1998 International Computer Symposium
Workshop on Image Processing and Character Recognition
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

# Towards Global Computing for Image Processing using Java

*Chungnan Lee, Shibang Yai, MinFong Horng and Chuanwen Chiang*

Institute of Computer and Information Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, ROC
Email:cnlee@mail.nsysu.edu.tw, cwen@impala.cie.nsysu.edu.tw

## Abstract

We discuss the issues of global computing and design a prototype system for group collaboration and distributed image processing in this paper. The proposed system provides an infrastructure to allow users to use the system resources, to upload image processing Java classes to our system, and to do volunteer computation. In order to support various input, output formats of image processing, the general I/O format is designed and is wrapped into objects. To reuse the existing parallel computing software, we also incorporate the Java applet-based computing with Java enabled PVM software to provide efficient reliable computing resources.

**Keyword:** Global computing, Java, Distributed image processing, PVM

## 1. Introduction

In addition to provide a vast of information, the Web can potentially provide a gigantic distributed computing power to meet the requirements of many scientific applications. The topics of distributed computing and group collaboration in Java [1] also become increasingly popular in the last few years, due to its cross platform characteristic. Examples of such systems are distributed applet-based massively parallel processor (DAMPP) [2], Javelin [3], IceT [4], Bayanihan [5], Tango [6], PageSpace [7], etc. The DAMPP system collects massive computing power by shipping a work applet to the client that hits the WWW server. The potential power for such system is unlimited. But it only asks service from the connected clients, purely relies on others machine power, and lacks ability to provide computing service. The Javelin system is also a Java-based distributed computing environment. It is similar to DAMPP in collecting computing power, but uses a broker to allow clients to request computing resources by submitting an applet or to provide computing power when it is idle. The IceT system, based on the traditional distributed computing paradigms, provides PVM-like distributed computing environment with the Java language. It incorporates a chatting tool into the system to allow the collaborative ability. Bayanihan discusses the idea of Web-based *volunteer computing*, which allows users to cooperate in solving a large parallel problem using standard Web browsers to volunteer their computers' processing power. In this way, the more powerful machine is, the more jobs it can finish. Also, if any work is left undone by a slow machine, or the machine fails, it eventually reassigns to other machine. The Tango system combines a Java-based collaborative environment with an executive providing general message filters and an event driven simulator together. Its applications include health care, education, and scientific visualization. PageSpace is targeted at supporting networked applications that require interaction among distributed software components and active processing. It introduces a notion of active Web pages that are capable of executing code.

Though the development of network computing in Java is tremendously increasing, there are still many questions to be overcome when using Java for science computation. For example, most systems are developed for some simple applications; what is the performance of Java? Can Java incorporate with the existing parallel computing software for the sake of reducing development cost? In the advent of Web and Java, in order to make "write once and run everywhere" and "the ideal of global virtual computing machine" come true, we discuss the issues of global computing and the design of a prototype system for global collaborative-based distributed image processing on the Web. The proposed system provides an infrastructure to allow users to upload image processing Java classes, to do volunteer computation, to use system resources, to interact and share information with each other through the collaboration tool. In contrast to Javelin, which needs both the upload applet and the Web page embedded in the applet to guide the clients to the uploading Web pages, our system allows the client only to upload Java classes and the server can dynamically distribute the Java classes to the clients. To reuse the existing parallel computing software we also integrate the Java applet-based computing with Java enabled PVM [8] program to enhance the performance of the system.

The remainder of this paper is organized as follows. In Section 2 we present the system design concept. Section 3 gives the system overview. Section 4 presents an example of hybrid computing. Conclusion is given in Section 5.

## 2. System Design Concepts

In this section we describe the basic design concepts, such as image computing services, general image format, volunteer computation, faults tolerance and security. We also discuss how to populate image processing Java classes, how to distribute components and tasks, and how to wrap objects.

## 2.1. Image Computing Services

One of the main goals of the infrastructure is to provide computing services for clients. In the proposed system, clients can connect to the Web server and request service by uploading image data and the related information to the server, then the server distributes the task and the related work classes to clients in the computing pool. Figure 1 shows the process of computing service. The infrastructure accepts the service request from clients and put the requests into the task distributor. For each client the server spawns a thread to handle its request. The task distributor distributes tasks of different clients to available work hosts and informs the data manager to collect the results. Once the computing is done and results are completely collected, the data manager sends the results back the clients through the socket connection. The parallel Java computing represents for the resources that are composed of PCs and workstations and connected by Java socket programs.

