"The Implementation of a Common User Interface Design Across a Heterogeneous
Host/Display Device Environment"

By Kime H. Smith, Jr.

Science and Technology Department
Pacific Bell
San Ramon, CA 94583

ABSTRACT

This paper describes the design issues, architecture, and implementation of the Pacific Bell Virtual Workstation (pbVWS). The objective of this project was to provide a working subsystem which would satisfy the following requirements: 1) define and enforce a company-wide "policy" for user interaction with Pacific Bell applications, 2) enhance end-user and programmer productivity, 3) support the existing user display device base yet allow easy migration to newer equipment over the next 3-5 years, and 4) provide equal or better usability than is now available for a given class of display device.

The solution to the above problems and the achievement of the stated objectives appears possible using a command-event protocol to exchange information between an application and an idealized "virtual workstation." The device-side protocol handler takes care of the details (and some of the semantic interpretation) of the commands to control the actual device whether it is a single-function terminal (e.g., VT100, 3270) or actual workstation. The pbVWS software implements such a handler for this range of devices.

1. INTRODUCTION

To understand the design issues and the resulting architecture of the Pacific Bell Virtual Workstation (pbVWS) in the context of the Pacific Bell information systems environment, the first section describes the problem motivating the work, a high-level description of the larger software system of which it is to be a part, and the resulting design objectives for the solution. In the second section, the software architecture of the pbVWS prototype will be discussed as well as some highlights of why certain design decisions were made during the building of the prototype. Results from experiences gained using the software for a pilot application will also be described.

The third section will assess the results of the research and propose future directions for our work.

1.1 The Problem

The problem of providing information systems with a quality user interface has many facets, as viewed from either the user or developer perspective.

User View. Users at all levels have problems with existing systems; sometimes computers are even "getting in the way" of achieving maximum business effectiveness. Furthermore, our ability to adapt information support systems to changing business needs (caused by changes in organization of business units) is limited. That is, systems which were once designed to meet the requirements of a particular community of users may no longer be used by the original organization. When groups are merged, the individuals within the new group may each have to access data by means of two or more applications which have radically different interfaces. The results are higher training costs, higher error rates, and higher staff turnover.

Greater dependency on computers for business functions and user demand for better interfaces are the upward pressures which mandate optimization of over-all system performance by focusing on the user rather than the machine in our future system designs.

System Developer View. The developer is forced to deal with a new set of commands and style of interaction with each new terminal type introduced. Furthermore, if a mixture of workstations and older single-function terminals is already deployed, the interface designer is forced to design for the "lowest common denominator" — the use of equipment features must be restricted to a set supportable by all terminals using the application. Thus, PCs are often relegated to emulation of older, less capable equipment, especially when en masse replacement of single-function terminals is not practical.

1.2 Background

The Pacific Bell Virtual Workstation software was completed and first demonstrated at Pacific Bell in July, 1987. It was a result of work defining the User Interface Services portion of the Technology Architecture component of the Pacific Bell Systems Architecture. The paragraphs below briefly describe these entities.
The Systems Architecture. The Pacific Bell Systems Architecture (pbSA) represents a target ideal for the company computing environment. This vision consists of data, application, and technology architecture components, driven by a model of processes and data necessary to support the business enterprise.

Briefly, the data architecture defines what, where, and how data will be stored. The application architecture identifies the business processes to be automated, and the technology architecture provides a vision of how data, processes, and information systems users will be interconnected.

The Technology Architecture. The Technology Architecture (pbTA) is a plan for software development which may be viewed as a very coarse-grained, object-oriented, distributed computing environment. Inheritance is not directly supported, however, and there are only three classes of objects. Furthermore, only instances of these three primitive classes of objects are treated as objects in the system. The main objective of the architecture is to make it possible to build software with maximum independence from hardware platforms.

