# DESIGN AND IMPLEMENTATION OF A DISTRIBUTED REAL-TIME IMAGE PROCESSING SYSTEM

*D. M. Wu, L. Guan, G. Lau and D. Rahija*

Department of Electrical Engineering
The University of Sydney
Sydney NSW 2006 Australia

## ABSTRACT

New development in the design and implementation of a distributed, real-time image processing system is presented in this work. The system uses an IBM personal computer as the front end to a remote computer *via* the Internet. The standard TCP/IP networking protocols are utilised to link the IBM-PC and high performance remote devices such as transputer networks and super-computers. The access to the powerful remote computers enables the system to complete complex image processing tasks in real-time. During processing, the image is transferred to the remote machine and then transferred back to the PC for display. The system serves as a prototype for a full-feature image processing and analysis package, as well as a programming platform for the research and development of new image processing algorithms.

## 1. INTRODUCTION

The introduction of powerful microcomputers in recent years has made personal image processing systems affordable to the individual researchers. Over the years, microcomputers have significantly improved in the area of computational performance, storage and display. These features are therefore ideal as a platform for inexpensive image processing systems [1, 2]. Although these systems functionally satisfy the requirements of most general purpose image processing, real-time applications of such systems are limited by the computing capability of the microcomputers.

This paper presents the design and implementation of a distributed, real-time image processing system using an IBM-based personal computer (IBM-PC) with access to a powerful computer via Internet The advantage of using such a configuration is the sharing of a high performance central computer or a processor network, which is publically accessible, between a group of users. This provides each user with substantial boost in processing power with minimum cost.

The system serves as a prototype for a variety of full-feature image processing and analysis applications, as well as a programming platform for the research and development of new algorithms. Since the system is non-trivial, the architecture of the software must be well structured and clearly defined. This goal is achieved using object-oriented design methodologies. The paradigm enables a more efficient way of modelling real life situations and hence more robust systems can be built.

## 2. SYSTEM OVERVIEW

The system, *VisionLab*, is an image processing system to be operated on a IBM-PC with a Ethernet network expansion card. The PC is used to control the overall performance of the system, to manipulate graphical display and to perform a number of simple processing tasks. The system also provides capabilities in networking so that a remote computer system is controlled by the IBM-PC via the Internet. This enables more elaborate, computationally intensive algorithms such as image analysis, feature extraction and pattern recognition to be executed in real time. With the advances in today's networks, the overhead in data transfers is small compared to the benefits acquired by using a high performance computer. Furthermore, the cost of setting up such a system is significantly lowered when compared to the cost of purchasing a high performance computer dedicated to an image processing application.

VisionLab is a versatile image processing system. Useful features provided by VisionLab are summarised follows:

1. The system supports remote access to a high end computer via the Internet using the TCP/IP protocols. This ensures flexibility in choosing the remote high performance computer, and the execution of complex image processing tasks in real-time.
2. Interactive operations are provided so that commands can be selected once the image is opened.
3. The system allows multiple images to be concurrently processed in the background, with or without access to the remote computer. This allows the user to attend to other applications while a batch file is being processed. In case of network congestion, the user can locally work on another image while waiting for a response from the network.
4. The system has a batch mode so that a set of commands can be pre-programmed and executed without interaction with the user.

0-8186-7123-8/95 $4.00 © 1995 IEEE

5. A history list of all operations performed on the image is maintained. This enables the user to revert to any state of the image after a sequence of operations.

Apart from setting up a system which is useful in many practical situations, a primary goal is to develop a software toolkit for developing new image processing algorithms. Emphasis is placed on providing a standard interface between modules within the software architecture so that code segments devised by different people can be easily integrated into the system. The standard interface also provides abstraction to the underlying hardware and hence aids the representation and description of the data structures for the programmer.

