HERMES : AN OBJECT-ORIENTED MULTITASKING SYSTEM FOR CONCURRENT DIGITAL SIGNAL PROCESSING APPLICATIONS

Antonios Anagnostopoulos and Georgios Kouroupetroglou
Division of Communication and Signal Processing,
Department of Informatics,
University of Athens, Athens GR 15784, Greece
e-mail: koupe@di.uoa.gr

ABSTRACT

This paper presents the design and implementation of the PC-based multitasking system HERMES, which supports the development of concurrent Digital Signal Processing (DSP) applications using object-oriented programming techniques under the MS-DOS operating system. The signal abstractions by Objects of the HERMES system and its architecture are described along with the software framework that enables the realization of highly reusable and modular code for rapid production of DSP tools and applications that can cooperate in real-time.

1. INTRODUCTION

Nowadays, the combination of general purpose PCs with advanced software techniques, such as object orientation, are used more than ever for real-world signal acquisition, processing and testing. Although there is a wide range of commercially available powerful and attractive digital signal processing software for rapid algorithm development (e.g. SRL [1], ISP [2], QuiqSig [3], MATLAB [4]) there is a lack of object oriented software systems for PCs to handle multiple concurrent processes and applications.

The development of such a system and its applications is desirable to conform with the following requirements: i. *Constant and uniform flow of information*: since the input information of such a system usually consists of a constant and continuous flow of data (as they are sampled or monitored by an A/D converter), the applications that manipulate these data should not fragment or alter the data flow. ii. *Processing-time optimization*: architectural decisions can have a greater impact in the overall throughput of a system than pure code optimization. iii. *Software reusability*: abstract application classes must be identified to be generic enough in order to fulfill the requirements of many similar applications'. iv. *Minimization of refinement overhead*: the abstract application classes must be easily refined, overriding the fewest possible methods. v. *Re-entrancy*: all the applications must be developed taking into account that their methods are to be preempted at arbitrary locations by the multitasking kernel, and to be called simultaneously by an arbitrary number of other applications' methods.

Although there are a few Object Oriented Operating Systems available, (some of them designed for real-time or even distributed processing), the HERMES system is an MS-DOS executable, eliminating the need of familiarizing its users with a new operating system and allowing for other useful applications/utilities to reside in the same workstation.

2. SYSTEM ARCHITECTURE

The HERMES system's architecture consists of several software layers (Fig. 1) in order to constitute, along with the underlying hardware, a platform that efficiently supports and executes applications. It is a closed architecture, meaning that each layer is implemented in terms of operations or classes of each vertically adjacent lower layer, thus reducing the dependencies and providing information hiding between layers.

As object orientation is proven to be a sound basis for the development of DSP systems [3,5], the HERMES system was designed and developed using the Object Modeling Technique (OMT) [6]. By using this object oriented methodology, that supports graphical notations for representing object-oriented concepts, we can provide a direct analogy between objects in a design and objects in the problem domain, making the design more intuitive and easy to understand. An overview of the Object Model of the HERMES system is presented in Fig. 2.