If one accepts that interaction with physical devices for data storage or I/O causes the most portability problems, then partitioning of the traditional monolithic application into data, application-specific, and user interface functions seems to be appropriate. By using a messaging service to transfer data and requests between the partitions, and defining a protocol to do this, it makes it possible to put the program logic which implements the various major functions on any appropriate physical device; it also greatly simplifies the interface.

User Interface Services. The main goal of the User Interface Services effort was to provide a common "look and feel" for users of our applications. The initial focus was to separate interface design issues from implementation issues. First, to achieve a "Single-System Image" (SSI), a definition and a specification of that image was needed. Second, a method of implementing that interface in software and insuring that the result would satisfy the major design criteria of the pbTA was required. The proposed solution for the second half of the problem formed the basis of the feasibility prototype, pbVWS, described in this paper.

1.3 Objectives and Strategy For The Solution

The major objectives centered on preserving the equipment base (while making it possible to migrate to more modern equipment without the need for massive cut-overs), separating the application and interface related program logic using a protocol (to adhere to the pbTA philosophy), and providing a useful and uniform interface to Pacific Bell applications.

To achieve these objectives, the protocol to be used had to be considered carefully and the pragmatics of implementation and reuse of the user interface services software had to be considered.

Design Goals. The following requirements were used to guide the implementation strategy:

- **Support the existing user equipment base.**
  
The solution could not require "dumping" existing equipment in favor of more modern workstation gear. With an embedded base of tens of thousands of asynchronous X3.64(VT100) terminals and an equal number of synchronous 327x (and compatible) single-function terminals, the cost to do a massive upgrade would be staggering and the logistics nearly impossible. Yet, the number of AT- and Macintosh-class workstations is also significant, and the number of engineering workstations is growing rapidly; the full power of the workstations should be exploited to support a modern user interface as well.

- **Adhere to a standard protocol for all communications between applications and the pbVWS.** This is required to support the planned pbTA by enforcing a single standard data/control stream between the application and User Interface component.

- **Provide equal or better usability than now available for a given class of terminal.**
  
The darkest nightmare of my colleagues was the belief that the result of this work for the "dumb" terminals would be a lame implementation of "yet another windowing system" with multiple second response time on the mainframes. Such an interface would indeed be a nightmare. We recognized from the beginning, however, that acceptable response time performance was a requirement.

- **Provide a consistent "look and feel" across environments.** A SSI implies that system behavior must be consistent and predictable, certainly across applications on the same system. Consistent and predictable behavior across terminal hardware domains is desirable as well for reasons given below.
Standard Protocol. Selection of a standard protocol for the application-to-data services link was far more straightforward than the application-to-user interface services link. In the former case, the Structured Query Language (SQL) appeared to be an emerging industry standard which could be adopted. A winning protocol which defines how data, commands, and events should be exchanged between an application and a UIMS has yet to emerge. X Windows from MIT, NeWS from Sun Microsystems, the OSI 12 Presentation Manager, and PHIGS have all been proposed as windowing/graphics system standards. The problem with adopting one of these systems as the protocol base for this project is that they operate at too low a level of detail. That is, they require the developer to deal at a level of detail which cannot ensure the uniformity of style we want in our applications.

By using a higher level protocol for control, two advantages accrue: 1) the system is easier to understand and use and 2) “policy” can be imposed on the resulting design. The second point has been viewed by some as a flaw in general purpose UIMSs; however, one only has to consider the success of the Apple Macintosh to see the value of cross-application consistency to a user community. The ROM-based user interface toolbox has the same “policy” enforcing effect as a higher-level (restricted) protocol. A higher level protocol also serves to shield the application from the details of specific display control. Potential problems in this approach are loss of the ability to meet unanticipated interface needs and lack of the level of detailed control necessary to implement “semantic feedback” [18] in some applications, particularly those which make use of “direct manipulation” [11]. So, a further requirement of this protocol is that it be easily extensible and provide for a broad-bandwidth connection between application and display, when required.