In addition to the computing service, the system can also provide the image processing applet to allow the clients to run the image processing on the local machine. This function is feasible using the new technology of Java JDK1.1.6 version and HotJava browser, which allows users to release their privilege of file access. Then the applet can read the data from users' local file system.

## 2.2. Volunteer Computation

When users are browsing our web page, they can become volunteer computing resources. Once other users make requests, our system will distribute these computing Classes of Java not only to our local computing resources, but also to those volunteer users' computing resources. In this system, we also use dynamic load balancing strategy to speed up the performance. By this method, we can augment the system's computing resources and capacities, and let the network users share their resources with each other. Figure 2 illustrates the concepts.

## 2.3. General Image Format

In order to support as many types of image processing functions as possible. We define a general data format for image processing I/O as illustrated in Figure 3. The template is allowed for the neighborhood operation or convolution, etc. The information for the template includes its height and width and the data type can be integer, float or Boolean. The general image format supports users' defined data format, which is processed as strings in the infrastructure and stored as files. Then the system just prints out the strings of the descriptions without further processing. The users need to interpret the data by themselves and must be responsible for the error handling for their own uploaded classes, the system just pipelines the error strings to the display.

## 2.4. Populating Image Processing Java Class

As described above, the infrastructure allows users to contribute his/her own Java version of image processing classes. So far, the image processing function in the system is quite primitive. However, as Java gains popularity, we can expect more and more image processing algorithms to be implemented in Java. The purposes of the proposed format are to allow users to conveniently use the image processing functions and to provide an easy way for users to prepare their image processing class that will be uploaded to the infrastructure. It also allows the infrastructure to invoke the uploaded components (Java Class) dynamically.

## 2.5. Wrapped Objects

Based on the general data format defined above, we can wrap these formats into classes as illustrated in Figure 4. The Java classes uploaded to the infrastructure must inherit these classes. First, the user imports the class of java.util.vector, Vector is a dynamic array, which can grow or shrink as needed, so we use it to store image data in the form of integer, float, or boolean in the vector. Three basic data structures *imageData*, *templateData* and *featureData* are defined. The "imageData" describes the I/O of image data and related information. The "templateData" contains the neighborhood operator and related information. The "featureData" contains the users' defined I/O features of image processing. After inheriting the classes defined above, the user can use the protocol as defined in Figure 5 to implement his/her image processing function in class. In the protocol, we provide several methods including *processing*, *getResultImage*, *getResultFeature* and *getErrorMSG* methods. The method of "*processing*" has three inputs that compose of "imageData", "templateData" and "featureData". The users' Java code for image processing is inserted in the the place of "processing" function. The slave programs can get results by invoking these methods of *getResultImage*, *getResultFeature* and *getErrorMSG*.

## 2.6. Distributing Components and Tasks

In contrast to other systems as mentioned above, the infrastructure uses a central control mechanism to manage all clients' uploaded classes. The client only needs to upload the class following the protocol provided above. This gives the server greater flexibility in dispatching the class and scheduling the workload. The server can dynamically distribute the Java classes to the computing hosts. The mechanism can be fulfilled through the Java Class's method "forName" and Java Object's method "newInstance" to invoke the components of image processing. A sample program is listed in Figure 6. It illustrates how a slave in the client sides to retrieve the required applet. It also provides Java Class's method "getMethod" to call these methods of *processing, getResultImage, getResultFeature and getErrorMSG.* Furthermore, the central control mechanism allows the system to keep tracking all the processes and maintain the consistency of returned data.

### 2.7. Load Balancing and Image Partitioning

The global computing infrastructure consists of local computing pool and the Web computing hosts. The further provides a reliable computing resource, the latter potentially provides a massive computing resource. Generally, the infrastructure consists of a cluster of heterogeneous workstations. In practice, even the configurations of the workstations are the same, their loads may not be the same. Hence, it is important to distribute the workload dynamically according to the ability of workstations, so that the computational efficiency can be improved.