## 3. PC SYSTEM DESIGN AND IMPLEMENTATION

VisionLab runs on most IBM PC computers or compatibles with a Intel 80386 or better processor, minimum of 8 MByte DRAM. Other peripherals include a high resolution SVGA monitor and a Ethernet network expansion card for LAN access. The network connection between the remote computer and the PC system is thus provided by the Ethernet network expansion card. The remote computer can be any computer which is connected to the Internet, running TCP/IP network software to allow users to remotely access and operate the computer [4, 5]. Image data are transferred to the remote computer for processing, and then transferred back to the PC for display.

The design of the software is divided into two sections: the local IBM-PC and the remote computer. Since many high-end machines are already connected to the network with installed networking software, most of the effort in the design is focussed on the local PC. One of the major hurdles associated with designing software for the IBM-PCs lies in the architecture of the Intel 80x86 microprocessor. The Intel microprocessor designers, in an attempt to provide compatibility with their older generation processors, somewhat restricts the development of their newer processors [6]. The compatibility requirement restricts the available address range to 1 MByte, and severely limits the full potential of the processor.

### 3.1. PC Operating System

As a direct consequence, one of the major problems with PCs is memory management. Programs can only directly access the 1 MByte base memory, known as conventional memory. The extra memory can only be accessed if the processor is switched into protected mode and thus enabling the extra address lines. This function is usually supplied by the operating system. However, the interface for this function varies between different systems. Consequently, an important issue in the design is to select an appropriate operating system which provides an easy method for this mode transition.

Other considerations for selecting an operating system for the development platform include support for graphical user interface and concurrent programming. There are a few operating systems available for the IBM-PC which satisfies these requirements such as OS2, Windows NT, XWindows, and MS-Windows. The MS-Windows operating system is chosen since it is the only system which provides most of the desired functions without large demands on resources such as disk space and memory. However, MS-Windows is not an independent system and requires MS-DOS for most of its underlying system functions. It therefore still exhibits problems with memory management as with traditional DOS systems. Fortunately, Windows automatically switch the processor into protected mode when it is invoked. As a result, it provides services to access extended memory up to 16 MByte which is plentiful for our applications. It also provides multitasking, although not in the preemptive form. However, a solution to overcome this has been devised with some extra low-level software.

The MS-Windows operating system not only solves the memory access problem, it also provides a user friendly system with all applications behaving in similar ways. Thus time required to learn how to use a Windows compliant application is reduced. The Windows environment also provides device independency. That is, programs can run in many hardware configurations without recompilation of code.

The Windows operating system also supports networking using the sockets paradigm introduced by the Berkeley Software Distribution of UNIX. This provides programs with an TCP/IP protocol interface and reduces the complexity of the software at the application level. On an abstract level, the programmer treats the interface as a port where information can be transferred and received. More importantly, this is also the common protocol employed by all computers connected to the Internet.

### 3.2. Object-oriented Design

One of the main objectives of the system is to design a programming platform for developing new algorithms. To achieve such a platform, the architecture of the software must be well structured and clearly defined. Modularisation of sub-components of the software is hence a main issue. This aids the researcher in understanding the overall system without being concerned with the low-level details such as hardware dependencies. As a result, some formal design approach is used to manage the overall system. For this system, the object-oriented design methodology is used.

The object-oriented design methodology has had considerable exposure in the recent years [7]. Indeed, it promises a more efficient and robust way of thinking and modelling real life situations. Object-oriented design is a technique that pushes to the extreme a design approach based on abstract data types. It has become popular in the past few years due to the appearance of new programming languages that support an adequate notation to map the design to the implementation. Another factor which contributed to its popularity is that the methodology raises the expectation for producing reusable software components [8,9].

In an object-oriented system, a direct correspondence between the real-world and objects is maintained so that they do not lose their integrity and identity. The technique allows software to be constructed out of objects that have a specified behaviour. Objects themselves can be built out of other objects. This can be extended infinity as long as the associations can be visualised conceptually.

