Object-Oriented Frame approach for improved interfaces design for image processing systems

Yu.V. Vizilter, Yu.V.Morzeev, A.A. Stepanov, S. Yu. Zheltov

State Research Institute of Aviation Systems (GosNIIAS)
7 Vicktorenko, Moscow, 125319, Russia
tel. +7 (095) 157.97.48, fax. +7 (095) 157.50.97
e-mail: yury@fenix.niias.msk.su

ABSTRACT

It is generally accepted viewpoint that the object-oriented programming (OOP) provides the most powerful technique for graphic interface design. Being problem-independent, such object-oriented window-based interfaces can be of use for any problem-oriented software design. In the image processing software design, the usual idea of such interface architecture is to work with images as with documents. However, this paradigm is not adequate here, because in image processing tasks user concentrates more on the whole processing scheme than on any individual image. The special software architecture named as Software Frames Net was developed to support the new, scheme-based interface. Our Software Frames are the objects with links and slots. Any processing scheme can be represented as a network of such frames. This approach provides visual programming of image processing schemes in the most natural way. The technique is proposed that makes it easy to adapt such frame-based software for any hardware and software platform. This technique allows to use any standard object-oriented libraries for the design of improved, frame-based interfaces.

We propose to use the *processing scheme window* as a basic interface unit.

1. INTRODUCTION

It is generally accepted viewpoint that the object-oriented programming (OOP) provides the most powerful technique for graphic interface design. The modern applications for *MS Windows/Windows NT* and *Apple Macintosh* programs prove this proposition by their nice look and very simple and attractive window-based user's interface. Being problem-independent, such object-oriented window-based interfaces can be of use for any problem-oriented software design. It is only the form that should be filled by the proper compound.

Today we have a lot of low-cost image processing systems on PC that look as well as *Microsoft Word for Windows* and work correspondingly. As an example we can consider some of well-known programs such as *PhotoFinish*, *PhotoStyler* and so on. Usually the software of this type accompanies some desk scanners (or other certain digitizers) and this circumstance determines both the compound and user's interface. These programs work with images *as with documents.* They prepare images to be insert into documents (formed using special text processor or publisher) and, finally, to be printed. So, it is easy to understand, why these programs use the standard *Multi-Document Interface (MDI).* There are many C++ libraries that support the object-oriented MDI design in the very efficient and easy way. Thus, image processing software developers can mainly concentrate on the problem-oriented programming.

However, in the general context of image processing as well as in any problem-oriented image processing area (such as medicine image processing, digital photogrammetry, multi-sensor sensing and so on) the mentioned paradigm - "work with images as with documents" - is not adequate. There are, at least, two important reasons to develop the other paradigm for interface design. First of them is the following. While the document-oriented systems make the emphasis on the resulting data (document) forming, the general-purpose (or problem-oriented) image processing system must provide and interpret a whole set of different results obtained at different stages of complex processing. We can describe this activity in terms of final report forming but it will be the very artificial trick. In practice, user concentrates on the whole *processing scheme* more than on the individual images. The other reason is connected with processing of renovating data. Image processing systems often must

organize the "on-line" processing of image data. Here, the "on-line" term means that all dependent data should be renovated while the corresponding source data are renovated. So, the user needs to have the possibility to operate with such processing schemes that could *process data automatically* after their edition is finished. That requires, in particular, to provide the visual edition (or visual programming) of processing schemes because the processing schemes must be observable for user and allow the edition in this observable (visual) form.

The outlined reasoning leads to the new type of user's interface for image processing systems. While the traditional document-based interface uses the *image window* as a basic interface unit, we propose to use the *processing scheme window* as a basic interface unit. All of possible *data windows* (*image windows*, *histogram windows*, *profile windows* and so on) are available too. However, they are hardly connected with their images (*icons*) on the processing scheme and do not exist without the processing scheme. Additionally, the *processing procedures* and *control operators* should be represented in our schemes: in the scheme window (in the iconic form) and (if it is required) as the individual windows of corresponding types. These windows are also can not exist without the scheme.

The new technique named *frame-based programming* was developed to implement these ideas. This object-oriented technique provides the visual programming and the following automatic functioning of arbitrary processing schemes represented as the *software frame nets*. This programming technology and corresponding improved user's interface will be described in detail applying to the general image processing context. The frame-based image processing system PISoft will be presented as an example of application of this technology.

2. FRAME PARADIGM AND SOFTWARE FRAMES

Any interface can be usable and efficient only in the case when it reflects the true internal structure of the system. So, the desired interface, mentioned above, supposes some software architecture that provides its functioning. At the beginning of this work we have formalized our intuitive comprehension of such architecture in the form of set of required features. They are the following:

- the **system must be controlled by events**;
- the **objects** of the system **must generate and process system events (system messages) automatically** according to data renovation and some set of logical rules;
- the **objects** of the system **must be interconnected by** the oriented **links** that determine the **possible ways of propagation of system messages**.