2.1. The Multitasking Extension Layer

This Layer is responsible for providing to the

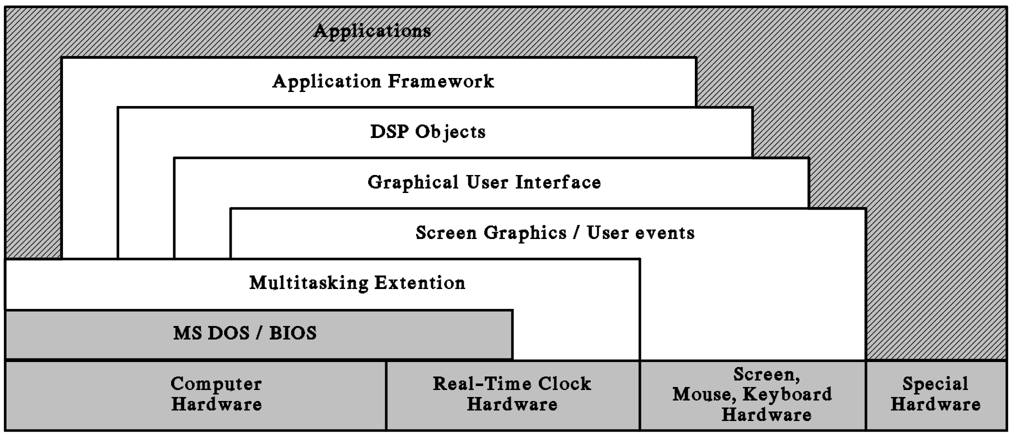

Fig. 1. The HERMES' layered architecture.

HERMES system some essential capabilities that are not offered to ordinary DOS applications; such as multitasking operation, linear memory addressing, shared memory and extended file handling capacity. This Layer consists of the following components:

i. The Multitasking Kernel that defines the Task Object and provides preemptive scheduling of tasks allowing many applications to operate concurrently. Task switching is implemented based on the round robbin scheduling algorithm. Furthermore, software triggering is offered in order to allow applications to trigger the task switching mechanism themselves.
ii. Memory Objects for the management (allocating, de-allocating and accessing) of the PC's extended memory. These allow linear addressing and access to up to 4 Gbytes of linear memory and provide local storage for the signal objects' buffers. The Memory Objects automatically align the requested memory to a 32bit boundary, for optimizing the system's performance in single and double precision arithmetic.
iii. Signal Objects for storing and editing data series. These Objects are designed to manipulate two major signal categories: signals of fixed or predefined size (such as prerecorded waveforms, windows, spectrums etc.), and signals of undefined size (such as waveform generators, continuous data acquisition). Furthermore, the Signal Objects' methods ensure protection of the stored data and synchronization of tasks that access common Signal Objects using properly defined semafores. The synchronization mechanism exploits the software triggered task switching mechanism by forcing the tasks that operate on common Signal objects to use only as much CPU time as is required to perform an operation. In this way there is no need to define different priorities to each task. This poses the following side-effect: although the maximum period that each task can be active is predefined, the resulting task switching frequency is higher, thus improving the system's performance.

2.2. Screen Graphics / User events Layer

This layer is a library of functions and objects responsible for providing all the basic screen, mouse and keyboard I/O operations with respect to the multitasking extension layer. Since the screen hardware is not designed to be addressed by a multitasking environment, the low level graphics functions are not re-entrant. Therefore, they have to block the Task Switching Kernel, thus establishing a temporary non-multitasking environment, and then to unblock it after completion of each screen operation. These graphics functions are hand-coded in assembly language ignoring any similar BIOS interrupts, in order to access the screen hardware directly ensuring thus fast screen updates. The keyboard and mouse actions are modeled by the User Event Object in the form of either the Keyboard Event or the Mouse Event Object and stored in the Event Queue.

2.3. Graphical User Interface Layer

The GUI layer provides a set of consistent graphical objects, with a predefined appearance and behavior, that serve as interaction components and as visualization components for graphically displaying the computational output. One of the earliest realizations of an object-oriented dialogue-independent software architecture for GUIs is the