Virtual Workstation. The virtual workstation paradigm was chosen to make the design of the user interface (and its control) easier for the application developer. Screen manipulation and local services are predefined and guaranteed to be available, regardless of actual equipment type.

The generic software block structure is depicted in Figure 2, below. The components are as follows:

- Physical Display. This is the actual hardware on the users desk.
- Native Display Manager. This is the software layer which deals with the lowest level of detail in the context of the native OS. For example, on the Macintosh this would be the Toolbox, on the PC it could be MS Windows and the associated VDI drivers, on UNIX systems, the CURSES package.
- Native OS. The native operating system.
- Messaging. This is the instance of the messaging services process of the pbTA for this component.

pbVWS. The protocol handler and local coordinator of ancillary services such as “local” file storage, text editing capability, and the like.

![Figure 2: Pacific Bell Virtual Workstation Architecture](image)

Implementation Guidelines. Support for single-function terminals, as noted above was a given requirement. Such support is needed today for retrofitting existing hardware. In the future, such support may be needed to provide for special application circumstances. An example is the deployment of computer access using inexpensive terminals where theft is likely.

X3.64 (VT100) class devices and IBM 3278 Class devices do not have “mice”; pointing is limited to cursor control keys; furthermore, “local” services such as private storage must be implemented on the processors to which they are attached.

Support for workstations is more straightforward, as the physical device more closely resembles the virtual model. IBM-PC Class Devices for this project use Microsoft Windows as the Native manager; Macintosh Plus devices use the Toolbox (via MacWorkstation). Sun-3 class devices were not used in the project, however, “generic” UNIX support by means of X-Windows may be the correct implementation choice.

2. THE VIRTUAL WORKSTATION: pbVWS

2.1 Approach and Architecture

The pbVWS may be characterized [8] as a UIMS using multiprocess message passing with “mixed control.” The connection between the application partition and the user interface partition is a narrow band-width connection — except under special circumstances, when the equivalent of a “permanent virtual circuit” may be established for occasions which require high bandwidth, as is needed for semantic feedback or file transfer.

Approach. The general approach to getting the job done was to go fast, light, and cheap. The
results of the study were intended to determine feasibility for funding of the real development effort — both the pbVWS and the pbTA in general. The time and money constraints were very real. We established six months as an outer bound on development of a demonstration system which would include as many of the environments as possible.

Rather than invent a protocol for the prototyping effort, we chose to use one which we had licensed from Apple: the non-supported version 2.0 of MacWorkstation[14]. This protocol is a medium-level protocol intended to control a Macintosh™ from a host application — almost exactly what we wanted. The commands and events were "generic" enough so that we could use the individual event handlers to take the appropriate action, even in the case of the "dumb" terminals, by using the "semantic" meaning of a command or event and interpret it in the appropriate way given the hardware context.

For example, a mouse click on a menu item on the Macintosh™, a menu selection in MS Windows™ on the PC, or the press of the enter key on a highlighted item on a 3278 connected to a VM system were considered to be part of the same equivalence class of events and therefore generated the same protocol event which was sent back to the host application.

**Components.** Virtual Workstation support was provided by MacWorkstation™ on the Macintosh™ Plus; event handlers for all other components were written by us using the native display managers where possible: "CURSES" on UNIX™ for ASCII support, MS Windows™ for PC/XT/AT and compatibles. Communications was jerrybuilt. Since the group working on this layer was not ready, we developed some "home-brew" software to get data where needed. The VM implementation used two disconnected virtual machines, one reading from a pseudo-TTY line and writing to a file, and the other reading commands and data from the file and handling the terminal interface.

**2.2 The Prototype**

An application was needed to test the various interface modules as they were developed. Furthermore, a means of demonstrating the effectiveness of the concept of separating the application and user interface code to management was needed. In some cases, temporary communications links were developed as well (for example, UNIX™ to VM).

