Portable Image-Manipulation Software: What Is the Extra Development Cost?

Yves Ligier, Osman Ratib, Matthieu Funk, René Perrier, Christian Girard, and Marianne Logean

A hospital-wide picture archiving and communication system (PACS) project is currently under development at the University Hospital of Geneva. The visualization and manipulation of images provided by different imaging modalities constitutes one of the most challenging component of a PACS. It was necessary to provide this visualization software on a number of types of workstations because of the varying requirements imposed by the range of clinical uses it must serve. The user interface must be the same, independent of the underlying workstation. In addition to a standard set of image-manipulation and processing tools, there is a need for more specific clinical tools that can be easily adapted to specific medical requirements. To achieve this goal, it was elected to develop a modular and portable software called OSIRIS. This software is available on two different operating systems (the UNIX standard X-11/OSF-Motif based workstations and the Macintosh family) and can be easily ported to other systems. The extra effort required to design such software in a modular and portable way was worthwhile because it resulted in a platform that can be easily expanded and adapted to a variety of specific clinical applications. Its portability allows users to benefit from the rapidly evolving workstation technology and to adapt the performance to suit their needs.

KEY WORDS: digital imaging, workstation, image-manipulation software, portable software, PACS.

ONE OF THE MOST challenging problems of picture archiving and communications system (PACS) development is the design of appropriate image-display manipulation tools adapted to medically oriented users. Because there are different requirements that depend on the clinical usage and the imaging modalities, it is necessary to provide different types of workstations in different sectors of a PACS. Reporting stations require a very high-speed display of a large number of images, whereas manipulation workstations need high processing performance. However, the aim is to provide a common user interface on the different hardware platforms to minimize the need for user training and support. The users should only need to be trained once to use this variety of workstations.

To achieve such a goal, we decided to develop a modular and portable software called OSIRIS[1] for the display and manipulation of medical images provided by the different imaging modalities. It is designed to handle images from different imaging modalities as individual images or in sets of images. This development is part of a hospital-wide PACS[2] under development at the University Hospital of Geneva. OSIRIS was developed using object oriented programming language C++ and its user interface is based on a graphic, window-based environment. Two different windowing environments are currently supported: the standard X-11/OSF-Motif and the Macintosh system (Apple Computer, Cupertino, CA). Our selection of windowing environment was based on the criterion of availability, manufacturer independence, and official support. We opted for Unix-based workstations with an X11 windowing system because it can be considered a standard. We also chose the Macintosh system because it is a widely available desktop system in the hospital (at least in Geneva) and many users have requested the ability to use it also as a medical imaging station. The choice of the X11 windowing system was made to insure portability over a large range of Unix-based workstations. Furthermore, development on the Macintosh platform was carried out with a common core showing portability to non–Unix/X11-based platforms.

The software is designed to allow easy customization and expansion of the image manipulation and analysis tools while maintaining a uniform user interface with the same basic functions accessible on all platforms. It must be adaptable to specific needs. For example, radiologists and clinicians do not analyze images in the same way; consequently, they require different processing tools. The system must also be easily expandable to accommodate new analysis tools needed for certain types of images. This

From the Digital Imaging Unit, Center of Medical Informatics, University Hospital of Geneva, Geneva, Switzerland.

Address reprint requests to Yves Ligier, PhD, Digital Imaging Unit, Center of Medical Informatics, 24 Micheli du Crest, University Hospital of Geneva, 1211 Geneva 4, Switzerland.

Reprinted with permission from Medical Imaging VI: Picture Archiving and Communications Systems, Society of Photo-Optical Instrumentation Engineers, 1992.

0897-1889/92/0503-0008$03.00/0

software is designed as a platform for new software developments onto which special tools and extensions can be easily added.

This report describes the OSIRIS software program and its structure. The different features and tools of the program and the software architecture are described. The cost and extra effort required for such a development is discussed further at the end of the report.

