Diamond Eye: A Distributed Architecture for Image Data Mining

Michael C. Burl, Charless Fowlkes, Joe Roden, Andre Stechert, and Saleem Mukhtar

Jet Propulsion Laboratory, MS 126-347
4800 Oak Grove Drive
Pasadena, CA 91109 USA

ABSTRACT

Diamond Eye is a distributed software architecture that enables users (scientists) to analyze large image collections by interacting with one or more custom data mining servers via a Java applet interface. Each server is coupled with an object-oriented database and a computational engine such as a network of high-performance workstations. The database provides persistent storage and supports querying of the "mined" information. The computational engine provides parallel execution of expensive image processing, object recognition, and query-by-content operations. Key benefits of the Diamond Eye architecture are: (1) the design promotes trial evaluation of advanced data mining and machine learning techniques by potential new users (all that is required is to point a web browser to the appropriate URL), (2) software infrastructure that is common across a range of science mining applications is factored out and reused, and (3) the system facilitates closer collaborations between algorithm developers and domain experts.

Keywords: image collections, mining, distributed systems, software, infrastructure, architecture, search, client-server, Java, database exploration

1. INTRODUCTION

NASA has been involved in remote exploration of the solar system for over forty years and, as a result, has accumulated a vast, geographically-distributed archive of images. Continued improvements in acquisition and storage technology are yielding new image sets with data volumes measured in terabytes (and beyond). Within these image collections there is a wealth of potential scientific information, but extracting the information by traditional means (manual labeling and analysis) is incomplete and often impossible due to the sheer volume of data. Maturing technologies in data mining, computer vision, and machine learning offer the potential to greatly enhance and, in many cases, enable new scientific explorations of large image sets. Early JPL successes in this direction include SKICAT [1], a supervised classification tool for cataloging stars and galaxies in the POSS-II sky survey, and JARtool [2], an adaptive recognition system that was applied to the problem of locating small volcanoes in the Magellan Venus dataset.

It may come as a surprise, but in both of these large-scale analysis systems, the core mining algorithms constitute only a small fraction of the overall software package. The bulk of the code serves to provide *infrastructure*. In broad terms, infrastructure provides the means for the user to manage and interact with data and algorithms. Examples of things we lump into infrastructure include:

- Methods to provide access to images stored in various formats and physical locations (memory, local disk, CD-ROM, local databases, remote URL's, remote databases, etc.).
- Tools that enable the user to easily browse and annotate images and other data sources such as image sequences.
- Facilities for selecting subsets of images, including selection based on "meta-data" constraints, e.g., finding images that satisfy certain latitude, longitude, and resolution constraints.
- Data management capabilities that provide persistent storage for the "mined" information and enable the user to query and visualize the results.

Send correspondence to Michael.C.Burl@jpl.nasa.gov.

- Job control that allows the user to monitor the progress of an operation, abort, or move long jobs to the background and receive notification when a job has completed.

Although this is by no means an exhaustive list, it should convey the message that infrastructure comprises a significant portion of any real, large-scale data mining and knowledge discovery system. Fortunately, much of the needed infrastructure is common across a range of applications.

The Diamond Eye architecture was conceived to factor out and reuse this common infrastructure and promote, especially within the planetary science community, the use of data mining techniques as a legitimate and enabling way to explore large image collections. In formulating the initial design, five system-level requirements were identified; specifically, the system should:

1. Enable users within the science community to access and use the latest image mining technology.
2. Encourage trial evaluation by potential new users, for example, by requiring only minimal hardware, software, and effort on the part of the user.
3. Support accumulation and querying of "mined" information.
4. Enable developers to easily add new algorithms and capabilities.
5. Factor out infrastructure that is common across data mining tasks.

In the resulting architecture, users (scientists) interact with one or more custom data mining servers through a cross-platform Java[*] applet interface. The applet allows the user to browse and annotate images, formulate queries, request data mining services, and display results. Each data mining server interprets user requests and provides access to the necessary (local or remote) images and data. In a typical deployment, the server is coupled with a high-performance computational engine such as a network of workstations to provide parallel execution of computationally intensive image processing, object recognition, and query-by-content operations. Separation of the user interface from the hard-core computation permits a user to work from a moderately equipped PC – a high-end client machine is not required. An object-oriented database associated with the server maintains information about the locations of image repositories, holds image meta-data (e.g., spatio-temporal relationships between images), and provides persistent storage for models and results. The database can also include raw image data, although a more common scenario is for the database to hold only generalized "pointers" that enable access to the raw image data.