"EasyMail" application. The first step was to create an application which would be easily understood by the people who would be attending the demonstrations of the prototype and which would adequately test the pbVWS components as implemented. Electronic mail seemed to be a natural choice. The path of least resistance (and greatest speed) was to port the MacHost™ component of MacWorkStation to our 3B20 UNIX™ system and throw together a skeleton application. Indeed, in less than a week "EasyMail" a UNIX™ mail system front-end was on the air using the Mac window/menu/mouse interface (via MacWorkstation™ and the Mac Toolbox).

The MS Windows version was finished next, followed by the UNIX™ and VM implementations.

**System Schematic.** The diagram below shows the configuration of the various hardware/software components used to demonstrate the prototype and thus the feasibility of using a single

![Figure 3: pbVWS Demonstration System Architecture](image-url)
version of an application to generate and respond to a single protocol stream while making use of terminal equipment of wildly differing capabilities.

The circle labeled “A” represents the “EasyMail” application running on the host 3B20 UNIX™ machine. The squares labeled with the letter “M” represent the jerrybuilt messaging service processes we developed to connect the 3B20 to the IBM mainframe. The dark gray rectangles indicate the location of the user interface services software.

2.3. The Pilot Project
As a result of the demonstration, we were presented with an opportunity to use the MS Windows™ version of the pbVWS software to build a production system for a small group of managers. An outside contracting firm had a team of about 3-4 programmers working on the windows based application for about 5 months when they advised their client that their proposed system could not be implemented as anticipated.

Using our user interface code, an application which satisfied the users’ requirements was developed within about 6 weeks by one programmer. The ratio of application code to pbVWS code was 1 to 4, a ratio within the range reported in the literature. The application and pbVWS were both implemented in C and both ran, in this case, under MS Windows™ on the ATs of the users. The application facilitated the access of information on various corporate data bases.

3. CONCLUSIONS

3.1 Results
By the end of six months, the “EasyMail” application was demonstrated.

Produced a Working Prototype. The demonstration was a success in that it showed that the proposed pbTA was feasible and that a SSI could be achieved across a network through various operating systems and devices. Performance in all environments was acceptable.

Met Design Objectives. We met all our design objectives: preserve the existing hardware base (while providing a means to readily migrate to different supported hardware — the migration is facilitated by cross-domain consistency and predictability), use a single protocol to transparently handle radically different device types, and provide a good user interface in each environment.

Benefits. The benefits for the user include preservation of investment in existing hardware, easier migration to new equipment, consistent user interface among applications resulting in lower training costs, predictable changes across environments, capability to take advantage of higher performance workstations when available, and a reduced error rate. For the interface designer, a modern interface can be developed “up-front” without the need to “design around” the lowest supportable interface. The application developer deals with a single protocol and so can become more adept with it and concentrate on application functionality instead of interface details, which results in a shorter development cycle.

3.2 Industry/Academic Perspective
The pbVWS software made use of an existing protocol which itself was in many ways an extension of the protocol used to handle “families” of single-function terminals. An insight in this project was to use the protocol to support a “virtual workstation” which could be used as a mental model for the software developer to design the application’s interface for the “high-end.” At the same time, the separation of responsibility for reification of the interface to a User Interface Services Layer (object) made it possible for a single version of the application software to support users of equipment with both greater and lesser capabilities than the nominal target, with no knowledge on the part of the application that multiple kinds of equipment existed.

Most (if not all) current implementations of UIMSs support single-function terminals only, exist in the form of “tool libraries,” or require high-end workstations. Other observations follow:

Follows trends, There has been a clear trend toward code reusability, modern user interfaces (window/menu/mouse) [6], and adoption of the UIMS or developer toolkit [7] [12] as a way to improve the quality of human-machine interaction and the productivity of both the user and systems developer [3]. Interest in network operating systems is also accelerating [4] [1] [2]. The pbVWS and the pbTA are tracking these trends.