OSIRIS DESCRIPTION

General Design

The OSIRIS software is designed as a general digital medical image-manipulation and -analysis software. It can be used in different clinical implementations with some degree of customization while maintaining consistency between different setups. The user interface, based on windows, icons, buttons, and menus, is adapted to the needs of physicians and does not require extensive skills in operating a computer. An OSIRIS session can contain several windows, each of which contains a set of images specific to a patient examination or study. The basic structure of the software is designed to be easily adaptable to different imaging modalities for radiological and nonradiological images. It is intended as a software program for interactive manipulation and analysis of images rather than a simple viewing tool.

OSIRIS is intended to be used with images from different imaging modalities, and as such it should provide a variety of generic processing tools applicable to different images. The processing and analysis tools are divided into two different types: the generic image-processing and -analysis tools, and the more specific quantitative and clinical analysis tools. The latter can be implemented in a modular fashion according to the needs of specific users. The first type of general image manipulation and processing tools is further divided into the following categories: (1) image manipulation tools, such as zooming and panning, contrast and intensity adjustment, magnifying glass, rotation and flipping, and reordering images; (2) image processing tools, such as filters, adaptive histogram equalization, isocontours, image combinations, display of global and regional histogram, and cross-section histogram; (3) annotation and drawing tools, such as text annotations and arrows and lines; and standard analysis tools, such as coordinate, angle, and local density measurement and regions of interest (ROIs).

OSIRIS offers an intuitive and user-friendly access to the generic tools such as zoom functionalities, inverse video mode, magnifying glass, rotation, flipping, image reordering, and color manipulation. Figure 1 shows an example of an OSIRIS window. The most important tools (zoom, inverse, etc) are directly accessible through buttons located on the left of the main window, and the other tools are accessible through the different menus. All operations can be executed through the manipulation of a pointing device such as a mouse or a trackball.

Image Display Functionalities

Using a conventional graphic window-based user interface, images are displayed in overlapping resizable windows. Because digital image modalities tend to provide sets of images (tomographic images, dynamic images, etc), a window may contain one or more images.

For the display of sets of images, two different modes of image presentation are provided: the stack mode and the tile mode (Fig 2). The tile mode allows display of all the images of a set side by side. The stack mode allows browsing through the images sequentially displaying only one at a time. A dynamic mode is also provided in which all the images of a set are displayed sequentially in a movie mode. This mode is often used to simulate real-time movements, such as a beating heart, or for viewing volume images from different angles. The speed of this movie mode is adjustable. Several movies can be run simultaneously in different windows for comparative evaluation.

Contrast and Intensity Adjustments

Another important basic tool is the manipulation of image contrast, intensity, and color settings, generally referred to as intensity window width and level. It is provided by the means of a dedicated control panel (Fig 3) on which the user can choose among different color look-up tables (gray scale or pseudocolors) and transformation functions (linear, logarithmic, etc). The adjustment of the range of intensities is performed through two cursors representing the lowest and highest intensity values as well as