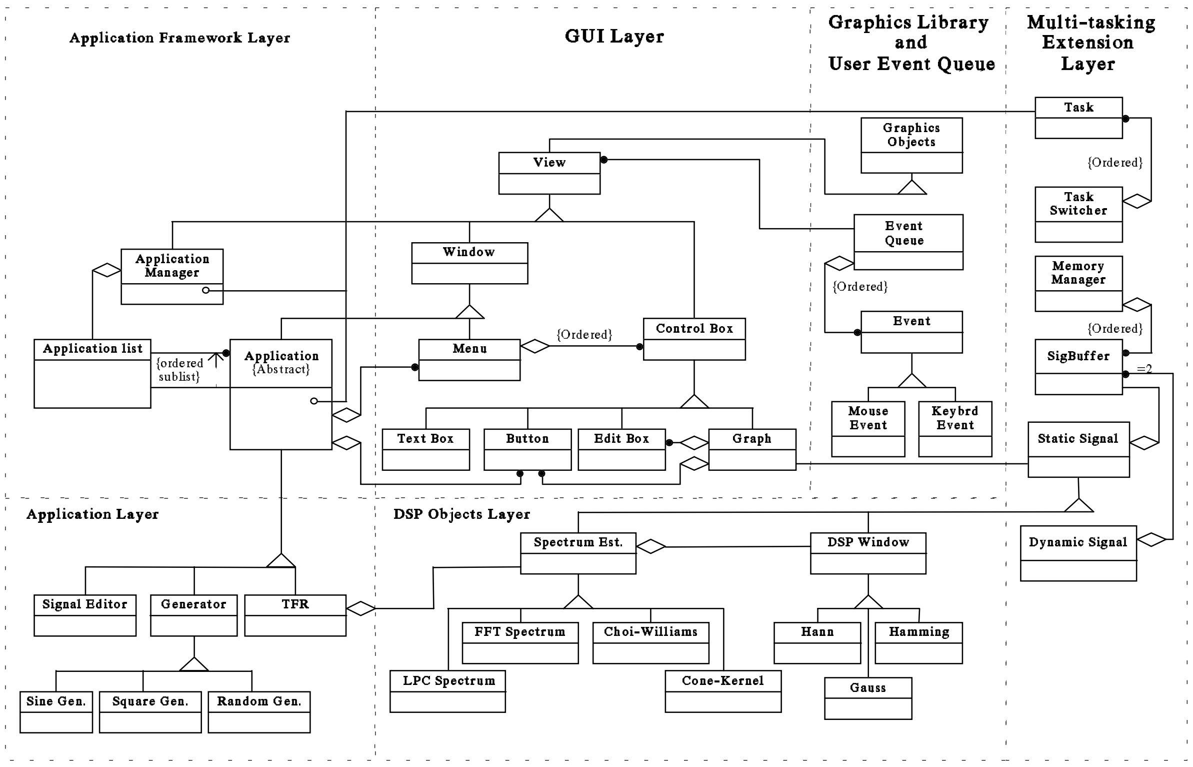

Fig. 2. Overview of the Object Model of the HERMES system.

Model-View-Controller (MVC) introduced in Smalltalk-80 [7]. In HERMES, a modification of the MVC software architecture is adopted: the Controller becomes a method of View, thus reducing the dependencies between application functionality (Model) and user interface related actions (View and Controller). Since Views may contain several sub-Views to implement complicated interaction structures such as menus, dialogues, etc., the correct Controller to receive user actions is automatically invoked; thus, a situation of having many Views accessing the User Event Queue simultaneously is avoided.

In this layer several interaction objects are introduced in order to control and configure each application (Windows, Buttons, EditBoxes, Menus, Dialogues etc.) as well as to visualize the Signal Objects in the form of signal, spectrum or spectrographic plots.

2.4. DSP Objects Layer

The functionality of the Signal Objects is specialized in this layer and novel objects are introduced in order to provide reusable signal processing objects that simplify the integration of the DSP applications. These objects implement: predefined weighting functions (Hamming, Hanning, Triangular, Rectangular), configurable weighting functions (Kaizer, Gaussian, weighted cosine), FIR filters, spectrum analysis (short time FFT, LPC) and Time-Frequency Representations (TFR) (Wigner-Ville, Cone-Kernel, Choi-Williams). In order to enhance the system's performance, all the methods that implement DSP operations are coded in assembly language. Additionally, the DSP objects provide also the appropriate methods that implement any necessary dialog to configure their behavior.

2.5. Application Framework

The Application Framework layer consists of the Application Manager object and the Application abstract Class. Only one instance of the Application Manager is allowed, since this is responsible for initiating (and terminating the) multitasking system, the graphics screen, the Event Queue and for instantiating the applications.

