Ghost/Spook
User Interface and Process Management in the PACE framework
Pedro Reis dos Santos Helena Sarmento Luís Vidigal
INESC IST/INESC IST/INESC
INESC - Instituto de Engenharia de Sistemas e Computadores
IST - Instituto Superior Técnico, Universidade Técnica de Lisboa

Abstract
PACE is an open system designed around a message-based communication protocol (Spook) and includes user interface (Ghost) and database management systems.

Spook offers new and powerful mechanisms to implement different message-based protocol vocabularies in a vendor-independent way for interprocess communication and file storage.

Ghost is a window system independent message-based server that uses Spook. It offers a full graphics editing process rather than an input-parsing utility.

Introduction
In an earlier phase of Computer Aided Design for integrated circuits, the emphasis was placed on the development of individual tools to assist the designer. However, the increasing complexity of integrated circuit design, involving the use of many dissimilar tools, shifted the emphasis to the development of complete CAD systems where the use of tools is simplified, minimising the overall design time. Current research addresses the problem of ensuring that basic mechanisms, covering things such as the human interface, data and design management, intertool communication and process and project management, are provided in order to allow a common operating environment to integrate CAD tools and to be used by designers.

The requirements of a common environment for CAD users and applications depends on designer and programmer needs. Concerning the human interface, the CAD designer seeks a powerful graphical interface consistent across tools, extensible and configurable. A network of workstations provides an underlying system where the designer can take advantage of a dedicated CPU while maintaining the benefits of timesharing over a large number of machines. Process management offers mechanisms to ensure the communication between CAD applications running on different machines. Services like process management and user interface are used by different tools. If they are implemented as stand-alone subsystems, the programmer can easily incorporate them into future tools.

CAD research and development already address user interface and process management issues.

The Oct/VEM[1] system offers a fully integrated environment where applications run on multiple machines communicating through RPC mechanisms. VEM is a UIMS based on the X[2] window system with a consistent and uniform user interface oriented for VLSI layout development. It guarantees the dialogue between the end user and the applications.

The EIS specification provides services for accessing, controlling, managing and exchanging design data and software. The goal of the EIS programme is to reduce the cost of large system development, especially by facilitating tool integration and data exchange. The UIMS component of EIS[3] is a multi-window interface where the application's appearance and behaviour from the user's point of view is specified by associating interaction objects with its script. Interaction objects have appearance and display behaviour while scripts formally describe dialogues. The execution environment is the run-time support for executing scripts.

This paper aims at describing current work at INESC concerning user interface and process management as services provided by a framework. Section 2 describes the requirements of user interface and process management. Sections 3 and 4 describe PACE[4] user interface and process management services. Finally, section 5 presents some conclusions and future work.

Requirements
With the advent of sophisticated graphics workstations connected in a local area network, powerful user interface and large computing resources are available. Process management controls the execution of a given task by multiple asynchronous processes on the network. An User Interface Man-
agement System (UIMS) is a tool designed to tailor and manage user interaction in an application domain to allow for rapid and consistent development.

**User Interface**

An UIMS can be described from two different viewpoints: the viewpoint of the software developer and the viewpoint of the end user.

A software developer regards an UIMS as a tool that provides support for the definition of the user/application dialogue, imposes external control on the application, offers a common presentation of the application's output and includes a component ensuring the interaction between an application and an end user. The use of an UIMS encourages programmers to develop better software by:

- providing a consistent user interface between related applications.
- encouraging development and use of reusable software components.
- insulating applications from the complexities of the environment.
- shielding applications from the effects of the end user.
- offering easy learning and use of applications.
- supporting error handling and recovery.
- allowing tailorability and extensibility of applications.

From the viewpoint of an end user, the primary goal of an UIMS is to support the easy and effective use of an application. An UIMS provides the following advantages to an end user:

- consistent user interface across applications.
- multiple levels of help or assistance.
- training.
- end user tailoring of the interface.
- extensibility of application.

**Process Management**

First computers were single-tasking, and processed batch jobs in sequence. Later, timesharing systems let several people use the same CPU, and share files as well. While this provided good interactivity, performance suffered with many people sharing a single CPU. Networks are being developed both to connect existing machines and to take advantages of the low-cost, high-performance microprocessors the semiconductor industry is turning on. The first networks were used to allow users to log onto remote machines or to move large data files back and forth among a group of machines. Then workstations made it practical for each user to have a dedicated CPU, while maintaining the benefits of timesharing. Now, workstation system users can receive computing services from the entire network of machines, within a single application.

