Journal of International Information Management

Volume 4 | Issue 1 Article 3

1995

MOSAIC: A dynamic menu interface for end users and system administrators

Jay M. Lightfoot
University of Northern Colorado

Follow this and additional works at: https://scholarworks.lib.csusb.edu/jiim

 Part of the Management Information Systems Commons

Recommended Citation

Lightfoot, Jay M. (1995) "MOSAIC: A dynamic menu interface for end users and system administrators," *Journal of International Information Management*: Vol. 4 : Iss. 1 , Article 3.
Available at: https://scholarworks.lib.csusb.edu/jiim/vol4/iss1/3

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in Journal of International Information Management by an authorized editor of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

MOSAIC: A dynamic menu interface for end users and system administrators

Jay M. Lightfoot
University of Northern Colorado

ABSTRACT

The environment for corporate computing has become much more complex than the average end user is willing to comprehend. A solution to this problem is the development of menu shells and user-interface management systems. This paper describes the design and use of a menu shell called Mosaic. In addition to the traditional shell function of running user programs, Mosaic provides security, improves data integrity, furnishes debugging and data usage support, and aids the system administrator in day-to-day tasks. Mosaic was used successfully at a medium-sized electronics corporation for two years.

INTRODUCTION

The environment for business computing has grown far more complex than the average end user is willing to comprehend. As an example, compare the microcomputer operating system of today with those of 10 years ago. Today's end user need be concerned with memory management, swap space, terminate and stay resident programs, esoteric device drivers, and disk compression to name a few. Add to this the confusion inherent in telecommunication. Multiple nodes connected to different operating systems across vast expanses of cyberspace confuse even the most diligent power user.

Those attempting to solve this problem have concentrated on improving the user interface. The idea is to make accessing the computer (and the data it oversees) more natural and less intimidating. The trend toward menu shells and graphical user interfaces are the direct result of these efforts. While these interfaces normally achieve the base-level goal of making data and programs more accessible, they typically fall short of the potential power of the interface. Menu shell interfaces on a network are capable of administering user accounts, providing security, furnishing program and data usage information, and giving debug support (Day, 1992).

This paper describes the design and use of a menu shell called 'Mosaic'. The shell was developed and used in a medium-sized electronics corporation for two years. Mosaic goes beyond merely accessing data and programs and extends the interface toward its full potential. The shell was very successful and points out a specific instance where the payoff from end user computing is better productivity.

Published by CSUSB ScholarWorks, 1995

MOSAIC OVERVIEW

Design Philosophy

The Mosaic shell[1] was developed to resolve the problems experienced by the financial users of an electronics corporation. Before the system was developed, each end user was responsible for knowing where their programs resided, which data files were required, and the order in which the programs were executed. Security in this environment was practically nonexistent. If errors occurred during execution, systems personnel were forced to depend upon information and observations provided by the end user to track down the problem. This was non-productive and frustrating for both end users and systems personnel.

The developers of Mosaic were familiar with the limitations of existing commercial menu shells, so they decided to build their own. COBOL was used to write the system because it was the only major language supported by the data processing department. As it turned out, using COBOL was beneficiary because the language has strong support for indexed-sequential file organizations and dynamic file access. Both were used extensively in the shell.

Mosaic was developed with a basic philosophy that can best be described by four design objectives. The first objective was to design a system that is easy to use, powerful, and robust. Most commercial menu products are designed for the casual user who logs in occasionally; not for the person who spends most of the day on-line. Specifically, these systems provide a rigid menu tree that must be navigated in a hierarchical fashion. The Mosaic design team decided to allow end users to dynamically define their own menu paths to improve productivity. In addition, the menu shell includes copious help facilities. At any point, a user can type "help" and obtain context dependent instructions. Finally, the system is designed to be *bullet proof.* The end user never sees the system prompt.

The second design objective was to build a shell that facilitates system administration and provides security. Most commercial shells are designed to produce a layer of abstraction between the end user and the computer. If successful, the end user is not required to know the operating system or the location of programs and data. While this is an important feature, a shell should also supply flexible tools for the system administrator to control data and program access. This is achieved in Mosaic by a password module that allows the system administrator to define accounts and security profiles for individual users, groups of users, and public access. The security profile can control very fine granularity access (e.g., controlled access to individual programs for individual users on specific terminals). The shell also keeps a full log of each users' activity. This feature, when combined with appropriately written programs, can collect data concerning the specific field values that were modified during the execution of a program.

[1]The system was called *Mosaic* because it brought together diverse, disjointed programs into a single composite picture. It is in no way related to any commercially available software of the same name.