Breaks ground. The exercise of instantiating the architectural design resulted in some novel spinoffs and insights for example:

- Designing for the “Highest common denominator” in a population of terminals and workstations can be carried out successfully. To achieve this we employed “Graceful degradation” of the design from the “High-end” (workstations) to the single-function terminal. Functionality falls off as a function of hardware device competence, and control of the degradation is wholly outside of the application; it is implemented in the user interface partition. In Figure 4 (below), the UNIX™ version (b.) of an equivalent Macintosh™ window (a.) is given. Note that the parenthesis of “Hypercard” on the UNIX™ version indicates the equivalent of “dimmed” on the Macintosh™, thus indicating functions which are not available to the user, but acting as a “place holder.”

- “Dumb” terminal support was not ignored in favor of high-end workstations. A “Good” interface was obtained — for example, the pull-down menu construct on the virtual workstation forces a wide-but-shallow menu structure [3] in the mainframe implementation.
Use of a middle-level protocol is valuable because it avoids the difficulty of learning a complex tool kit, helps establish a system-wide "policy" for application "look and feel," allows for easy modification of the appearance of the interface "on-the-fly," yet gives the developer sufficient control over the interface to meet the interface needs of the application. An ideal balance between ease of use and precision of control is difficult to achieve, however; therefore, extensibility is important.

A 'User Interface Server' concept was a fall-out of the development of the pbVWS software for asynchronous terminals connected to UNIX: in some workgroups, many inexpensive terminals attached to a super-micro based UNIX system has the best cost benefit ratio.

"User Environment Compatibility" describes the way that some of our design decisions were resolved. Does one implement a "better" text editor for the user or let the user continue using his current editor? Our intuition was to take advantage of the various environments, that is, using the interface on the Macintosh the user gets a text edit window; on the PC, an MS Windows scratch pad; and on VM, an XEDIT screen. Thus, the user does not have to learn a new editor, and the application "fits" into their chosen work-environment. Yet the interfaces yield the same reasonable result across systems. When a user looks for a function, he or she chooses the same menu item in nearly the same location; the behavior of the application interface is predictable across hardware/OS domains.

3.3. Future Direction
The prototype and pilot studies need to be extended and new work in related areas needs to be done to answer additional questions:

Implications for Research. Network Operating Systems such as Mach, Sprite, and V are beginning to emerge from the laboratories [2] and present opportunities for new solutions to residual problems and alternatives to existing sub-optimal solutions. The software design process will be greatly affected by user interface design and prototyping tools [5] [9] [10] and we need research in this area.

There are several issues which need to be pursued before a production of the system can be implemented. Among them are: finding good ways to accommodate existing applications, design of a protocol to handle extensibility, fine-grained control, new semantic objects such as FORMS. Furthermore, performance issues need to be resolved. Our plan is to fully instrument the production interface for performance analysis purposes. Also, performance for a multithreaded terminal handler needs to be investigated.

Implications for Development. A set of guidelines and a standard protocol supported by a virtual workstation model alone will not be sufficient to achieve a Single System Image; we will need to develop a set of tools for the interface developer which will permit the rapid prototyping and implementation of the user interface in a pbTA-compatible, hardware-independent way.

The user needs to have some control over aspects of the "look and feel," such as color, window style, error messages, menus, and so forth, to increase personal satisfaction with the system and enhance individual productivity. Therefore, tools for users such as built-in macros which allow the user to define some of the behaviors of the system are important as well.
4. ACKNOWLEDGEMENTS

I would like to thank Ray Straka, Jack Stromberg, and Fran Hildebrand for encouraging me to develop and refine the Technology Architecture and to build the User Interface prototype. I would also like to express my thanks to Dave Turner, Chuck Clenfield, Lee Heslop, and particularly Irving Greisman, for a job well done and for sharing my enthusiasm for the project, building it, and getting it all to fly.

5. REFERENCES