In Section 2 we provide an in-depth view of the Diamond Eye architecture. Incidentally, the name Diamond Eye comes from the fact that one of the core capabilities the system is designed to support is the ability to find objects of interest, figurative "diamonds", in large datasets. In Section 3 we describe a current image mining application involving cratering studies on planetary and small body surfaces and show how the Diamond Eye system is being used to facilitate this study. In Section 4 we describe how algorithms are incorporated into the system. In Section 5 we place the system into context by discussing alternative approaches to image analysis and mining and outline how the Diamond Eye concept differs. In Section 6 we summarize and present directions for future work.

2. SYSTEM ARCHITECTURE

As diagrammed in Figure 1, the Diamond Eye system consists of the following components: a custom data mining server, a computational engine, an object-oriented database to manage data, models, and results, and one or more client applets that provide users with a graphical interface to the system. For maximum efficiency, the central components (server, computational engine, and database) should be co-located with the image repositories to be analyzed.

2.1. Client Applet

The client is implemented as a Java applet, which provides the user with a platform-independent[†], remote interface to the system. Since most computationally intensive tasks are performed on the server side, the user needs only minimal hardware, for example a PC or Mac with a network connection and a web browser. The applet is dynamically

[*]Java is an object-oriented language developed by Sun Microsystems for platform-independent, distributed computing in a heterogeneous networked environment.

[†]Platform independence simply means that the same code can be run on various hardware (Mac, PC, workstation) under a number of different operating systems (Solaris, Linux, Windows95, MacOS).

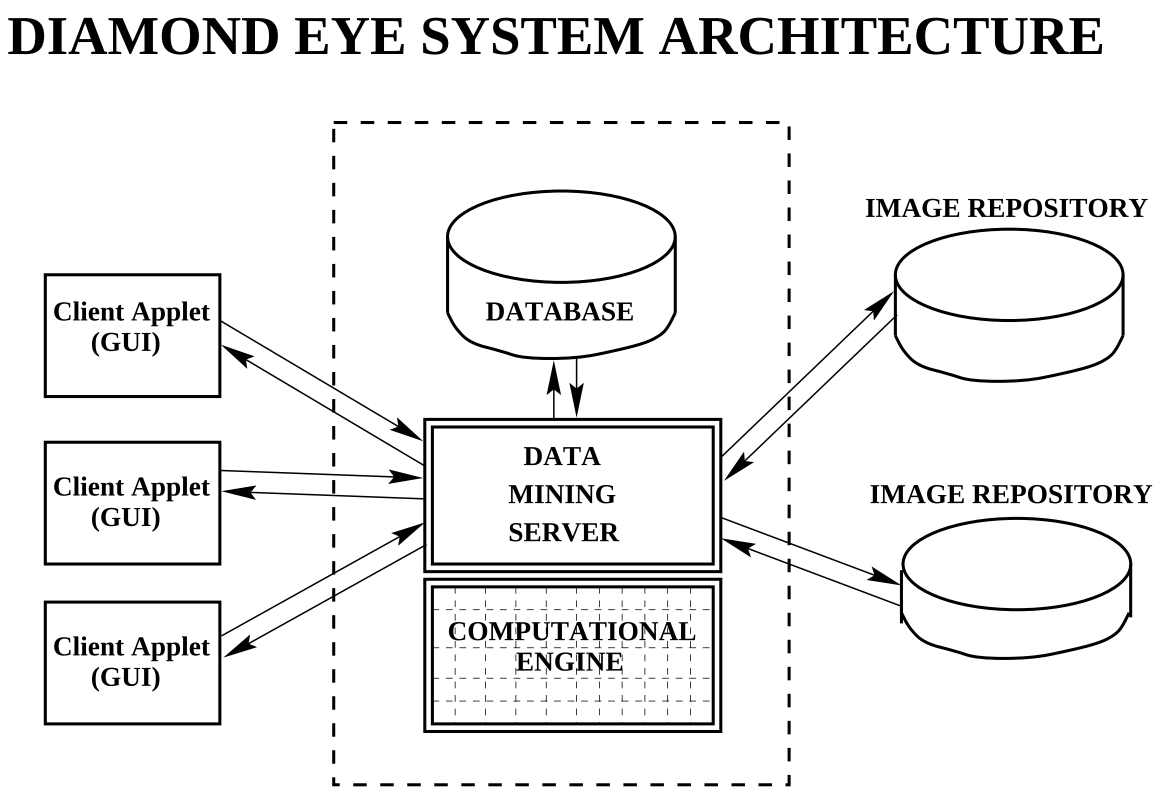


Figure 1. The basic Diamond Eye architecture consists of a custom server, which is closely coupled with an object-oriented database and a computational engine. Users interact with the server through an applet interface. The server executes client requests and handles data routing, such as retrieval of images from external databases.

downloaded each time the user connects to the system so there is no need for the user to deal with installing software, obtaining and applying bug patches, or replacing outdated software; the latest stable version of the code is available automatically. Thus, realization of the client program as a Java applet provides a significant maintenance dividend for both the user and software developers compared to traditional client-server arrangements.