We utilize a dynamic load-balancing strategy to prevent machines from idleness. At first, we partition a job into many tasks and distribute tasks to every host. For host who has finished one task already, will get another new task from the server. The remained tasks are distributed by the same way. Of course, to further improve the efficiency by reducing the communication overhead, the system can dispatch a number of tasks to the hosts based on the actual load behavior of the workstation, which is estimated from the execution time of the previous task.

The partitioning of images is dependent on the numbers of image data and the number of computing hosts. Let *Ni* be the number of image data and *Nw* be the number of computing machines. If *Ni* is larger than *Nw*, our system will not partition these images. The system uses each image frame as a task unit and the load distributing method mentioned in the above paragraph is used to dispatch the tasks. When *Ni* is less than *Nw*, the system will divide these images into sub-images. The number of sub-images is equal to the portion of integer of *machine_number/image_number*. Of course, the sub-images must have the overlap region, which is determined by the template data, or the users' provided information. If the number of sub-images is Nsi, and the constraint of *Nsi* is $2<=Nsi<=((image\_height/template\_height)/2)$. Then, the sub-image is used as one task unit and the distribution method is the same as previous method. In other words, the distribution of image processing should prevent from idle machines and make powerful machines responsible for more tasks during execution.

### 2.8. Faults Tolerance and Security

The faults may come from the computing errors, the host disconnections and the host failures. During the components (classes) are distributed to computing hosts, if computing errors, such as the data formats mismatched or exceptions. etc., happen, the system will use the *error_msg* mechanism in Figure 3 to pipeline the error messages generated by classes to the display screen. If the class executes correctly, the slaves get "NULL" message and return the results to the server. If some exceptions are produced, the system will get the string of error messages.

Due to the central control mechanism, once a computing host is disconnected from the computing pool and the results of the tasks cannot be returned, the server will reassign the tasks to other computing hosts. At the same time, the disconnected machines are set to the status of fault and are eliminated from the computing pool.

The security of server is an important issue and attracts great concern. Java itself is a language of strongly type and safety, and it has no pointer operation. Besides, the JVM (Java Virtual Machine) has the ability of garbage-collect memory to prevent memory leakage. In addition to Java internal security, we will test the uploaded classes in a sandbox before moved to the public area to avoid the destructive components slipping into our system such as programs that have infinite loop or always allocate new memory, etc.

## 3. System Overview

### 3.1. Background

Just like other applications that need huge of computation resources to process jobs, the parallelism of image processing has received great attention over the years. Due to the growing development of the Web technology, it is very essential to work out a proper Web-based working infrastructure that can support both distributed computing and group collaboration. Thus, we design the system to allow users to upload image processing Java classes by following a simple protocol, to allow the server to dynamically distribute the Java classes to the clients, and support collaborative work.

Generally, there are two ways in using Java for global computing: one is Java applet-based distributed computing; the other is Java enabled distributed computing. Java applet can be downloaded, when the Web page is requested. The potential of computing power could be huge, because machines connected to the Web server can be regarded as its processors in the virtual distributed computing environment. Theoretically, the distributed Java applet-based massively parallel processing can be possible, but it can never become a daily-use for the science computation for a couple of reasons. First, the network bandwidth can cause problems. For example, the image processing that needs to operate on a huge number of image data will cause the network traffic jam. Second, the computing power mainly relies on the connection of someone's machine, it may not be there when you need it. Third, Java applet is not allowed to use the I/O on the client machine that restricts the flexibility of using applets. Some other disadvantages of using Java applet-based distributed computing are that its' execution efficiency is still lower than other languages such as C/C++.

Furthermore, Java applet control platform is not as flexible as those existing parallel computing platforms, such as PVM.

Two mechanisms, integrated computing mechanism and hybrid computing mechanism, are developed to enhance the system performance and to avoid the disadvantages described above. We will describe these mechanisms hereafter.