The software design and implementations is divided into two sections. For the IBM-PC, object oriented design provides a solution for building such systems. Initially, the classes and their most fundamental relationships are found. A minimal set of operations are then specified for each class which in turn are refined by specifying their dependencies on other classes. Inheritance is introduced to maximise reusability of common sections where appropriate. The software design for the IBM-PC is greatly influenced by the Object Windows Library (OWL) provided by the Borland compiler tools [10]. OWL provides an interface between the application and the operating system and hence relieves the programmer of the low level details which are system dependent. The library is a powerful set of building blocks for constructing Windows applications with full-featured user interface. It uses the object- oriented paradigm to encapsulate the Windows Application Programming Interface (API), insulating the programmer from the details of Windows programming. Applications communicate with the operating system via messages.

As the IBM-PC serves as the front end to the distributed system, the software is significantly more complex compared with that of the remote computer. Therefore, emphasis is placed on the software architecture for the IBM-PC. The software system is schematically illustrated in Figure 1. At the centre of the system is the *Event Handler.* This is responsible for handling Windows system messages and dispatching them to the appropriate manager for further processing. The main object types of the system are the *Multi-Window Manager, Process Scheduler, Network Manager, History Manager* and the *Batch Job Manager.*

The Multi-Window Manager handles the responsibility of a Multi-Document Interface (MDI) which includes opening and displaying multiple images in the application's window. It also mediates the users' requests for operations on a particular image. The operation can be performed either locally, or carried out on the remote computer during a network session. A network manager asynchronously receives responses from the packet driver via the Sockets interface and controls the communication between the local device and the remote device. Operating asynchronous provides the flexibility for the user to attend to other tasks using the local PC when the remote device is busy.

The Process Scheduler is responsible for subdividing each processing task into slices so that multiple tasks can be executed concurrently. It uses a round-robin scheduling scheme with all processes running at the same priority. This imposes a fixed time on the amount of continuous processor time that may be used by a process. If a process exceeds the time limit, it is preempted from its processor and places at the end of the list of process waiting for the processor. This object basically handles the task of the kernel in an operating system. This is necessary because Microsoft Windows does not support preemptive multi-tasking.

The History Manager and Batch Manager are objects which enhances the usability of the system. The History Manager is responsible for saving an image to disk before an operation is performed on the image. This enables the user to recall the state of the image after any previous operations, which is extremely useful when comparing the effects of a particular set of operations. The Batch Manager allows the user to build custom functions from a set of basic commands provided by the system. As a result, more complex operations can be devised without recompiling the software. The Batch Manager mediates the batch text file between the Editor, Process Scheduler and Parser. The Text Editor provides the tools for editing the batch script while the Parser translates the text file into the internal representation for execution.

There are three main data structures in the system which included *the Image, Batch Script* and *History Stack objects.* The following paragraphs describe each of these data structures in detail.

The Image object contains the display and internal representations of the image. All operations which may be performed on the image are specified by the methods of the object. This prevents the data from being mis-used. The display representation conforms to the Device Independent Bitmap (DIB) structure defined by the operating system. A DIB consists of two distinct parts: a structure which describes the dimensions and colours of the bitmap, and an array of bytes specifying the pixels of the bitmap. The internal representation is related to data structures directed to image processing. For instance, after segmentation is performed on an image, it can be described as a set of regions. The boundary of the region can be completely specified by chain codes. These representation are useful for further processing on the image such as classification. In addition, the class serves as a basis for future extensions of functionalities using inheritance relationships.

The Batch Script is a sequence of batch commands. Each command has the syntax

```
macro [Files ...]
```

The data structure stores the lists of image operations as specified by the macro, together with the list of images that the macro should apply. The structure is used by the Process Scheduler for execution.

Finally, the History Stack contains the temporary filenames that the History Manager used for saving an image to disk before an operation is performed on the image. The stack is implemented using a circular bounded buffer. When the buffer is filled, the filenames at the top of the buffer are discarded, together with the corresponding images which have been previously saved to disk by the manager. Consequently, the History Manager only keeps track of a fixed number of most recent steps.