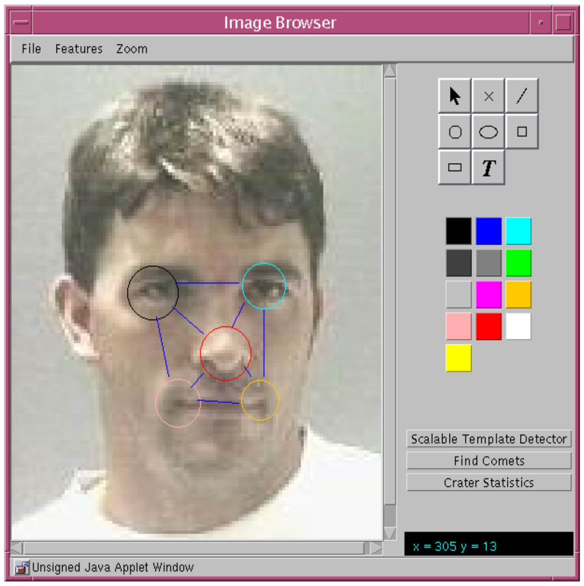
To fetch images from a database or to process data, the client sends a request to the server which operates remotely (e.g., at JPL). The applet on the user's machine maintains a connection to the server using Java Remote Method Invocation (RMI) [3]. RMI is a high-level package that enables communication between Java objects running on different computers without requiring the programmer to deal with the low-level details of forming socket connections, packing and unpacking objects into byte streams, or structuring messages according to a protocol. All of this basic communication happens transparently.

The client applet provides several "modes" of operation that provide distinct capabilities to the user. Modes that are currently implemented include: ImageBrowser, PlotBrowser, QueryBuilder, and ThumbnailBrowser. The top-level user interface for each of these modes is shown in Figure 2.

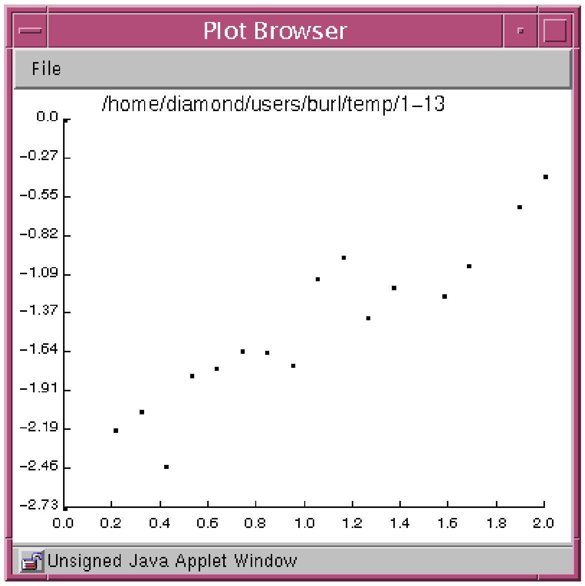
2.1.1. Image Browser

The ImageBrowser mode allows the user to load and display images. Images can be annotated by overlaying text labels and/or drawing geometric markers such as circles on the image. Since some mining algorithms must be trained, the annotation capability provides a way for the user to supply any needed ground-truth data. As an example, the JARtool algorithm developed in earlier work [2] can be adapted on-the-fly to find user-selected objects; the user must simply provide examples of the desired object by circling instances in a set of training images. The algorithm then constructs an appearance model that can be used to locate novel instances of the object in a new set of images. Manually supplied ground-truth annotations can also be used for quality assurance, e.g., cross-checking the performance of a labeling algorithm on a subset of images.

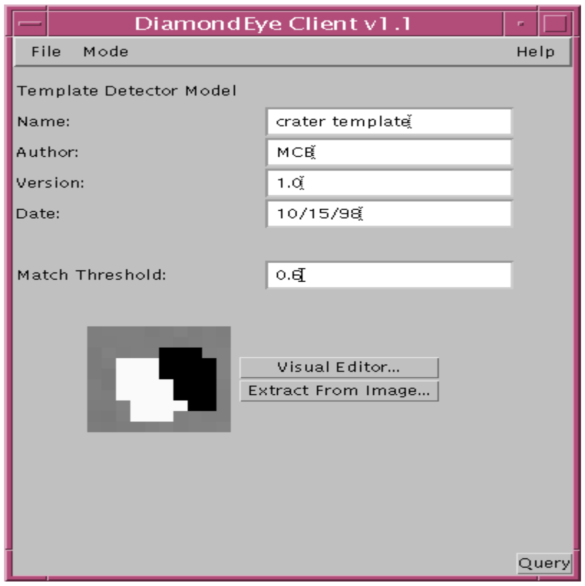
(a)