### 3.2. Integrated Computing Mechanism

In this subsection, we describe the integrated mechanism to allow users to make requests of image processing, to offer components and to join into the computing pool. Figure 7 shows the flowchart of the integrated computing mechanisms. We describe the flowchart as follows:

i. The users can upload image data and related information to our server and choose a processing function. These requests are added to the "Request Queue" of the "Task Distributor". In the request queue, each processing function points to the related tasks.
ii. The resources manager must keep the status of each computing resources such as "busy", "idle" or "fault". Hence, the task distributor can be aware of how many computing resources are available, so that it can get the IP address of available machines from the resources manager.
iii. The task distributor asks the class manager about the classes of the users' requests and related information.
iv. Then the task distributor distributes the image data, related information and classes (components) to the computing resources as illustrated in Figure 6.
v. After the tasks are processed, the computing resources immediately return the results to the server. The data manager rearranges these results based on the users' updated requests.
vi. Finally, the data manager returns these results to the corresponding users or reports the related error messages.

### 3.3. Hybrid Computing Mechanism

In this subsection, we discuss how to integrate the Java distributed computing and PVM. This is an example of the integration of both Java computing and Java enabled PVM computing. The PVM's master program is written in C/C++. We first wrap the PVM's master into Java's libraries that can be called by the Java's native method. Then the system can call the image processing functions written in PVM. Assume that we have the same functions written in both Java and PVM programs. Then the system can utilize both of the computing resources. The mechanism of the hybrid computing is shown in Figure 8. Because the Java applet can't directly communicate with C/C++ even by the native method, we design tags for both PVM and Java distributed computing to keep trace of the progress of their tasks. If the progress of the tasks is overlapped such as the PVM's tag over the PJC's tag, or vice versa, then the system knows that all the tasks are done. Eventually, the system assembles the results and sends them back to the user.

### 3.4. User Interface

We briefly introduce the user interface of the proposed system in this subsection. Figure 9 shows the user interface of image processing services. The users can browse their images and fill related information. After choosing a processing function, the user can submit their requests to our system. If the processing function needs the template or other feature, the interface will pop up the "Browse Template" or "Browse Feature" options. Otherwise the user interface just shows the "Browse Images" option.

Figure 10 shows the user interface for a user to upload their Java classes. The users can browse their components file (Java classes) and fill in the related information such as the component name, image type, etc. If the users' applet needs other feature, they will upload the feature description file. After filling these options, the users can submit their processing components to our system.

When a user would like to contribute his computing resource to the computing pool, he can push the "Offer Computing Resource" button in Figure 11 to commit the volunteer computation.

### 3.5. Collaboration Support

To support the group collaboration functions, such as multi-users interactive discussion, design, information sharing, the system discussed above is embedded into a collaborative infrastructure [9]. The infrastructure provides collaborative tools to handle the local and remote inputs and outputs in a form of centralized model. The tools include chatting box, whiteboard, and video conferencing. The tool is developed in a modular way to facilitate future tools and improvement. The infrastructure provides flexible collaboration control methods to initiate and terminate collaborative sessions, to join or leave ongoing sessions, and to invite a new participant in a collaborative environment. Figure 12 shows the infrastructure that consists of clients, collaborative server, Web server, distributed computing platform, and applications. Each component is realized as the cooperation of some agents. A user first accesses the environment from any WWW browser. Then a Java byte-code encapsulating the interface in shared workspace agent is shipped to the client site. From the front-end interface, the user can input parameters for collaborative environment, set up peripheral devices, and activate the distributed image processing application, and submit Java classes and data. The user interface for the group collaboration is shown in Figure 13.

## 4. Hybrid Computing: A Corner Matching Example

In this section, we evaluate the performance of Java and

PVM distributed computing in the infrastructure. We use a corner matching which is used in image mosaic and implemented in both C version of PVM and Java applet. The execution of C version in PVM environment is invoked as a native method in Java. That means we wrap the C version of corner matching. Therefore we can integrate the computing resources of parallel Java computing and PVM.

Table 1 lists the execution time for corner matching written in both Java and C versions and running under different platforms. Using Netscape 4.02 running on Pentium 200MMX with 64MB RAM, the performance of Java is similar to the performance of C/C++ program running on Sun Ultra Sparc 1 without optimization. However, optimizing the performance of C/C++ program (using optimization option of C/C++, -O) is about 9.6 times faster than that of Java. Unfortunately, to run Java on Sun ultra Sparc 1 with either Netscape 3.03 or Netscape 4.0b3 which is no JIT (Just In Time) compiler is very inefficiency.