## 4. DISTRIBUTED SYSTEM DESIGN

On the remote computer, the processing functions are in the form of commands. Each function is built on the model of a standard UNIX filter. The input formats to the command are specified by the individual commands, depending on their requirements. The PC generates commands to execute these functions remotely using the TCP/IP "exec" service. The service executes the command after verifying authentication, it mediates the transfer of input and output data before and after the command. This relieves the responsibility of setting up a module at the remote computer that handles the data transfers.

To establish network connection for a distributed computing environment between the host computer and the PC system, designers try to make each distributed application behave as much as possible like the non-distributed version of the program. In essence, the goal is to provide an environment that hides the geographic location of the computers and services, making them appear to be local. The first step in distributed programming is to define the protocol for communications between the computers. In the system, the TCP/IP protocols are chosen because the remote machine provided TCP/IP services which allow users to login to the server remotely. TCP/IP allows computers of all sizes, from many different computer vendors, running totally different operating systems to communicate with each other. It forms the basis for the world-wide Internet which is a wide area network of more than a million computers that literally span the globe.

Whenever an application program uses TCP/IP to communicate it must interact with the operating system to request service. From a programmer's point of view, the routines supplied by the operating system define the interface between the application and protocol software. Because TCP/IP is heavily influenced by the UNIX operating system, the application interface extends the conventional I/O functions available on UNIX. The result became known as Socket Interface.

The Socket abstraction is analogous to a file descriptor in the conventional I/O. Once a socket has been created, it can be used to wait for a incoming connection or to initiate a connection. The way the socket is used is entirely determined by the application. In this system, a wrapper class is built around the Socket Interface so that an object oriented interface is created. With is configuration, the Socket class hides the details of handling sockets as a conventional I/O file descriptor, rather taking the literal meaning of a socket, as an access point for the user, where data transfer takes place. The Network Manager thus handles and manages a set of sockets as required by the application.

The system is being used as the development platform for a number of real-time image processing projects. One of the most important is digital mammogram analysis. Due to the subtle nature of the microcalcifications, the very high resolution required in the digitization results in 32 Mbytes for each digital mammogram. It is a real challenge to process the image in real-time or near real-time. The processing algorithm is implemented on a Cray supercomputer which network is connected to the front-end PC at the University of Sydney via the Internet. A near lossless compression scheme is used to reduce the amount of data before transmission. Initial results show that the system can provide real-time performance. Detailed report will be available at the time of conference presentation.

## 5. CONCLUSIONS

This paper presents the design and implementation of a IBM-PC based distributed image processing system. The set-up provides a means for real-time image processing by using a remote host computer for computationally intensive tasks. The PC is used to mediate the data transfer of images between the local and remote devices and for the display of the images. In addition, classical image processing operations such as spatial filtering and enhancement are also implemented on the PC. As a result, an alternative method is provided for real-time image processing which was only available for more expensive systems.

## 6. REFERENCES

[1] R.L. Miller, "High resolution image processing on low-cost microcomputer," *International Journal of Remote Sensing*, vol. 14, no. 4, pp. 655-667, 1993.

[2] R.A. Schowengerdt and G. Mehldau, "Engineering a scientific image processing toolbox for the Macintosh II," *International Journal of Remote Sensing*, vol. 14, no. 4, pp. 669-683, 1993.

[3] Inmos Ltd., "Some issues in scientific language application porting and farming using transputers," *The Transputer Development and Systems Databook*, Inmos Ltd., 1989.

[4] D.E. Comer and D.L. Stevens, *Internetworking with TCP/IP Vol III: Client-Server Programming and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[5] W.R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley Publishing*, Reading, Massachusetts, 1994.

[6] B. Kauler, *Windows assembly language and systems programming*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[7] J. Martin and J.J. Odell, *Object-Oriented Analysis and Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[8] B. Stroustrup, B., *The C++ Programming Language*, 2nd Ed., Addison-Wesley, 1991.

[9] C. Ghezzi, M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[10] Borland International, Inc., *Borland ObjectWindows for C++*, Borland International, Inc., 1993.