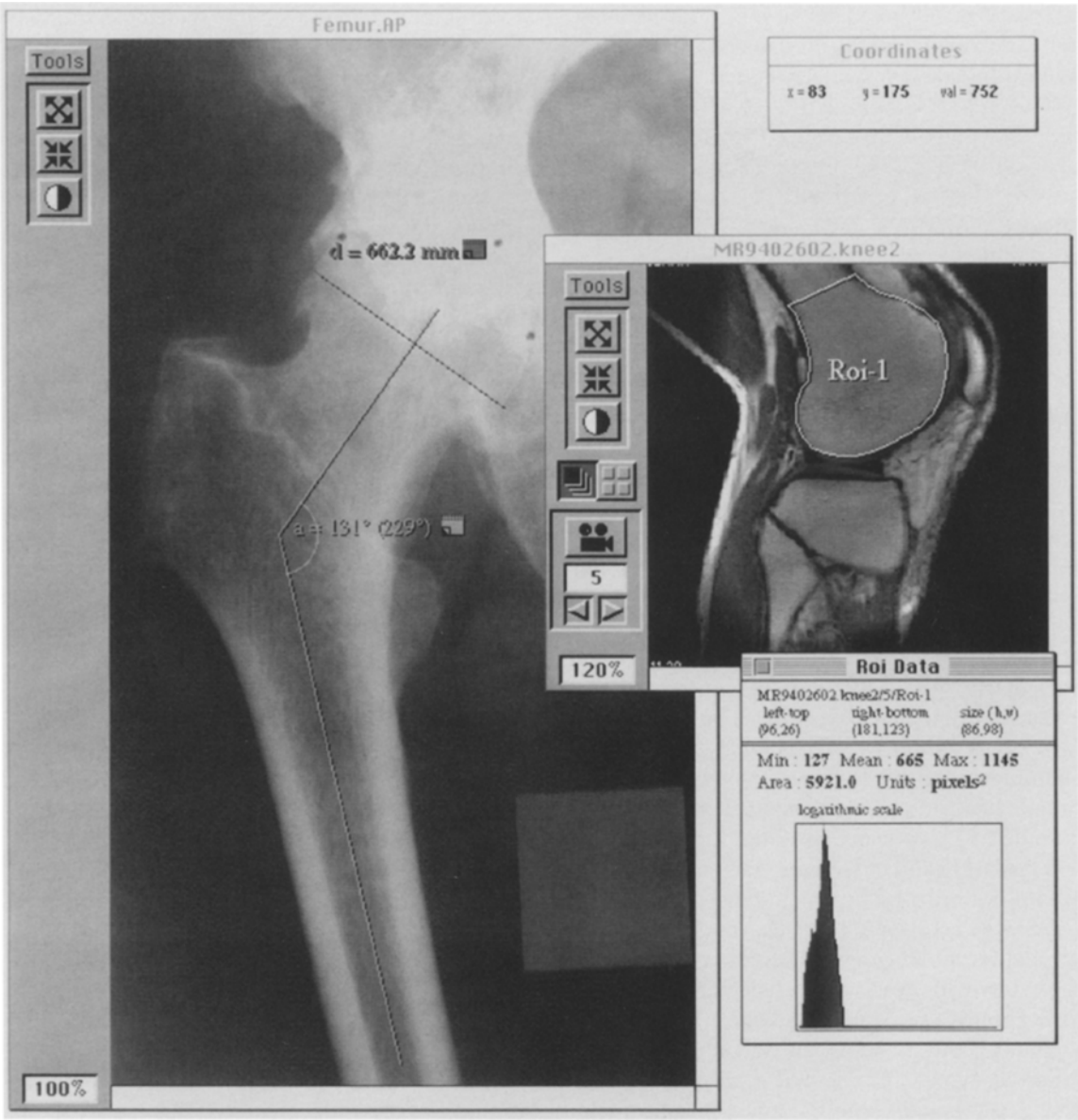

Fig 1. Typical window of the OSIRIS user interface.

by setting the intensity window width and level values, allowing easy adjustment of the contrast and intensity of an image interactively in real time. Also, because images displayed in different windows could be used for comparative evaluations, the settings of each window are adjustable separately.

The dynamic range of some images may largely exceed the range of intensity levels that can be displayed at one time on a screen. The software performs an optimal mapping of the image intensity values into the display range. Typically, medical images will have an intrinsic dynamic range of up to 12 or 16 bits deep (4,000 and 65,000 intensity levels, respectively), whereas most of the common display systems support 8-bit displays that will allow only 256 levels to be displayed at one time. Interactive adjustment is made in real time on the available dynamic range of the display (typically 256 levels), and

