DESIGN AND IMPLEMENTATION OF AN INTERACTIVE GRAPHICAL ENVIRONMENT
FOR GENERIC MATHEMATICAL MODELS SYSTEM
FOR FLUID TRANSPORTATION THROUGH PIPELINES

E. Correa, N. Bencomo
Grupo de Modelos Matemáticos
Escuela de Computación, Universidad Central de Venezuela
Apdo. 47002, Caracas, Venezuela
and e-mail: ecorrea@conicit.ve, nbencomo@conicit.ve

Abstract
This document describes our experience in the design and implementation of an Interactive Graphical Environment with an objects oriented methodology, for a set of mathematical models and other common applications found on sites supervising and planning the transportation of fluids through a pipeline network. The Graphical Interface design was guided towards its utilization with different pipeline networks as well its presentation in dynamic windows environment.

I. Introduction

A Pipeline Transport Models System consist of a great amount of submodels and application modules. Consequently, they are generally not easy to use, and a friendly User Interface with high performance and modular implementation for easier maintenance becomes really important. The ability to adapt to different pipeline networks is also interesting. During the design phase of the application, we made special emphasis on creating a system with the characteristics just described.

What is presented to the user is based on information supplied both by the process that implement the mathematical models and real time information. The mathematical models were developed by our group. The real time information is the result of handling the input provided by the RTUs of the instrumentation and control system (SCADA). The information display has the following characteristics:

Fig.1 Overview of the User Graphical Interface.
- reflects graphically the physical-chemical operational characteristics that describes the crude oil transportation through the pipelines, using the following state variables: pressure, flow, temperature and density.
- shows inventory graphs and crudes’s balance.
- shows graphs, comparing values computed via the mathematical models and the values supplied by the SCADA.
- notifies the user in leaks detection and the event of their localization, conditions of over and under pressure.

Based on operational data in a given instant, the system can predict the behavior in a short period of time of the whole pipeline network.

The group of programs that compose the User interface were implemented using the C Language, the X Window System and OSF/Motif in an Unix environment, in the Mathematical Models Laboratory, Science Faculty, Central University of Venezuela.

II. Design of the Graphical Interface

Since the conception of the design, we preserved as an objective the generality of the final product. An easy-to-use graphical environment, capable of being executed in different hardware platforms or architectures and easy-to-adapt to any kind of pipelines control systems.

The methodology used for the Graphical Interface design was based on the Objects Oriented Programming (OOP) paradigm, which helped to satisfy basic principles when designing Interfaces, such as: maximum user control over the application and not vice versa, clarity and simplicity, aesthetic, robustness (minimal number of "bugs"), immediate and tangible feedback for the user and conceptual, visual, functional and linguistics consistency.

The first step was the application requirements's identification. Figure II shows a modular diagram that outlines the organization of the interface in order to satisfy these requirements.

```
Graphical User Interface

Initialize Start RTM

Virtual Instruments
Profiles
Histories
Historic Profiles
Scada - Model
Inventory
Batch Tracking
Instruments Analysis
Pick Tracking

Forecasting Reorder Environment Printing Help

Virtual Instruments
Profiles
Histories
Batch Tracking
Pick Tracking
```

Fig. II Interface Organization.
The following steps consisted of identifying the necessary objects and define the valid operations as well as their functional description.

The conception of the interface was based on a set of objects named "shells", or "shell objects", others identified objects were virtual instruments pile and palette's option.

The shell objects, or simply shells, were used as intermediaries between the window manager and the application. In consequence, a shell could be seen as a window that permits the communication with the user doing a specific requirement. A shell always introduces the of look-and-feel characteristics of the window manager being used.

Shells are comprised of other elements (labels, graphics, etc.), according to the function associated with them. Shells are so-called because they isolate its components from the external world just like the shell of an egg isolates it from the external world.

