos Konstantinides, Daniel Tretter, and Qian Lin. Automatic digital red-eye reduction. Technical Report HPL-97-74, HP Laboratories, June 1997.
- [7] Daniel Tretter and Andrew Patti. A distributed image processing system using the world wide web. Technical report, HP Laboratories, 1998. in preparation.