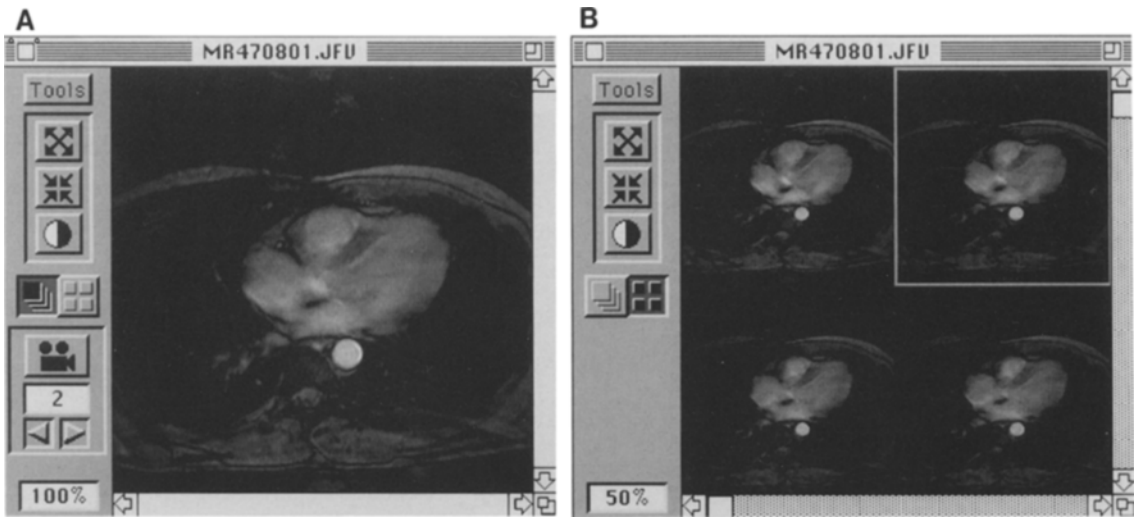

Fig 2. Stack (A) and tile (B) display modes.

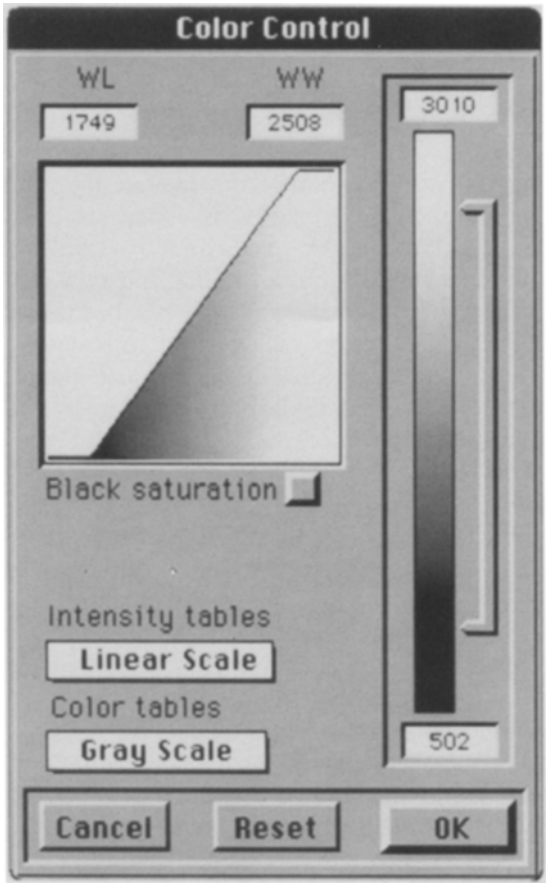

Fig 3. Color-control dialogue window.

when done the image in its full dynamic range is remapped to the screen.

Overlays

OSIRIS provides a way of defining and displaying graphic overlays that can be added directly to an image and manipulated as separate objects. Two kinds of overlays are possible: annotations and regions of interest. Annotations allow the emphasis of certain parts of an image by attaching a short text to a given point in the image (Fig 1). With OSIRIS, an annotation consists of two parts: a short label attached to a point on the image with an arrow pointing to it, and an extended text that is displayable in a "Post-it," a pop-up window that resembles Post-it brand self-adhesive notes (3M Commercial Office Supply Division, St Paul, MN) where the user can type comments and descriptive remarks related to a particular object on the image.