The third design goal was that the shell should aid systems personnel in debugging problems that occur when end users execute programs. Nothing in data processing is more frustrating than a sporadic, seemingly random bug. The only way to fix this type of problem is to meticulously reproduce the situation in which the bug occurred. Unfortunately, end users can seldom recall every detail of an extended work session, so this information is not available. Mosaic solves this problem by using the data stored in the security logs. Programmers can use the details captured in the log to duplicate the situation that caused the bug.

The final Mosaic design objective was that the shell should be dynamically extensible. That is, the system administrator can modify the behavior of the shell from within the shell. Commercial shell products normally are maintained by modifying files using special commands which are run at the operating system level (i.e., external to the shell). This requires the system administrator to learn the shell and a special shell maintenance language. This is unnecessary duplication. Mosaic can be maintained by selecting menu elements from within the shell. This ensures simple, consistent shell maintenance and gives Mosaic the desirable characteristics found in a user-interface management system (Linton, Vlissides, & Calder, 1989).

Results of Using Mosaic

The Mosaic shell was used by the financial and human resource functions of the corporation for over two years. During that time, it dramatically improved end user productivity. Users were able to concentrate on the task for which they were hired rather than on the eccentricities of the operating system and the network. In addition, new employees were productive much sooner and existing employees were automatically cross-trained due to the simple, consistent interface.

Mosaic made system administration and program maintenance much easier. Since changes could be globally implemented by a single modification, the system administrator was able to effect changes in minutes instead of days. Programmers benefited because they were better able to duplicate the conditions that caused program errors. Finally, overall system security and integrity were improved because end users were allowed to access only the programs and data for which they were authorized.

IMPLEMENTATION DETAILS

The End User Interface

When the end user first logs into the Mosaic shell, he is presented with a standard password screen that asks for the username and password (without screen echo).[2] If help is desired, he may also type "help." If he does not successfully enter the password information in three attempts, the user account is deactivated for ten minutes and an entry is made in the security log concerning the time and terminal location of the failure.

[2]The personal pronoun "he" will be used throughout this paper to represent both male and female users. This is done to avoid the grammatically incorrect use of 'they' and to reduce the confusion caused by switching pronoun gender.

Once the user has successfully entered password information, he is presented with a text-based menu of valid system options. An example user menu is shown in Figure 1 below. The options presented vary depending upon the contents of the user security profile. Users with high security clearance have more options than those with low clearance. A major design characteristic of Mosaic is that users are shown only functions for which they have permission to execute. This differs from many commercial shell programs that *gray out* or disable functions that users may not access. This improves overall system security since the end user does not know what he is not allowed to do.

Figure 1. Mosaic Payroll System Menu

```
PAYROLL SYSTEM MENU

TC    Timecard Menu
TAX   Tax Reports Menu
MAN   Create Manual Checks
IN    Input New Employees
CAL   Calculate Payroll
MNT   Maintain Payroll File
PRT   Print Payroll Checks
RES   Restore Old Files
BAK   Backup Payroll Files
UTL   System Utilities

Enter selection, route name, help, exit,
or <return> _____
```

The prompt for the menu screen is automatically placed at the bottom of the screen. This prompt asks the user to enter his menu selection, a route name, help, or exit. Each of these options are described below.

- The **menu selection** is a three character abbreviation of a specific action or sub-menu. The menu selection is described in the body of the menu screen. If the user selects an action (i.e., executable program or command procedure) then the program is executed and control returns to the menu. If a sub-menu is selected, the appropriate lower-level menu is displayed. When the return key is pressed without making a selection, control pops up to the next menu level.

- The **route name** option allows the user to execute a predefined menu path. These paths allow the user to bypass the traditional hierarchical menu structure. Each route name is unique for the user that creates it. This means that many users may create different routes with the same name without conflict. The route function is a useful productivity tool for the end user who performs routine tasks often and does not wish to chain through a menu hierarchy. Instead, he can jump directly to the desired action. Routes are created by executing a special utility function that memorizes keystrokes in a *learn* mode. Any number of routes may be defined.
- **Help** shows a help screen that describes the functions available.
- The **exit** function allows the user to exit the shell (logoff) from any point within the menu tree. This is another feature to improve user productivity. At no time is the end user allowed to see the system prompt. Confirmation is required before the exit is performed.

As long as the user works consistently at the terminal, he is not required to re-enter a password. Anything he is allowed to do within the system is 'pre-approved'. If the terminal is idle for more than ten minutes then password entry is required for any action except the exit function. This provides security when a user leaves his desk without logging off.