Table 2 lists the sending time of two 512 by 512 size image files and two corner points files and the execution time for different platforms. As one can see, the communication overhead becomes a problem for Java computing.

Table 3 lists the execution time on one PC, parallel Java computing, PVM, and hybrid distributed computing, respectively. We define the relative computing power of Pentium 133 with 32MB RAM as one and compare its execution time with the rest of machines. With four PCs, the speedup for parallel Java computing is 2.413. In the PVM environment, the image data server is mounted as a network file server by the machines in the cluster. Hence, the communication time can be neglected. Under the circumstances, the speedup for four Ultra Spac 1s on PVM is 5.117. With four PCs on parallel Java computing and four Ultra Spac 1s on PVM the speedup can be further improved but not that much.

## 5. Conclusions

We have discussed the issues to develop a global computing system for distributed image processing. We have presented a prototype, which allows users to upload the image data and related information to the server for image processing on WWW. The system also utilizes a load balancing strategy to extend the ability of computation in the global computing environment. We integrate the existing Java applet-based computing with Java enabled PVM software to provide more efficient reliable computing resources and to reduce the development cost. Experimental results show that the execution speed of Java applet is still slower than that of C program.

## 6. References

1 James Gosling, Bill Joy and Guy Steele, The Java Language Specification, Addison-Wesley, Reading, MA, 1996.

2 L. Vanhelsuwe: "Create your own supercomputer with Java", http://www.javaworld.com/javaworld/jw-01-1997/jw-01-dampp.html.

3 P. Cappello, B. Christiansen, M.Ionescu, M. O. Neary, K. Schauser, and D. Wu: "Javalin: Internet-Based parallel Computing Using Java", Concurrency:Practice and Experience, Vol. 9, No. 11,pp.1139-1160, Nov. 1997.

4 P. A. Gray Vaidy S. Sunderam: "IceT: Distributed Computing and Java" , Concurrency:Practice and Experience, Vol. 9, No. 11,pp.1161-1167, Nov. 1997.

5 L. F. G. Sarmenta: "Bayanihan: Web-Based Volunteer Computing Using Java", http://www.cag.lcs.mit.edu/bayanihan.

6 L. Beca, et al. "Tango – a Collaborative Environment for the World Wide Web," in http://trurl.npac.syr.edu/tango/papers/tangowp.html.

7 P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche, "Coordinating Multiagent Applications on the WWW: A Reference Architecture," IEEE Trans. Software Engineering, vol. 24, no. 5, pp. 362-375, May. 1998.

8 G. A. Geist and V. S. Sundream, "The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations," Proceeding of the Sixth Distributed Memory Computing Conference, IEEE, April 1991, pp. 258-261.

9 Tain-chi Lu, Chung-Wen Chiang, Chungnan Lee, and Tony-Yee Lee, "A Web-Based Distributed and Collaborative 3D Animation Environment," ACM 1997 Workshop on Java for High-Performance Network Computing, Las Vegas, 1997.

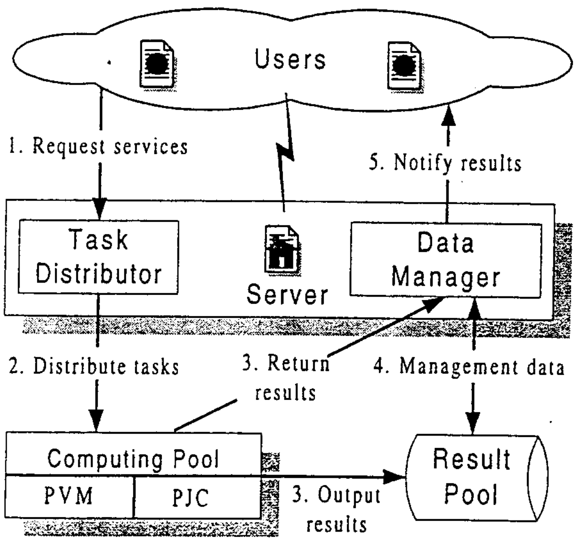


Figure 1. The computing service mechanisms used by the system. The PJC is parallel Java computing.