To be successful, the system must make it easy for programmers to:

- make individual procedures run on a variety of computers.
- write an application capable of running either locally or over a network.
- provide network and vendor independence.
- ensure good overall performance.
- offer excellent tools for application development.

The ideal environment for the design of electronic systems is a network of workstations, sharing a single file system and interprocess communication. A circuit can be edited on a graphics workstation and interactively simulated on a powerful mainframe, while the results are sent back to the workstation for simultaneous logging.

**GHOST**

Ghost is the framework user interface management system – UIMS – providing a uniform and consistent interface across tools. It controls the presentation of data, handles input and activates application modules.

The Ghost is a server that like X allows foreign applications to log themselves, to present their output and to receive user events. Besides the functionality of X or NeWS[9], Ghost offers a very high level description of the output primitives, handles the user interaction and can help applications to perform more specific operation (for instance, dragging, rubber-banding, picking and selection).

For Ghost, an application is collection of algorithms and rules to be applied to graphic objects (output primitives). An application loads Ghost with the graphic representation of its data and the description of its commands, including argument types and the way arguments can be acquired (mouse, keyboard, etc).

**Architecture**

As mentioned before, Ghost is a server where applications, running as independent processes, connect themselves through IPC mechanisms (see figure 1). An application can also be linked to Ghost as an extension, needing only to recompile the code with a different compilation parameter. This improves the performance of the application algorithms, especially if they require intensive graphic input and output.

The Ghost server is responsible for the identifier mapping. All objects created by an application are referred by an identifier (ID). The ID has two major function: firstly it offers
remote applications a way to refer to the objects they created (a pointer is only valid in the process address space); secondly it offers a simple way to share or just refer to other tool's objects. Ghost is responsible for handling the mapping and translation between IDs and memory positions, to allow tools to access objects.

Application Interface

The Show class and the other four associated classes, Scene, Actor, Group and Sentence, comprise the Ghost application interface (see figure 2). The instances of this C++ classes are therefore objects that tools can create and manipulate and so they are the only objects in the system with an ID. The Show class has no ID since it is the tool entry point. It controls the tool connection to the Ghost server and is identified by the connection stream that it holds. This connection does not represent a real connection, since the application can also run in the Ghost process as an extension. If the application runs as an independent process, the underlying Spook protocol handles the interprocess communication.

The imaging model is based on two different kinds of objects: actors and groups. An actor is a collection of points, lines, polylines, rectangles, circles, etc. and is itself a graphic primitive. An actor can be compared to a graphic segment as defined in graphic packages like GKS[7], CGI[8] or PHIGS[9]. In Ghost the actors or graphic segments are editable and hierarchical. A group describes how a graphic primitives is to be drawn. Its functionality may be compared with GKS bundles, X graphic context or EDIF[10] groups.

A Scene is a sheet of white paper without physical limits onto which the application can draw as well as receive user commands. A Scene can be a schematic sheet, a waveform display, a layout or just a palette with the available primitives. For each of these Scenes the tool can draw different objects and as they are functionally different, different commands can be defined on each one. On the other hand, if some sharing arises the application can use objects of one Scene on any other, provided that the Show is the same.

Application Commands

Ghost aims to free applications from the tedium of input processing, such as low level input events like an input character or a mouse button press. To provide powerful input processing, Ghost loads a detailed specification of each command from the application. The application only defines the expected input data (integer, string, point, actor, etc) and suggests the interaction mode. How the actor is picked or selected depends on the Ghost user dialogue language. For instance, to define a command to move an actor the application specifies the first argument as an actor, the second as a point and suggests that dragging should be performed.

The Sentence controls the input. Each Sentence is associated with a command. It is through Sentences that commands are created and downloaded to the server, and they are also activated by the server when the respective command was given by the user.

Protocol

Each object in the server has a twin object in at least one tool. This twin of the Ghost object allows the tool to perform indirect operations on the server object by manipulating in a transparent way its own object in its own address space. This ensures that conceptually different tools can be integrated together and, as far as the communication protocol is unchanged, the server can be modified without having to recompile every tool. Ghost defines a set of messages as an extension to the Spook (see next section) base protocol. Using an object-oriented approach the messages are grouped around the class object they refer to.
SPOOK

SPOOK (Stub Protocol Kernel) is the framework subsystem providing mechanisms for process communication and low level data storage. It accomplishes, in a machine independent way, the usual file input/output and server-client communication (see figure 3). Spook supports several simultaneous protocols describing different vocabularies for inter-application dialogue. A protocol is a set of messages understood by the two tools involved in the communication. A message is an atomic transfer entity with an opcode and a pre-defined format.