(b)



(c)



Template Detector Model

Name:	crater template
Author:	MCB
Version:	1.0
Date:	10/15/98
Match Threshold:	0.6

(d)

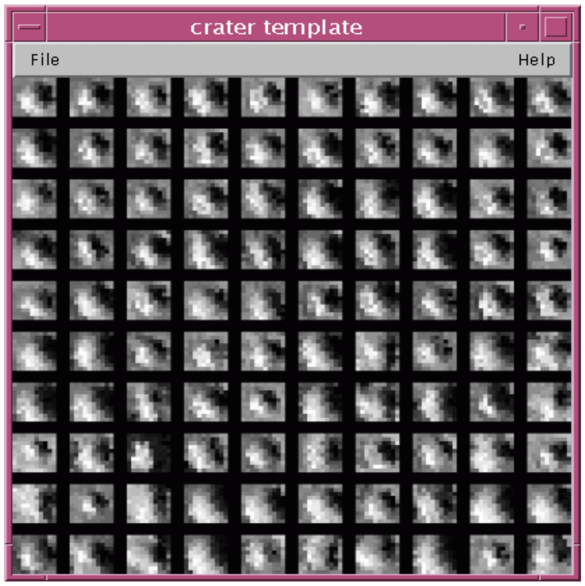


Figure 2. Four modes are currently supported in the client interface. (a) ImageBrowser enables the user to load, display, and annotate images, as well as execute algorithms on images and annotations. (b) PlotBrowser allows standard 2-D plotting. (c) QueryBuilder enables the user to construct visual query models of various types. The model shown in the figure is a template detector that can be used to find craters that look similar to the template. (d) ThumbnailBrowser is used to examine mosaics of thumbnail images returned in response to a query. For each thumbnail in the mosaic, there exists a *linkback* which makes it possible to go back to the specific image and region within the image from which the thumbnail was derived.

From the ImageBrowser mode, the user can execute algorithms that perform single-image operations. A primary goal in the system design is to provide an infrastructure that is reusable over a variety of applications; hence, special care was taken to insure that there is a uniform way to get user-supplied parameters and raw data (both images and annotations) from the ImageBrowser to the server-side algorithms. Conversely, the results produced by an algorithm must be returned from the server to the client, and the client must know how to handle the results. Since these issues affect both the client and server design, they are addressed in more detail in Section 4.

2.1.2. Plot Browser

The PlotBrowser mode allows the user to perform standard plotting functions. A PlotBrowser window can be invoked directly or from another mode. For example, there exists a utility for histogramming the sizes of displayed markers that is available from the ImageBrowser mode. Once the histogram has been computed, a PlotBrowser window is displayed to present the results to the user. More advanced capabilities for interacting with plots are planned, but not yet implemented.

2.1.3. Query Builder

In the QueryBuilder mode the user can interactively build a visual model of an object for which he would like to search. After a model is defined, it can be named and saved for later use. Visual query operations involve "applying" a model to a set of images and ordering the results according to some criteria, such as the degree of match between regions of the image and the model. Thus, the result of a query operation is a set of ordered "thumbnails" (small regions of an image or reduced size versions of whole images) that match the search criteria.

Several forms of query building are supported. Figure 2c shows one of the simpler forms. In this case, the user provides a template that is similar in appearance to the target object either by extracting a region from an image (an existing example) or by hand-painting on a canvas with the mouse and a palette of gray levels. The user can execute a search by clicking the query button in the lower right portion of the window. A dialog box pops up to allow the user to specify the subset of images over which the query should be conducted. The template is then applied to each image in the set using normalized cross-correlation to assess the similarity between regions of the image and the template. Locations where the correlation is high are selected as potential matches. The best matches over the set of images are presented to the user as in Figure 2d.

2.1.4. Thumbnail Browser

The ThumbnailBrowser allows the user to view and interact with mosaics of thumbnail images. Mosaics are typically generated as the result of a visual query performed by the user but may arise in other ways. Double-clicking on a thumbnail causes execution of a *linkback*. The linkback displays in an ImageBrowser window the specific image and region within the image from which the thumbnail was derived. Linkbacks are important because users often need to see an object, especially an unusual one, in the context of the whole image.