A second type of overlay is provided for regional analysis of the images and their content. Regular and irregular ROIs can be drawn manually on the images. Regular ROIs consist of circles, ellipses, and rectangles, and irregular ROIs can be drawn as polygons or using a pen tool for free drawing of irregular geometrical shapes. An ROI can be edited or modified at any time, allowing the user to further adjust or adapt the contour to corresponding structures

on the image. An ROI not only is an outline but also consists of a specific region of the image on which different measurements and processing tools can be applied, such as filters. Statistical data of the contents of the region (maximum, minimum, and average intensity values, histogram of intensity distribution, etc) can be displayed in a separate window. All the overlays are handled as objects that are separate from the image itself. They can be moved, modified, deleted, or saved with the image.

Image-Processing Tools

OSIRIS provides some generic image-manipulation tools such as zoom, image rescaling, magnifying glass (which allows the enlargement of just a portion of an image), rotation, flipping, and reordering of the images. Processing and analysis tools such as filters, adaptive histogram equalization, isocontours, image combinations, cross-section histogram, coordinates, angles, local density measurement, and ROIs are also provided.

These image-processing functions provide a generic set of tools that can be used by more specific analysis tools. For example, ROIs can be generated by automatic or semiautomatic image segmentation algorithms. These segmentation algorithms can be developed for particular imaging modalities, and the resulting contour will have the properties of manually defined ROIs. Conversely, quantitative analysis tools (using clinically relevant parameters) can be applied to manually drawn ROIs on an image or set of images. Typical examples of specific analysis tools could be the measurement of vascular stenosis, the evaluation of cardiac wall motion, or the quantification of the shape of some bone structures.

SOFTWARE ARCHITECTURE

Design

The two environments selected for the development of the OSIRIS software are the Unix-based workstations with X11 and OSF/Motif windowing system and the Macintosh computer family with its own windowing environment and the MacApp development library provided by Apple. These two environments are very different, thus making it impossible to carry out the development of OSIRIS for one platform and then install it on the other. We decided to develop OSIRIS for both platforms simultaneously while trying to minimize system-specific developments. We opted for an object-oriented approach[3] that allowed us to structure the software into different modules in a relatively independent manner. We designed most modules as abstract data types. An abstract data type describes a family of data structures not by an implementation (as an algorithm would) but by the list of services available on the data structures. From the outside world, an abstract data type is viewed as a "black box" providing a set of services. Such an approach allows us to specify what a module has to offer to other modules without exposing what it is and how it is implemented. Designing each module as an abstract data type ensures a high cohesion inside each module while limiting the coupling between them. We chose C++ as the standard implementation language. Its notion of class allows the implementation of the concept of abstract data type.

After choosing the implementation language, we had to solve the problem of system-dependent libraries. The graphic user interface, including the management of the mouse, the keyboard, and the windows, is based on such libraries. Thus, we have been forced to study the different system libraries in detail and elaborate a virtual library that can satisfy our needs and be implemented on each type of workstation. This library is not just a tool kit, it also defines the core of the application structure as the MacApp library does on the Macintosh platform. There is no such generic structure provided with X/Motif.

Figure 4 shows the OSIRIS software architecture. The UIN (Unité d'Imagerie Numérique) kernel defines a generic application structure from which OSIRIS is directly inherited. This kernel has been implemented on both systems. Most of its implementation is system specific, but include common object definition and class interfaces. The OSIRIS kernel is a second layer built on top of the UIN kernel that provides the basic functionality for the management of a set of digital images. It includes some system-dependent parts such as the objects composing the user interface (eg, buttons and menus). Some other parts are system independent such