We refer these ideas as a *Frame paradigm*, because the objects of such system have all features of *frames* that are well-known in Artificial Intelligent area [1].

To implement this paradigm, we have developed a new type of the program objects called *"software frames"* (SF). In the prorgamming context, the "software frame" notion generalizes the usual "class" notion that is a basic conception of the object-oriented programming (OOP). When the usual OOP (based on the "classes") concentrates on the design of objects with their properties, the Frame-Oriented Programming (FOP) mainly deals with object interconnections and interactions.

In the most general case, our frames are the objects with *links* and *slots.*

Links are the special data structures that determine the behavior of the frame. There are links of two types: "*synchronic"* and "*information"*. Any frame activity can be generated only as a reaction on the message from the other frame connected with this frame by the "input" synchronic link. Conversely, any frame can send the message only for those frames that are in the list of its "output" synchronic links. Any external data are accessible for reading and writing only through the "input" and "output" information links and only when the frames-"owners of data" permit to do this. So, isolated frame can not act and can not "see" any external data. Frames work only together.

In the programming context, we can say that any *slot* is a pointer to the object of some type. It means that any frame is a *container* object that can contain some other objects. In some sense, frames provide the software interface for their slots and, conversely, slots express the semantic essence of their frames.

This approach presumes any data processing scheme to be represented as a *Processing Frame Net* (PFN). The PFN is an asynchronous net of software frames interconnected by links. Any PFN must include *Data Frames*, *Processing Frames* and *Control Frames*. The frame of any type has the *input and output synchronization links*. Besides:

- any data frame has a *data slot* of some type, and the type of stored data determines the proper type of corresponding data frame;
- any processing frame has a *procedure slot* containing some processing procedure and a set of *link*s (pointers) to the required input and output data slots; the proper type of the processing slot is determined by the type of executable procedure;
- any control frame has a *control operator slot* filled by some control operator (e.g., if, case, for, etc.) and the set of *links* that provides the required data from other slots that participate in the conventional (Boolean) expression of the control operator; the proper type of the control slot is determined by the type of implemented control operator.

The programming of PFN supposes the following steps:

1) creation of the required number of data frames copies;

2) creation of the required number of processing frames copies and tuning their connections with data frames;

3) creation of the required number of control frames copies and tuning their connections with data frames, processing frames and other control frames.

The PFN is the software architecture that, being realized in the graphic manner, provides the improved interface for image processing systems. In the next section we shall consider the interface of this type as it was implemented in our C++ *"OFrame for Windows"* library.

3. FRAMES AND WINDOWS

The described above frame-based technique is obviously platform-independent. However, the certain frame-based graphic interface is always connected with some graphic platform such as MS Windows, X-Windows, etc. Thus, our software contains two parts. The first part realizes software frames as basic object class *TBasicFrame* and provides their functioning in the arbitrary frame net. This sublibrary can be carried onto any hardware platform with any operation system. The second part of our library provides the certain graphic interface for such frame nets. It depends on the concrete platform using.

Especially for frame-based interface design, we have developed the efficient technique that allows to incorporate interface elements into the software frames. The main idea of this technique is very simple. We evolve the class *TVisualFrame* (from basic class *TBasicFrame*) by adding of two new slots. The first slot contains the interface object that visually represents this slot on the scheme and named as *TFrameIcon*. The other slot contains the special window object that visually represents the slot data (if it is required) and named as *TFrameWindow*. Besides, the object of *TFrameWindow* type (a) provides special menu connected with this frame and (b) processes system eventes generated by the concrete platform (if your platform generates them).

Any programmer whenever tried to develop such interface objects can say that this task is sufficiently hard. Fortunately, this work was already done many times before us and there are many ready-made libraries that can be of use. Usually they contain at least one important class that describes the proper window with its menu and message analysis procedure. Our *TFrameWindow* object can be easily produced from this class.

Our software platform was the IBM PC with Windows 3.1 operation media and we have used the object-oriented library *OWL* (ver. 2.0.) by Borland Int. that provides the easy support of MS Windows' basic interface elements. So, our *TFrameWindow* is based on *TWindow* class contained in *OWL*.