The abstract Application Class is a member of the Window Class responsible for properly connecting each application to the Application Manager, establishing itself as a separate task and providing the functionality for handling its screen appearance, initiating menu operations and terminating the application. An important feature

	Spectogram	Wigner - Ville		Cone Kernel		Choi - Williams	
Analysis Size	Real (msec)	Real (msec)	Analytical (msec)	Real (msec)	Analytical (msec)	Real (msec)	Analytical (msec)
64	0.2	0.2	0.7	0.3	0.8	0.5	1.2
128	0.4	0.4	1.5	0.7	1.9	1.6	3.8
256	0.8	0.9	3.0	1.8	4.9	5.7	12.6
512	1.7	1.9	6.6	5.4	14.1	18.4	45.4
1024	3.7	3.9	14.0	17.9	44.7	70.1	178.3

Table 1. Performance of the assembly language implementations of the Spectogram, Wigner-Ville, Cone Kernel and Choi-Williams analysis methods.

of the Application Class is that it allows any of its methods to be established as independent tasks, thus enhancing the flexibility of the application development. This class should be refined to perform the desired operations in order to become a complete application.

2.6. Applications Layer

A number of Application Objects are currently available under the HERMES system to perform: signal editing, window design for Rectangular, Triangular, Gaussian, Kaizer, Hamming, Hann, and other weighted cosine windows, windowed FIR design, Signal generators for sinusoidal, squarewave and random signals, signal acquisition through a plugged-in A/D board, spectrum analysis and spectographic display of TFRs. These are members of the Application Class and can be used either as they are to perform experiments, or as a basis for further refinement to produce custom applications.

3. RESULTS

Since Signal objects are allowed to be shared amongst applications, the HERMES system can be used in a variety of ways: various operations can be performed either in parallel (for example various spectrum estimations of a common signal), or in series in order to cooperatively perform complex experiments. Furthermore, many independent experiments can be performed concurrently exploiting the capabilities of the workstation.
The HERMES system is capable of producing results in real time provided that the operations involved are fast enough. The measurement of the performance of some analysis methods (including all necessary operations i.e. windowing and real to analytical transform) that produce logarithmic power spectrum estimations are presented in Table 1. Measurements are performed on a PC equipped with a Pentium @ 75MHz CPU.

4. CONCLUSIONS

Although not a programming language, the HERMES system by specifying high level constraints, significantly simplifies the development of DSP experiments by either modifying or combining the proposed objects/applications. The information-hiding of the object oriented architecture verified that a potential developer of tools or applications does not need to be involved with the details of its structure, beyond a familiarity with the declarations (statements) provided for communicating with it. Furthermore, since the underlying architecture supports multiple threads and data protection and synchronization, applications can cooperate to produce results even in real-time depending on the sampling rate and the overall calculation complexity.

5. REFERENCES

[1] G. Kopec, "The Integrated Signal Processing System ISP," IEEE Transact. ASSP, vol. ASSP-32, No. 4, pp. 842-851, Aug. 1984.
[2] G. Kopec, "The Signal Representation Language SRL," IEEE Transact. ASSP, vol. ASSP-33, No. 4, pp. 921-932, Aug. 1985.
[3] M. Karjalainen, "DSP Software Integration by Object-Oriented Programming: A Case Study of QuickSig," IEEE ASSP Magazine, vol. 7, No. 2, pp. 21-31, April 1990.
[4] The MathWorks Inc., Natick, MA. Matlab User's Guide, Aug. 1992.
[5] P. Daponte, L. Nigro and F. Tisato, "Object-Oriented Design of Measurement Systems," IEEE Transact. Instrum. Meas., vol. IM-41, No. 6, pp. 874-880, Dec 1992.
[6] J. Rumbaugh et al., *Object-Orinented Modeling and Design*, Prentice Hall, Englewood, 1991.
[7] G. Krasner and S. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, No. 3, Aug./Sep. 1988.