2.2. Server

The server is written as a Java application (program). Its main function is to provide user authentication, data access, and algorithm execution services. The server has explicit control over which resources (including algorithms) a user can access. This control is important since (1) the system is accessible from any external web connection and (2) it allows the user interface to be tailored for specific users and domains. For example, a user with interest in cratering would not want his interface to be cluttered with options that are specific to an unrelated domain such as astronomy.

After validating a user's login, the server listens for requests from the client. Multiple users/clients can be logged in to the server at one time. A common client request is for the server to provide data for visualization, e.g., from the ImageBrowser mode the user may want to open and display a specific file located on the server. To accomplish this goal, the client executes a method called `openFile`, which is a remote method defined for the Server object running on the server. A message identifying the method to be invoked and the arguments to be given to the method (e.g., the name of the file to be opened) is composed under the RMI interface and sent to the server. The Server object executes its `openFile` method and returns the (byte) data from the file to the client.

The server also brokers workspace needed for transient user data and algorithm execution. Algorithms are implemented as external programs or scripts with communication between the server and algorithms taking place through files. To run an algorithm and get the results back, the server and algorithm must share knowledge about the names of the files that will hold the algorithm input and output. The server determines what the filenames should be and communicates this information to the algorithm through command line arguments.

In the current implementation, we have used only a single server. However, for working with image data, it is desirable to have the algorithms that operate on the images be geographically "near" the data to avoid having to send large quantities of information across a network. For this reason, the Diamond Eye architecture will eventually support multiple servers with each server co-located with a large image repository. In this way, the computation is moved to the data rather than the other way around. Ultimately, servers will cooperate to meet user requests in a timely fashion.

2.3. Computational Engine

Since most image mining algorithms are computationally intensive, the Diamond Eye design includes a high-performance computational engine that is accessible by the server. In the current realization of the system, the engine is a network of Sun UltraSparc II workstations. Requests from the client for mining services are passed to the server; the server then executes the requested algorithms in parallel on the computational engine and returns the results to the client. An assumption underlying the system design is that the decrease in computational time provided by high-performance hardware will outweigh the combined costs of communicating the desired operation from the client to the computational engine and getting the results back to the client.

A number of image mining algorithms can be easily parallelized using the Message Passing Interface (MPI) [4] and a task-queue paradigm. One processor maintains a list of images to be processed (a task queue). Each of the other processors grabs an image from the queue and applies the desired algorithm to the image. After a processor has analyzed its initial image, it grabs and processes the next available image, and so on until the queue is depleted. A natural load balancing occurs between the processors since faster (or lightly-loaded) machines end up handling more images. This type of parallelization can be used for many image mining algorithms and is highly effective when the number of images to be processed exceeds the number of processors; however, it does not provide a speed-up when processing a single image. For single-image speed-up, a data parallel approach can often be used. In this paradigm, an image is split into subimages and each subimage is handled by a different processor. After each subimage has been analyzed, the results are merged.

2.4. Database

For security reasons an applet running on the user's machine has no access to the user's file system. Any information that should persist from one session to the next must be stored and retrieved by the server. In the Diamond Eye architecture, an object-oriented database provides persistent storage for the user (and for the server itself). A rich collection of information can be maintained in the database including the following:

- user profiles (logins, preferences, user data spaces)
- images and image meta-data (structure and relationships in the image domain)
- image annotations and content information (generated by the user or an algorithm)
- query and recognition models
- algorithms

Although the server's file system could provide data storage, use of a database enables efficient querying and reasoning about the mined information, as well as automatic maintenance of indexes and data relationship constraints.

Object-oriented databases provide significant advantages over traditional relational and object-relational databases. An object-oriented database uses the language's native structures – in our case Java objects – for its storage format and can therefore support an applications program interface (API) that offers transparent storage and retrieval. The "impedance mismatch" that often accompanies software development using relational and object-relational databases is eliminated. This frees the developer to focus on designing appropriate classes and methods, rather than worrying about mappings between the program's representation and the database's representation. Another advantage is that database queries are efficiently and naturally handled, even when the structure of the data is complex.

An important role of the database is to organize the meta-data associated with a given image collection and enable the identification of subsets of images based on meta-data queries. Meta-data refers to any external information *about* an image such as the planet from which the data was collected, latitude and longitude on the surface, sensing modality (SAR, optical), time of collection, resolution, etc. As an example, a user may want to execute a query to determine all images of a particular region on Mars that meet a given resolution constraint. The Diamond Eye database includes a persistent IMAGE class, which is able to hold image data and meta-data. This class can be subclassed to add domain specific elements as required. Relationships between images are supported by maintaining image collections (pointers to other images) within each IMAGE object. These relationships enable an image to contain or be contained in other images. This essentially allows arbitrary graph structures to be formed as needed to represent relationships between images in a domain collection.