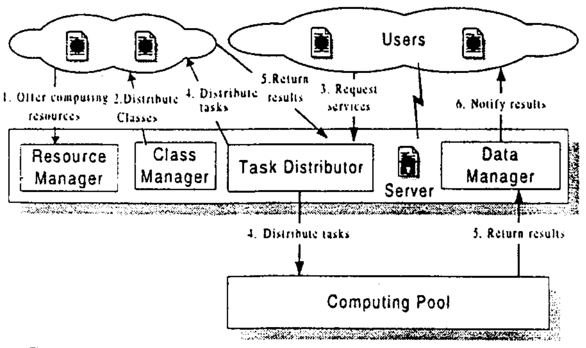


Figure 2. The flow diagram shows how the Web users volunteer to offer their computing resources and how the Web users request the computing service.

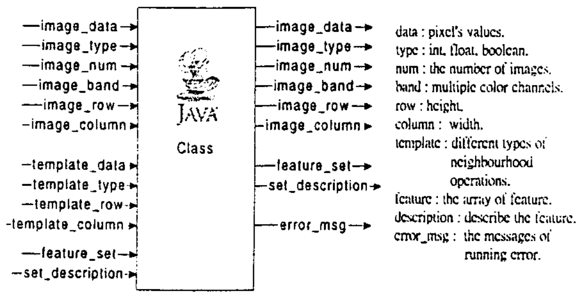


Figure 3. The general data format for the input and output of an uploaded classes.

```
import java.util.Vector;
class imageData
{
   // pixel's values
   public Vector image_data;
   // type: "int", "float", "boolean"
   public String image_type;
   // multiple color channels "1", "2", "3"
   public String image_band;
   public int image_row;      // height
   public int image_column;   // width
}
class templateData
{
   // different types of neighbourhood operations
   public Vector template_data;
   // type: "int", "float", "boolean"
   public String template_type;
   public int template_row;          // template height
   public int template_column;       // template width
}
class featureData
{
   public String feature_set;   // the string of feature
   public String set_description; // describe the feature
}
```

Figure 4. The I/O format of the component are wrapped into objects.

```
Class className
{
   imageData outImage = null;
   featureData outFeature = null;
   String errorMsg = null;
   public void processing(imageData image,templateData
      template , featureData feature)
   {
     //...........
     // add user's code for image processing application
     // here.
     //...........
   }
   public imageData getResultImage(){
     return outImage;
   }
   public featureData getResultFeature(){
     return outFeature;
   }
   public String getErrorMSG(){
     return errorMsg;
   }
}
```

Figure 5. The methods of components.

```
   :
String   strClassName;// user's class name
// input,output image
ImageData   inImageData,outImageData;
// input template
TemplateData   inTemplateData;
// input,output feature
FeatureData   inFeatureData,outFeatureData;
// error message
String   errorMsg;
   :
Class   dynamicClass = Class.forName(strClassName);
Object   objectClass = dynamicClass.newInstance();
Class[]   paramType = new Class[] {imageData.class,
   templateData.class, FeatureData.class };
// call processing function to process image
Method   mainMethod =
   dynamicClass.getMethod("processing", paramType);
Object[]   methodParam = new Object[]{inImageData,
   inTemplateData, inFeatureData};
MainMethod.invoke(objectClass, methodParam);
   :
```

Figure 6. Slaves (Computing resources) invoke components and their related methods.

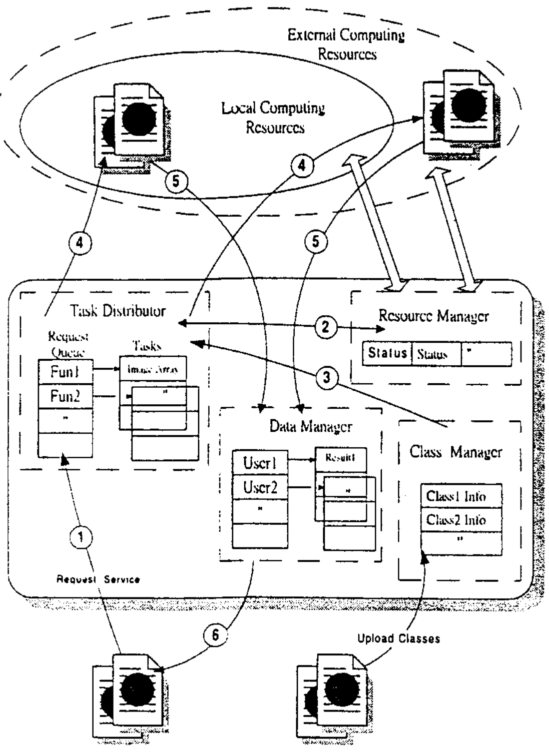


Figure 7. The flowchart of the integrated computing mechanism.