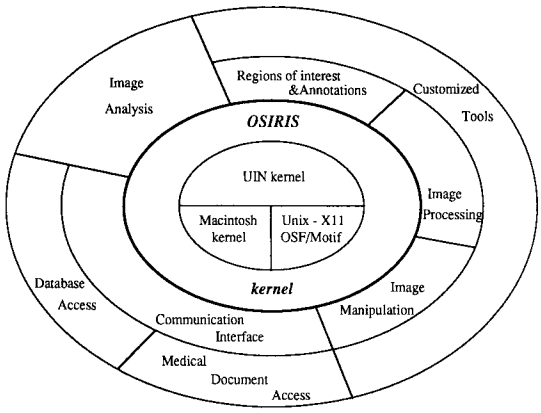

Fig 4. OSIRIS software architecture.

as the management of a set of images in a window (eg, OMedImage, ORealImage, OLayoutManager, and OMedStudy).

Figure 5 presents the generic application structure provided by the UIN kernel and the derived application structure used for OSIRIS. It is important to note that the UIN kernel is more than just a tool kit providing some independent tools. On the Macintosh side, it has been implemented with the MacApp library, which defines a similar application structure. On the Unix-based workstation, such a library does not exist and one had to be developed.

The UIN kernel includes a set of classes of which the most important are the following: UINApplication (creation/destruction of an application, management of a list of documents), UINDocument (reading and writing of a document in a file), UINWindow (management of windows), UINPane (management of the different parts, called panes, that constitute a window), UINPixPane (management of a pixmap in a pane), and UINDialog (window dialogues).

Object Ownership Structure

Figure 6 gives an overview of the object ownership structure, which is more appropriate than the class hierarchy structure to give a different view of the complete software architecture.

The most important objects of this structure are the following: the object Medical Study manages a set of images specific to a patient examination. The number of images can vary from one to 100. Each object of this type corresponds to a window in which images are displayed. In a normal OSIRIS session, several windows can coexist. The Layout Manager takes care of the management of the different display modes (stack and tile modes). The Medical Image handles all the information related to an image such as the image itself, annotations, ROIs, and the image-related data. The Real Image deals with the original data read from disk (image with 8- or 16-bit pixel depth) and provides some basic image-manipulation facili-

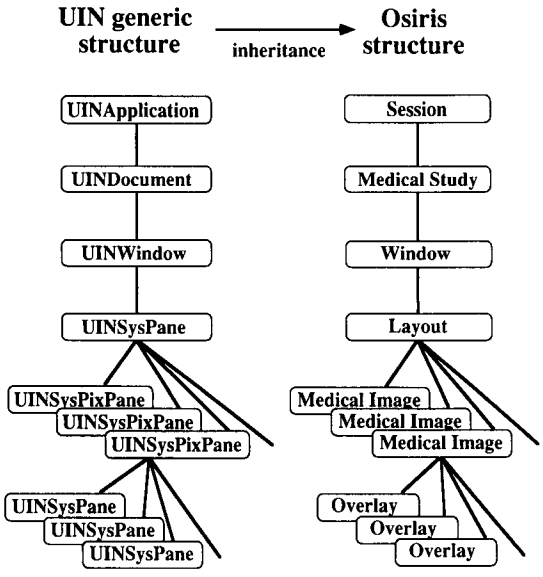

Fig 5. Generic application structure and OSIRIS application structure.

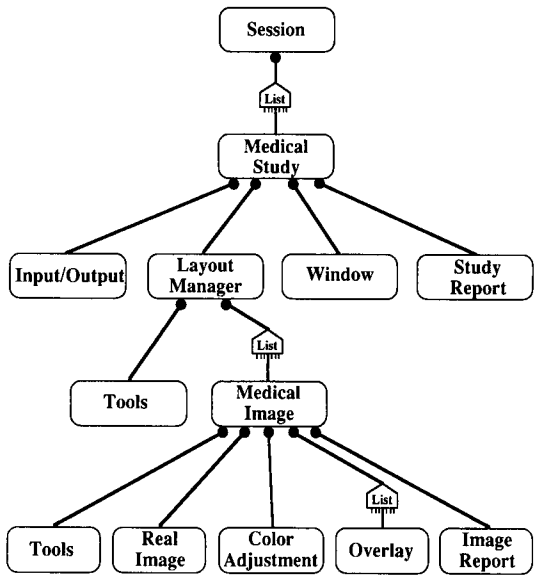

Fig 6. Object ownership structure.