The other special object - *TFrameIcon* - is the very simple structure. It processes only two events: *frame activation* and *frame deactivation*. Correspondingly, it can "draw" the frame *icon* (symbolic notation) on the scheme in two different states: *active* and *passive*. The sense of the "activation" event is the following. When the human operator wants to work with the certain frame in the interactive mode (for example, tune the procedure parameters), he should select its icon on the scheme. Then the *TFrameIcon* object changes the icon from passive to active state, actuates the frame window and executes a lot of other required actions. After that, the interactive access to the frame is available through the frame window and its menu. Since the work with this frame is finished and the user selects the other frame, the "deactivation event" takes place. It forces *TFrameIcon* object to change the icon from active to passive state and execute all of other predicted actions. So, you will have no serious problems with this object at any software platform.

There is one more important object that we did not mention yet. It is the *frame net manager.* This object, is realized in the introduced sublibrary as a *TFrameManager* class and it provides the control over the list of system messages. For interface generation, this object must be associated with the scheme window and provide the *message loop* that processes both the frames' messages and the messages from operating system (if it generates them). Applying to *MS Windows*, we have associated this object with the *TApplication* object from *OWL* and overwrote the *MessageLoop* method of this object. Thus, we obtained the new class *TFrameApplication* that can generate Windows applications provided by our improved interface.

Using the technique described, one can develop the analogous interface library for any software and hardware platform. Moreover, if you have some original image processing system developed (or developing) using the usual "document-based" interface, you can easily reconstruct this system to the frame-based manner. Your ready-made window objects can be directly used as the *TFrameWindow* objects; your data objects and procedures can fill the corresponding slots in data and procedure frames. For instance, the previous version of our PISoft software had the usual Windows' application interface. The whole reconstruction of our project to the frame-based manner was finished in three months. And the frame-based architecture and interface provide the new quality for our system.

4. PISOFT SYSTEM

In this section we shortly describe the *Visual PISoft for Windows (ver.7.0)*. It is the image processing workbench provided by visual programming of modular processing schemes. It also can produce the executable programs (Windows' exe-files) that realize the user's processing schemes developed in this workbench.

The offered software PISoft is destined for the researchers in the field of image processing and can be equally used in the research and education spheres as an integrated image processing media.

The basic innovation of the offered software which differs it from the preceding versions is the usage of the original *Visual Programming Conception* based on the original **Frame Approach**. This conception provides the possibility to create the semantic frames net, which carries out the required processing problem in the visual mode.

The PISoft modules make up the logically closed problem-oriented enlargement of the service library OWL and make it possible to insert the elements of the raster images analysis and processing in the originals user's programs. Besides, the PISoft 7.0 library contains both the image processing procedures and some additional tools that facilitate the researcher to create his own original software (3D-visualization windows, profile windows, histograms windows, various measuring tools, etc.)

The 7.0 version has inherited the following well-known processing procedures:

- nonlinear rang and sigma filtering;
- linear convolution with Gauss and arbitrary masks;

- local derivatives and gradient orientation estimation with modified Sobel operator;
- edges and areas extraction by Marr operator;
- morphological operations of enlargement, compressing, opening and closing with arbitrary masks;
- histogram transformations and threshold segmentation.

Visual PISoft for Windows allows to realize the full image processing cycle by the visual programming of the frame net. The worked out PISoft-applications can be used to solve the wide range of the problem-oriented image processing problems. So, we can refer the *Visual PISoft* as a software developer's tool. It allows to design complex image processing applications without any of language-based programming.

5. CONCLUSION

In this paper we outline the new technique named as the frame-based programming that provides the improved visual interface for image processing software design. To implement these ideas in practice, we have developed a new type of the program objects called "software frames". Our software frames are the objects with links and slots. Any image processing scheme can be represented as a network of such frames. The programming technique of the frame-based interface design is discussed and the frame-based image processing software PISoft is briefly described.

The main results are:

- generic image processing software does not work with images as with documents but the most of modern object-oriented interfaces support only this style of work.
- the improved interface must use the entire processing schemes as the basic logical units.
- any processing scheme can be efficiently represented as a software frame net.
- the frame-based approach provides the visual programming of image processing schemes in the natural and simple way.
- the frame-oriented programming provides powerful data management and simultaneously allows to design attractive windows-based interfaces for image processing systems.
- the technique proposed makes it easy to adapt the frame-based software for any platform.

The suggested application Visual PISoft for Windows (ver.7.0) is the excellent illustration of the described principles and ideas.

REFERENCES

[1] **Stepanov A.A., Vizilter Yu.V., Morzeev Yu.V., Zheltov S.Yu.,** *"Complete Object-Oriented Image Processing"*, **1995**, (to be published).

[2] **Vizilter Yu.V., Morzeev Yu.V., Zheltov S.Yu., Stepanov A.A.,** *"Frame Paradigm and Object-Oriented Image Processing for Photogrammetry*", **1995**, (to be published).