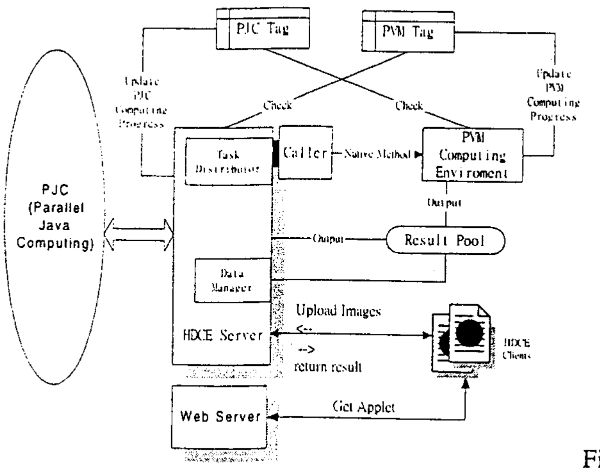


Figure 8. Hybrid computing by using the resources of PJC and PVM.

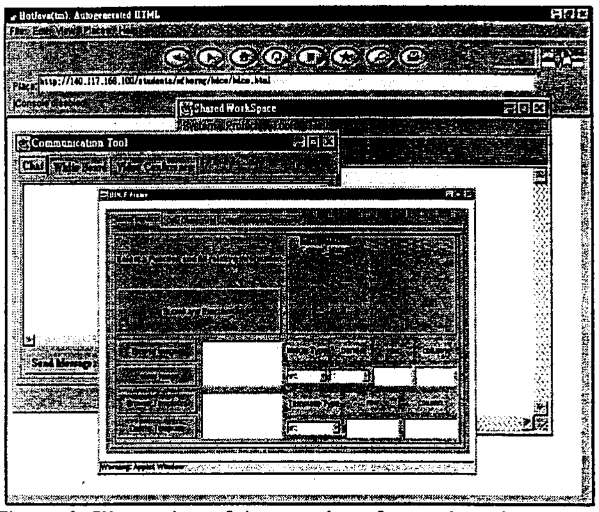
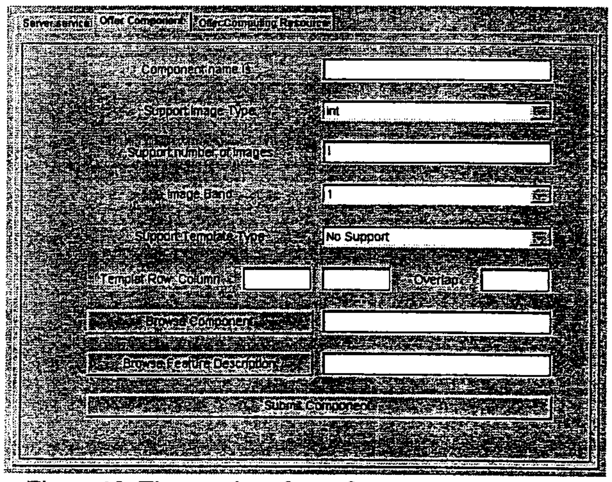


Figure 9. Illustration of the user interface, when the system provides the services to the Web client.



Component name is

Support Image Type: int

Support number of images: 1

Image Band: 1

Support Template Type: No Support

Templat Row, Column

Overlap

Figure 10. The user interface of component uploaded.

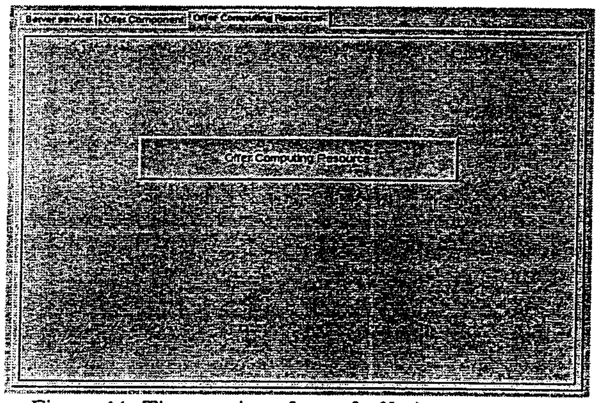


Figure 11. The user interface of offering computing resource.