ties such as rotation, flipping, and filtering. The image display is managed by the parent class UINPixPane of the class Medical Image. The displayable image pixel depth is system dependent (currently 8 bits on each system) and can be different from the original image pixel depth (which is usually 8, 12, or 16 bits). The color adjustment is realized by the OColorPanel class.

The user interface has also been designed and implemented using an object-oriented paradigm. Every component is an object. Thus windows, menus, icons, and buttons are independent objects capable of handling the basic display and behavior functions and send a message to other objects when activated. The user interface objects are generally inherited from the system library classes and are system dependent, but the user interface remains the same for any workstation. Our virtual library binds directly into MacApp on the Macintosh and into OSF/Motif widget library under X-11.

Extensibility and Portability

OSIRIS has been designed to be used as a platform allowing an easy integration of more specific analysis tools. The object-oriented paradigm is based on the notion of modularity and independence between modules. A new tool can be developed as a separate object, and its integration requires the modification of the source code in only a limited number of other objects.

A new tool will work according to the following schema: (1) activation through a menu command, (2) possible requirement of a dialogue window to specify some values or options, (3) access to different data structures, (4) treatment on these data, and (5) display in a result window.

Activation through a menu command is the only task requiring the insertion of some instructions in the existing code. The dialogue and result windows can be easily developed from the UINDialogWindow and the UINDataWindow provided by the UIN kernel. They are system independent. A complete schema is provided by OSIRIS to access the different data structures such as image pixel values and overlays. The treatment to apply to the data must be defined by user specific algorithms.

We decided to develop OSIRIS simultaneously for two different types of workstations. It was therefore important to precisely identify which objects would be system dependent and to group them in a basic library. The goal was to avoid the development of identical software modules in parallel and to develop only a single module compatible with both systems. Thus, the task of porting OSIRIS to another environment would mainly consist of implementing this basic library (UIN kernel) with the facilities offered by any new graphic environment. Furthermore, the X-11 windowing environment is available on a large variety of Unix-based workstations on the market and should allow an easy transfer of OSIRIS to all of them.

WHAT IS THE DEVELOPMENT COST?

To the generic question of extra development cost, the simplest answer is that the cost is higher than that for just one development but lower than the cost of two separate developments. To give a more elaborate answer, we must consider the following questions. Why is such a development desirable? Is it necessary? Are there any advantages? What are the potential long-term savings that can be expected from such added development costs?

Considering our hospital setup and users' requirements, it was necessary to provide our software on the different environments (Sun/Unix/X-11/OSF-Motif–based workstations and Macintosh workstations). It was important to avoid having different development teams for each environment working separately. Such an approach would result in two completely different software programs that would be harder to update and maintain. We opted to have a single team developing a single software with two roots.

The team is composed of four full-time people including two designers and two analyst/programmers. The two designers work jointly to elaborate the software structure by defining objects, each working as an expert for each system to ensure the compatibility of the defined objects with both systems. The two programmers work independently, one on a Unix-based workstation and one on a Macintosh. Designers are also involved in the programming effort according to the same schema. One part of the code is system specific (mainly the basic

library, the UIN kernel), developed for both environments, but another part is system independent, developed by one person. Such a software component is then controlled and integrated by another team member for the other system. This control by a different person is very important. It serves as a quality-control mechanism because an effort is made to produce easily understandable codes, and the discovery of bugs at this phase saves a considerable amount of time. Up to now (July 1992) the source code is approximately 45,000 lines, of which two thirds are system independent and one third is system specific, corresponding to the implementation of the UIN kernel and the basic user-interface objects. With the system's current architecture, an advanced user can easily develop custom tools (eg, processing filters or quantitative analysis algorithms). The code of such specific tools should be based on the OSIRIS kernel and therefore remain system independent.