Image annotations are maintained in the database via a MARKER class. A marker is simply an annotation of an image which identifies and localizes things deemed interesting or relevant. Subclasses of the MARKER class include BOUNDEDMARKERS such as circles, rectangles, polygons, etc., and SIMPLEMARKERS such as points, text, and lines. A marker can be created by a user who identifies an object of interest, such as a geologic formation, or by a recognition algorithm that attempts to identify objects or content in an image. The database maintains the relationship between markers and the images which contain them. Thumbnail images and their *linkbacks* will eventually be implemented using this facility. (In the current implementation, an ad hoc file-based technique is used.)

A USER class exists that allows the database to keep track of user profiles, including logins, preferences, and the user data spaces. Each user is able to maintain private image sets, models, and results. An ENTITY class allows users to name a real-world object present in an image collection and group the markers that represent this object. A LABELING class represents an event, in particular the creation of a set of markers by a labeler (human or algorithm).

At this stage of development, the IMAGE, MARKER, USER, ENTITY, and LABELING classes have been designed, and proof of concept demonstrations have been generated for some database capabilities. Database class design is continuing. Future versions of Diamond Eye will support storage and retrieval of queries, recognition models, and image mining algorithms. An important consideration that remains to be addressed is the design of a client mode that will enable both novice and experienced users to navigate the database, perform complex queries, and reason about any mined information. For example, the user may want to know if there is a systematic difference between the crater sizes that are estimated by an algorithm and the sizes estimated by human labelers. We believe that an object-oriented database with well-designed classes will provide the foundation for answering such questions.

3. IMAGE MINING EXAMPLE

A large number of images of planets and other bodies within the solar system (e.g., moons and asteroids) have been gathered by NASA spacecraft. Craters, which are one of the most prevalent geological features in the data, provide planetary scientists with important clues about surface age, collisional history, and subsurface structure. As an illustrative example of how the Diamond Eye system can be used in a real application, suppose that a scientist is interested in knowing the size-frequency distribution of craters in a particular region of Mars. Note: this is in part a hypothetical discussion because some of the capabilities described, especially those related to the database, have not been fully implemented in the current version of the system (DE 1.2).

With the Diamond Eye approach, the scientist begins by pointing his web browser to the appropriate URL to download the client applet. Through the applet interface, the scientist can formulate a query to the database to determine the names of all images of Mars satisfying certain latitude, longitude, and resolution constraints. The server receives the query request through the RMI interface and passes the query on to the database. Assuming the relevant meta-data has been imported, the database will determine the appropriate image names (more generally a set of pointers) and return a list of these to the server. If desired, the scientist can quickly construct a mosaic of image thumbnails and scan through, for example, to exclude certain types of terrain from the analysis. The scientist can also refine the query by restricting to images obtained by the Mars Global Surveyor mission (excluding earlier images from the Viking missions). The scientist may then issue a request to the system to apply the crater-finding algorithm to the set of images and send notification when the analysis is complete.

After the server receives this request, it must fetch the images to be analyzed. (For simplicity, the server pre-fetches all of the images; however, more sophisticated schemes are possible.) The fetching program must be sophisticated enough to obtain images from the server's database, as well as from the web and other locations, e.g., a remote relational database. The server then invokes an external algorithm `find_craters` which processes the images in parallel on a network of workstations. The results of the processing are recorded (temporarily) in the database. When all images have been processed, the server notifies the client that the results are available. The scientist may then want to load full-size versions of some of the images and observe the detected crater positions to insure that the algorithm has performed as expected. The scientist may also want to compare manually labeled craters with algorithm-labeled craters. A variety of tools can support this comparison, including confusion matrices, blinking between markers, and thumbnail mosaics that show the "contentious" cases. The scientist can review the results and correct any errors made by the algorithm. When the labeling is satisfactory, he can request that a size-frequency plot be generated. The labeling results (and plot data) can be committed to permanent storage within the database.

The scenario above applies to a certain "life stage" in the knowledge discovery process, namely once a satisfactory algorithm exists for the domain and the work to be done merely involves applying the algorithm to a set of images and keeping track of the results. The Diamond Eye architecture is also beneficial much earlier in the life cycle, when collaboration and exchange between the domain experts (planetary scientists) and algorithm developers is essential for addressing the problem and iterating toward a solution.