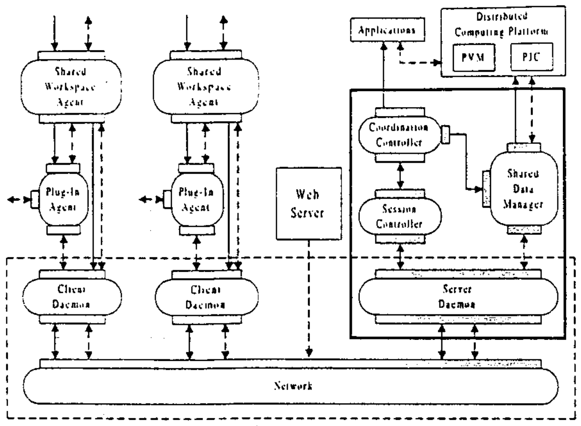


Figure 12. Block diagram of the collaborative infrastructure.

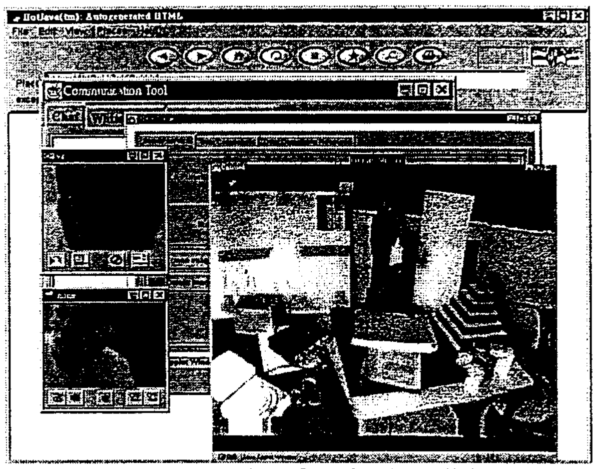


Figure 13. The user interface for the collaborative infrastructure.

Table 1. The execution time for corner matching module written in Java and C version and running under different platforms. The unit is second.

| Time / Machine | C version with optimization | C version without optimization | Java applet under Netscape 3.03 | Java applet under Netscape 4.0b3 | Java applet under Netscape 4.02 |
|---|---|---|---|---|---|
| Pentium 200 MMX 64MB RAM | 0.88 | 2.86 | 9.280 | Not available | 6.860 |
| Ultra Sparc 1 SunOS 5.51 | 0.713 | 5.2004 | 58.93 | 84.183 | Not available |

Table 2. The sending time of two 512 by 512 size image files and two corner points files and the execution time for Java computing on different platforms. The unit is in second.

| Machine / Time | Pentium 133, 32MB RAM | Pentium 120, 48MB RAM | Pentium 166, 32MB RAM | Pentium 200, 64MB RAM | Ultra Sparc 1 SunOS 5.51 Netscap 3.03 |
|---|---|---|---|---|---|
| Startup and sending time | 5.550 | 5.220 | 4.340 | 2.870 | 11.672 |
| Execution time | 10.270 | 11.100 | 10.220 | 6.860 | 58.939 |

Table 3. The execution time for matching 28 pairs of images on one PC, Java parallel computing, PVM, and our distributed computing platform. The unit is in second. The linear time indicates the idea situation. The parallel efficiency is the ratio of achievement and the unit is in percentage (%).

| Environment / Time | One Pentium 133 32MB RAM with relative computing power as one | Four PCs with relative computing power as 4.428 | PVM environment using four Ultra Sparc 1s with relative computing power as 6.693 | PVM and Java environment with four PCs and four Ultra Sparc 1s with relative computing power as 11.121 |
|---|---|---|---|---|
| Execution time | 291.56 | 120.824 | 56.983 | 42.775 |
| Linear time |  | 65.845 | 43.562 | 26.217 |
| Parallel efficiency |  | 54.50% | 76.45% | 61.29% |