At the beginning of this project (June 1990), the specifications of the OSIRIS project (user interface and functionalities) had already been established as a result of a first prototype developed on the Macintosh and called the CALIPSO/Explorer.[4] From that point, we encountered different problems because it was our first real experience in software development according to the object-oriented paradigm. We spent a few months trying to elaborate the right object structure, but none was found to be satisfactory. We decided to begin implementing a first version and then to refine our structure as we progressed. The following development environments were also new for the team: a new programming language C++ (common to both environments), new windowing systems X-11/OSF-Motif, and MacApp. The first months were spent developing expertise with these different environments.

We chose the C++[5] programming language because it is an emerging standard and is widely available, but there is no consensus about libraries. The main problem is that there are different windowing systems on different machines. The tool kits available for building applications are also different. The solution adopted in OSIRIS consists of designing a virtual kernel, the UIN kernel (Fig 4). Parts of this kernel need to be implemented for the different host environments considered (depending on the hardware, the operating system, and the windowing system). Because it has already been implemented on two different environments, we assume that it will be easier to port the kernel to a new environment.

CONCLUSION

OSIRIS image-manipulation and display software is being developed at the Geneva University Hospital to display medical images. It constitutes an essential component of the PACS system under development. This software is designed to be portable across a variety of workstations to satisfy different performance needs. The present version runs on a Sun workstation (Sun Microsystems, Inc, Mountain View, CA) under Unix-X11-OSF/Motif and on Macintosh workstations. A special effort has been made to design the user interface and the software architecture in such a way that is usable by physicians and non–computer-oriented users and easily adaptable to their needs. The choice of a window-based graphic user interface provides the most flexible and convenient environment for the implementation of complex image-manipulation tools through a graphic and icon mouse-driven interface. The use of object-oriented programming allows a modular and expandable architecture. It also allows better portability between platforms through the isolation of hardware-specific functions in generic object classes.

Although the development costs of software, which is portable across very different workstation environments, is higher than for a single environment, the quality of the software produced is significantly superior. The result is a product that is both flexible and maintainable and therefore likely to evolve over time. The OSIRIS package and its source code are currently being distributed to universities and research groups to promote the development of additional processing and analysis tools. Several groups are also porting this software to different hardware platforms. It has been ported to IBM RS-6000 workstations (International Business Machines, New York, NY), to Hewlett-Packard HP-APOLLO 9000 (series 720) workstations (Hewlett Packard Co, Palo Alto, CA), and to

the DEC-5000 family from Digital Equipment Corporation (Maynard, MA).

Along with the local usage in Geneva, this platform was also adopted as part of a European teleradiology project called TELEMED, regrouping 17 partners from 9 different countries. It is used as a consultation workstation for long-distance teleradiology using high-speed networks over Europe. Further developments are in progress that will allow cooperative work between different users who can remotely manipulate images on several workstations simultaneously. This extra feature of program synchronization on multiple workstations is a specific requirement for teleradiology and remote consultation.

REFERENCES

1. Ligier Y, Funk M, Ratib O, et al: The OSIRIS User Interface for Manipulating Medical Images. Proceedings of NATO ANSI meeting on Picture Archiving and Communication System (PACS) in Medicine, Évian, Switzerland, 1990

2. Ratib O, Ligier Y, Hochstrasser D, et al: Hospital Integrated Picture Archiving and Communication System (HIPACS) at the University Hospital of Geneva. Proc SPIE 1446:396-404, 1991

3. Booch G: Object Oriented Design with Applications. Redwood City, CA, Cummings, 1991

4. Ratib O, Huang HK: CALIPSO, an interactive software package for multimodality medical image analysis on a personal computer. J Med Imaging 3:205-216, 1989

5. Ellis M, Stroustrup B: The Annotated C++ Reference Manual. New York, NY, Addison-Wesley, 1990