In the early stages of development, the scientist can use Diamond Eye to "show" the algorithm developers the type of images that are of interest and, more importantly, to identify the target objects, e.g., by manually labeling images. The algorithm developer can then define a preliminary algorithm, generate results, and show these to the scientist. It may be desirable to more closely examine objects that generate false alarms and discuss these with the scientists. The scientist can then offer insight that can be used to improve the algorithms. After several iterations, more rigorous evaluation of system performance will be warranted. Comparisons with human experts can easily be conducted to quantify how close the algorithm is to production-readiness. Although some of this exchange could happen in other ways, e.g., by posting images to a web site, the Diamond Eye system offers a higher level of interaction and exploration.

4. EXTENSIBILITY

In this section we describe in greater detail how algorithms are handled within the Diamond Eye framework. An "algorithm" consists of two parts: an executable file residing on the server that performs the actual processing and a Java "algorithm object" that wraps the executable and enables it to be treated by the client and server in a uniform way. All algorithm objects are derived from an abstract ALGORITHM class, where a class is essentially a set of data structures and a set of methods for operating on those structures. To say that a class is *abstract* means that it has methods which are defined, but not implemented. An abstract class serves to factor out commonality, but it cannot be used directly. Instead, a subclass must be defined in which the programmer provides a concrete implementation for the abstract methods.

One of the abstract methods in the ALGORITHM class is `displayDialog`. After the user has clicked a button to run a particular algorithm, the `displayDialog` method is invoked on the client. This method provides the algorithm a chance to obtain any additional input (parameters) required from the user. How the parameters are obtained is up to the designer, who can use something as simple as a text entry box or something as complicated as a rotatable globe on which the user can click and drag to specify a latitude and longitude range. The concrete implementation is left to the designer. (Note: if no additional information is required from the user, the `displayDialog` method will just be a `return` statement.)

The parameters from `displayDialog` can be used by the server in several ways depending on the specific algorithm and the way the algorithm expects to receive information. The most common approach is for the server to embed the parameters as command line arguments into a string that will be executed by the server's operating system. The algorithm designer must implement a method called `getCommandLine` that tells the server exactly how to perform this embedding. A second approach is for the server to export some or all of the parameters to one or more temporary files that will be read by the algorithm. The *names* of the parameter files are then passed to the algorithm through command line arguments. The algorithm designer must tell the server how many temporary files are needed and, of course, how the parameters should be structured within the files.

The server may need to export other information (beyond user parameters) for the use of the algorithm. For example, if the user has drawn circles on an image and wants a histogram of the circle sizes, it will be necessary for the server to save the circles to a file that the histogramming algorithm can access.

The actual algorithm is invoked using the `exec` method from the JAVA.LANG.RUNTIME class. When the algorithm completes (signaled by the `waitFor` method), the server obtains the algorithm results by reading in the appropriate files. The `importFiles` method (implemented by the algorithm developer) accomplishes this task. The results are then returned to the client.

We have found that this framework for adding algorithms to the system is both flexible and straightforward. The primary drawback is the overhead of communicating through files. To improve on this aspect, we are investigating the use of Java native method calls through which C *functions* are called directly from Java so that communication happens through memory.

5. ALTERNATIVE APPROACHES

It may help the reader place the Diamond Eye architecture in context by differentiating it from a number of alternative approaches for analyzing image data.

Visualization and Image Browsing Programs: Programs such as `xv`, `NASAview`, and even `Netscape` enable a user to look at images; however, they do not provide the additional functionality needed to perform image mining operations.

Remote login with X-windows: Using capabilities available under X-windows, a user can remotely login to another machine and run an image mining application program with the display set back to his own machine. This method is slow and does not provide much interactivity for the client. Part of the problem is that the server bears the entire load including the user interface. Even a small movement of the mouse requires the client machine to consult with the X-server. Also, unlike Java the X-windows system is not well-supported on all platforms.

Traditional client-server: In a traditional client-server arrangement, the client is an installed application. Communication is conducted via a custom client-server protocol over sockets. Earlier we noted that a client *applet* which is downloaded with each session leads to easier code maintenance and removes from the user the burden of downloading, installing, and updating software. The Java/RMI approach also provides a higher-level communication protocol, which simplifies the programmer's job of developing and extending code.

Stand-alone mining applications: The earlier JARtool and SKICAT systems were stand-alone applications that were developed for specific platforms (Sun workstations using the SunOS or Solaris operating systems). In contrast, the Diamond Eye approach is platform independent (at least from the user perspective).

A problem with the stand-alone systems is that each user group must duplicate resources such as a database and computational engine or make due without these resources. The Diamond Eye architecture relies on resource sharing to provide high performance at relatively low cost. Users can work from a modest platform and still obtain the benefits of a high-end system.

Stand-alone systems also present a significant barrier to the potential new user who typically does not have the resources or incentive to download, compile, and install a large, complex software package for the purpose of trying it out. In contrast, opening a URL on the web to access the Diamond Eye server is relatively simple and does not require significant effort from the prospective user.