INTERNAL SYSTEM DETAILS

The key to a successful shell program is a simple, flexible user interface. The previous section adequately describes the Mosaic system from the standpoint of the end user. The system is powerful, easy to learn, and versatile. As might be expected, many things must take place behind the scenes to make this possible. This section describes the major low-level implementation details of the Mosaic system.

System Environment

Mosaic was designed to run on a Digital VAX 11/780 computer using the VMS 5.0 operating system; consequently, certain characteristics are unique to this environment. First, VMS allows long filenames (up to 64 characters at the installation where Mosaic was written). Second, the VMS environment is capable of running command procedures written by other command procedures. Thus, a hierarchy of executable procedures can be dynamically generated and performed directly from the operating system. Finally, VMS 5.0 is a character based operating system (i.e., no Windows icons and no mouse capabilities). Because of this, all screen displays are relatively simple ASCII characters. Modern computers manufactured by Digital continue to support the VMS operating system, so Mosaic does not require an antique computer.

End User Profile

Every valid Mosaic user must have a user profile before he is allowed to login. The user profile is defined by the system administrator via a utility program that is only available from

Published by CSUSB ScholarWorks, 1995

the administrator's menu. The profile is stored in an indexed-sequential file which contains the user name, password, valid terminals for access (is applicable), valid systems for access, and protection level for each system. These elements are described below.

- The **username** is the primary key for the file and can be up to twenty characters long. Traditionally, it is the end users last name, but can be any meaningful combination of characters that denote individual users or groups of users.
- The **password** holds the user passwords required for login. They are initially set to the username by the system. Once access is gained, the user may set the password to any string (up to 20 characters).
- **Valid terminals** allows the system administrator to specify the terminal identification number(s) from which the user is allowed to access the system. Up to five terminals can be specified, or the word "ALL" can be entered to denote any terminal. If the computer installation uses a terminal server, then the valid terminal option is disabled. This is necessary because terminal servers assign a new logical terminal number for each login session. There is not a consistent physical connection established to each terminal.
- The **valid systems** array is very important. This entry names the three-letter abbreviation of the system(s) that the individual user may access. Each system has a unique abbreviation that identifies it throughout Mosaic. For example, GL could stand for general ledger, AP for accounts payable, PR for payroll, etc. If a system is not named in the valid systems array, the user is not allowed to see or access the sub-menu associated with that system. Up to 99 systems can be assigned to each user. The system administrator has a special code of "SYS" to denote any system currently defined. In this way the administrator always has complete access to all systems.
- Finally, the **protection level** is an array that associates the level of access allowed for all valid systems (up to 99). One protection level is associated with each system. The level ranges from 0 (no access) to 99 (full access). This works in concert with the leaf-level menu actions (i.e., programs and command procedures at the end of a menu tree). Each menu action is assigned a protection level. For a user to be able to see and execute an action he must have a system protection level equal to or greater than the menu action level. If a user has 0 level access, he may see the menu title, but nothing more. When used with the "SYS" administrator code, different levels of system administrator may be defined. A 99 level administrator is traditionally the database administrator.

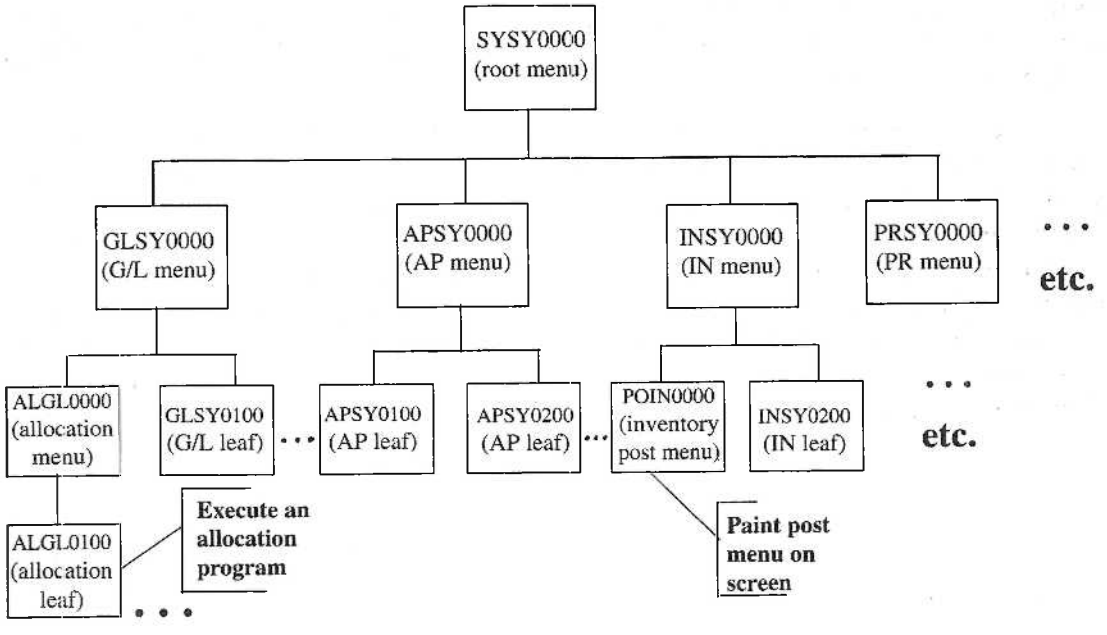
The login process takes the username and password provided by the user and attempts to read the user profile. If a profile does not exist then the login attempt is unsuccessful. A successful login attempt generates several internal actions. First, a unique timestamp is created that combines the username, terminal identification, and login time. This timestamp is used to uniquely identify all actions that occur during the specific work session. Next, a file containing a single record is written (using the timestamp as a name) that contains the system profile information. This file is checked each time the user attempts an action during the session. Finally, a sequential history file is opened to record all activities that occur during the work session. Every