The types of shell objects that were introduced in the interface are (see Fig.III):

1. Pipeline Shell
   1.1. Instruments
   1.2. Graphs
2. Curves Graph Shell
3. Bar Graph Shell
4. 3D Graph Shell
5. Batch Information Shell
6. Instruments Analysis Report Shell
7. Option Palette Shell

The classical questions that are asked in the process of object identification and description are: what are the properties of an object ?, what is the object responsible of performing ?, Which objects are related to the this object ?, and others. The answers, in our case, should try to satisfy the required generality and user control over the application. In consequence, it was necessary that shells with graphical and numerical information, could be handle as documents on a desk, having the user even the options of reducing or enlarge them at will. That is why all the shells present shifting and resizing operations (including their content). The several content types introduced for different shells became the basis for describing different shell classes.

The pipeline tracing feature was implemented as a programming module totally independent of the shell implementation that contains it; therefore, the flexibility of displaying several pipelines topologies is guaranteed.

It has been demonstrated by experience, that the utilization of graphical representations to communicate with the user makes the underlined system more productive. The graphical components of the applications are dialog boxes and menus using icons and (2D or 3D) color graphs.

Several users could work with different versions of the graphical environment. The user has control over:

- Colors used by each application widget.
- Types of fonts used.
- Placement of the windows in the application.
- Text that appears in the application windows.

The software can be easily adapted to other languages. In this way, the capacity of the system to be deployed internationally is guaranteed.
III. Communication with the rest of the modules

The Graphical User Interface was designed as an independent module within the group of modules that comprise the pipeline control system. Each one of the modules is a process in the computer. There is a main process within the group of mathematical models that is in charge of integrating the information supplied by the processes of the mathematical models. The main process is called the manager process. Throughout the interface execution, the application keeps updating the graphs shown to the user. It is therefore necessary to provide a strict and fast synchronization in the communication between the interface and manager process.

According to this scenario, the manager/interface communication was achieved via a bidirectional data interchange using shared memory segments and system interrupts in order to gain in performance at low cost.

The first process to be executed is the graphical interface process. It then takes the following actions:

- establish the connection with the X server,
- create the shared memory segments,
- execute the manager process. The interface process sends the manager process:
  * shared memory segments identifiers.
  * the interface display identifier.

The communication manager/interface is different depending of the direction of the communication:

Direction: Interface --> Manager
The interface process communicates with the manager process through Unix signals. The information is transferred through shared memory.

Direction: Manager --> Interface
The manager process communicates with the interface using the facilities for communication among clients provided with X Window System. As in the previous description, the information is transferred using shared memory. The manager process orders the execution of a temporary X process providing the X connection interface. This process has only one function: to indicate the describing key of the current communication to the interface process. This temporary process is part of the interface, and has no relationship with the mathematical models programming.

By using the communication among client facility, many responsibilities are put on the X server and simplifies programming. The reason why this kind of communication is not used in the direction Interface --> Manager is because it compels the manager process to maintain an X connection, in which case the objective of keeping independence between the manager and the interface would not be achieved.

Shared Memory Segments description.

Among the data items kept in shared memory are:
- synchronization flags.
- data Structures associated with the graph's types offered by the interface, such as:
  . values to be drawn,
  . number of values to be drawn,
  . pipeline localization related to the graph,
  . data associated to information like: batch information in the pipeline network, leak alarms, "pick" tracking, etc.

1468
Once one of the processes uses the memory resource, the process becomes the memory owner; therefore the memory can be considered as a critical resource. However, it is possible that another process needs to use a specified data area of the shared memory that is not been used by the current owner process, and not be allowed to use it. In order to avoid this race condition we considered using several independent shared memory segments.

IV. Conclusions

By using the Xlib library we had a very fine level of control for drawing graphs in detail. However the complexity of programming was greatly increased in favor of this power.

The Xt and OSF/Motif programming libraries were used were applicable. This permitted the development modular code, which is convenient for its maintenance. It also cuts development by a big factor when compared to using lower level programming libraries like Xlib.

Implementation of this application taught us that for some applications it is definitely necessary to combine the three different programming levels of the X Window System: Xlib, the Xt toolkit and specific GUI like OSF/Motif.

Bibliography

Fig. III View of the Graphical Interface with several shells.