Client-Server Communication

Clients and server, using the SPOOK protocol, communicate through a bidirectional channel. If client and server are in the same machine, the connection between processes is established using UNIX IPC (in the current version). Otherwise, a protocol such as TCP/IP or DECnet is used. Only TCP/IP is currently employed.

To offer a highly interactive environment, a high bandwidth channel is needed. Therefore, a virtual circuit communication like the X window system protocol is adopted instead of the datagram approach generally used by RPC packages [11].

Folder Storage

In addition to the client-server mode, Spook provides data storage mechanisms. Objects packed into messages can be read from or written to a file. In order to store a message (object) the tool must provide a unique identifier for later access. Identifier generation is managed by another layer, releasing the tool from this task, as in a database system.

A transaction is a set of messages selected from a folder table. The transaction definition depends on the access mode. In store mode all folder messages (objects) are written, while in update mode only the selected ones are written.

If store mode is chosen, the version number is increased and objects are written to a new version of the file. In update mode the update file is used for appending, preceded by a header which includes a timestamp (machine current time). Two reading modes are also provided: load mode and restore mode.

Protocol

Each instance of the class Protocol represents a different protocol (vocabulary) that can be used by tools. The class Protocol ensures the compatibility of the vocabulary status (version, release) and implements the set of services (class SpookService). The services are accomplished by three derived classes: Client, Folder and Server (see figure 4). The Server class holds the addresses where clients can connect themselves. Whenever a client requests a connection, the server creates a ServerClient instance that holds the new connection stream. The SpookService includes information such as a descriptor and numeric representation, allowing communication between processes running on machines from different vendors. An instance of class Client is created by an application requiring a service offered by a server. To create this instance the server address must be supplied. Messages can be synchronous (request) or asynchronous (send). The ClientBuffer and Server-
Buffer classes deal with the encoded messages. They get from SpookService the numeric representation type of the encoded information. This class has functions to add and get the different data types. The information transmitted on the buffers includes an opcode identifying the messages.

Scheduling
To schedule client-server message communication and queue handling a module has been developed. This module can process the low level input/output as well as to keep track of the process state. An input processing routine will be called whenever a message is received and an output function is activated when output can be performed without blocking the output stream. I/O routines are supposed to exchange messages between queues and I/O streams.

When there is no I/O pending the scheduler will scan for pending messages on any queue under its control. If some message is pending, a programmer supplied routine is called to process it. This processing is done until no more I/O is pending and all queues are empty. The programmer can supply queues that are not associated with any I/O stream, using them as storage mailboxes.

Conclusions
In this paper we described the user interface and process management subsystems provided by the PACE framework.

Spook embodies the system's process management, which provides for dynamic linking of applications through interprocess message-based facilities. It offers a powerful mechanism which allows the end-user to have a virtually unlimited set of components. Spook allows communication among processes running in different machines from different vendors. Spook is responsible for byte swapping and floating point conversion when processes are running in dissimilar architectures, respecting the alignment constraints imposed by those architectures. Using the same vendor-independent conversion mechanisms, Spook also provides an archiving utility to store system objects.

The future of UIMS lies in their ability to provide features beyond those normally found in hand-coded applications. Our UIMS consists of a message based server. It is easy to add a Ghost interface (Show) to applications, because of Ghost's self-contained object-oriented structure and message based communication protocol. The application interface is easy to customise, because of its object-oriented structure, in which the class hierarchy can be extended for different applications. Ghost has been designed to be window system independent. It currently runs on X11, but can be easily ported to other systems. The main difference between this approach and other UIMS approaches is that the interaction is seen as a full graphics data editing process rather than as an input-parsing utility or screen management program. Having adopted this view, one can identify classes of objects to hold data and control output as well as manage dialogue interaction.

Although not yet completed, Ghost already offers applications a graphical, highly interactive user interface. When fully implemented, Ghost should be able to handle a large spectrum of applications. It should be easy enough to use, so that even end-users will be able to modify the user interface.

Using services and the database subsystem a browser has been developed. Currently two different tools, a schematic editor and a wave form editor, are being developed in order to test both Ghost and Spook. These three prototypes show the potential of Ghost/Spook to reduce the coding task while providing powerful graphics and high performance client-server communication.

Future work involves the development of new drivers for PostScript printers and X11/Motif toolkit set. Two other interpreters are planned, one for LISP or similar language allowing the construction of new commands and one that provides a direct manipulation language to offer more elaborate interaction modes with the user. To enrich the applications available to the end-user, possible new tools are also being considered.

References