Web-based visualization and analysis tools: A number of web-based tools for data analysis are under development, e.g., WebWinds [5,6], new versions of SAOimage [7], etc. Although some objectives are similar, especially in the client interface, these tools are designed only to support standard data analysis and visualization, not large-scale mining. These tools often tend to be "stateless" in the sense that there is no database behind the tool that provides a mechanism for accumulating knowledge and reasoning about mined information.

Distributed Agents: The idea of performing tasks using a distributed set of specialized programs known as *agents* has received considerable attention in recent years. Some agent-based systems, such as the InfoSleuth [8] system being developed through a consortium arrangement at MCC, have objectives that are similar to those of Diamond Eye; the difference is in the details. For example, InfoSleuth requires that meta-data knowledge be mapped into a large, common vocabulary (semantic ontology). In contrast, Diamond Eye will use an object-oriented database to organize and reason about meta-data.

Image Understanding Environment The Image Understanding Environment [9] (IUE), which was developed under a DARPA program, is also promoted as "software infrastructure", however, the infrastructure in this case supports the algorithm developer. In particular, there is a large set of C++ classes that provide constructs useful for image processing, e.g., a `linked_set_of_edgels`. The IUE may offer value on the algorithm development side, but does not provide the data access, persistent storage, database capabilities, or client interface that are needed for large mining applications.

Content-based Retrieval Systems: Within the computer vision research community, there are a number of experimental systems designed to access image content. Some of the better known systems include QBIC [10] (IBM Almaden Research Center), Photobook [11] (MIT Media Lab), and the Visual Image Retrieval Engine [12] (Virage). The focus in each of these systems is on a particular algorithm. Since the Diamond Eye architecture is designed to support a range of image mining applications, it is "algorithm-neutral". A number of different algorithms can be incorporated into the system and offered to users, including algorithms for content-based retrieval.

6. SUMMARY AND FUTURE WORK

Diamond Eye is a distributed software architecture that enables users (scientists) to analyze large image collections by interacting with one or more custom data mining servers via a Java applet interface. The Diamond Eye system enables users to easily access the latest image mining technology and provides infrastructure for a range of data mining tasks. The system has been used successfully to demonstrate preliminary crater counting and comet segmentation algorithms to our planetary science collaborators. Ongoing work, funded in part by a grant from the NASA Space Science Applied Information Systems Research Program, will attempt to develop the system into a premier resource for the use of the planetary science community.

ACKNOWLEDGMENTS

The research described in this article has been carried out in part by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. One of the authors (C.F.) participated in this project under the umbrella of the Summer Undergraduate Research Fellowship (SURF) program administered through the Office of Student-Faculty Programs at the California Institute of Technology. The authors thank Will Colwell for testing early prototypes of the Diamond Eye system.

REFERENCES

1. U. Fayyad, N. Weir, and S. Djorgovski, "SKICAT: A machine learning system for the automated cataloging of large-scale sky surveys," in *Tenth International Conference on Machine Learning*, pp. 112–119, Morgan Kaufmann, 1993.
2. M. Burl, L. Asker, P. Smyth, U. Fayyad, P. Perona, L. Crumpler, and J. Aubele, "Learning to recognize volcanoes on Venus," *Machine Learning* **30**, pp. 165–194, 1998.
3. "Java remote method invocation – distributed computing for Java." Sun Microsystems white paper. http://java.sun.com:8081/marketing/collateral/javarmi.html.
4. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, MA, 1994.
5. A. Jacobson, A. Berkin, and M. Orton, "LinkWinds: Interactive scientific data analysis and visualization," *Communications of the ACM* **37**, pp. 42–52, April 1994.
6. "WebWinds." URL. http://webwinds.jpl.nasa.gov/.
7. "SAOimage: The Next Generation." URL. http://hea-www.harvard.edu/RD/saotng.html.
8. R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, "InfoSleuth: Semantic integration of information in open and dynamic environments," in *ACM SIGMOD International Conference on the Management of Data*, 1997.
9. J. Mundy, "The Image Understanding Environment program," *IEEE Expert: Intelligent Systems and their Applications* **10**, pp. 64–73, Dec 1995.
10. M. Flickner, H. Sawhney, W. Niblack, *et al.*, "Query by image and video content – the QBIC system," *Computer* **28**(9), pp. 23–32, 1995.
11. A. Pentland, R. Picard, and S. Sclaroff, "Photobook – content-based manipulation of image databases," *Int. Journal of Computer Vision* **18**, pp. 233–254, Jun 1996.
12. "Visual Image Retrieval Engine." Virage home page. URL. http://206.169.1.82/market/vir.html.