user has his own history file and all new entries are appended to the end of this file. Thus, a complete history of everything the user does is recorded in the file. This will be discussed below in depth.

Menu Structure

The static menu structure is stored in an indexed-sequential file with a key that combines the system code with the parent menu code and a four-digit menu identifier. Every action in a menu has a different four-digit menu identifier. This generates a unique key that can be used to represent a hierarchy of menus and actions. For example, "ALGL0100" is the 0100 action of the allocation sub-menu under the general ledger system. Figure 2 shows a diagram of a sample static menu tree. Notice how the nodes in the diagram can either be menu nodes (identifier 0000) or leaf nodes (any other four-digit identifier). Leaf nodes are associated with command procedures of the same name. Thus, the name of the leaf node is also the name of the procedure to execute to perform the leaf action. Stored in the record for each action are a protection level and a valid terminal array. These are used to compare with the individual's user profile. A successful match allows the end user to execute the action represented by the entry. An unsuccessful match generates an error.

Figure 2. Sample Static Menu Tree



Published by CSUSB ScholarWorks, 1995

The menu structure is called *static* because it is merely a template to be used by individual work sessions. Each session reads the information stored in the menu file to generate a dynamic menu for that session.[3] This design was adopted to allow different procedures to be generated for different levels of users and to avoid the problem of record locking when many users attempt to access a single file. It also makes the implementation of user defined menu routes very simple. Another benefit is that disk space is not wasted by storing the full menu tree; only the tree template and the generating procedures need be stored permanently.

When a menu node is selected by the user, the nodes directly below the menu node are read and the screen is painted with a menu of those lower level actions that the user is allowed to execute. When a leaf-level action is selected, control is passed to the procedure with the same name as the node. This procedure subsequently executes some program or performs some action. When the procedure completes, control returns to the calling procedure. As the user backs out of the dynamic menu hierarchy, the lower levels of menu are automatically deleted by Mosaic.

Dynamic Extensibility

All menu shells require some way for the system administrator to modify or update the shell interface. For example, new users need to be added, programs are modified, and security levels change. Most commercial shells require that this be done externally to the shell using a special shell language or initialization file. Further, in almost all cases, these changes cannot be made while active users are in the system. The Mosaic development team felt that this was overly restrictive; consequently, all shell modifications in Mosaic can be made from within the system while the system is active.

Users with 99 level system privilege have access to the administrative utilities menu. This menu contains programs that maintain the user profiles, edit the leaf-level procedures, edit, purge, and print history files, backup and restore data files, and maintain the static menu file. The advantage of this design is that the system administrator can make changes from within the system and all changes are effective immediately. For example, if a new program is released, the static menu file can be modified in real time by making one change via the utilities menu (while users are active). As soon as users access the new menu screen, the changes are in place. The same holds true for security and password modifications. On-line, real-time maintenance is one of the most powerful features of Mosaic.

[3]Recall that VMS allows command procedures to generate other procedures that can subsequently be executed. This capability is used to build the dynamic menu structure "on the fly."

History File

New entries are added to the end of the user history file every time a dynamic menu structure is generated or actions are performed. The history file stores the session timestamp, the start and end time of the action, and a text description of what action was performed. Also stored in the history file are route invocations, menu navigation actions, final logout, and invalid login attempts. Since the history file is sequential, it can be accessed using any text editor (for debugging purposes) or analyzed by an external database package to determine data usage patterns.

An optional feature of the history file is the ability to store details of what the user does within each executable program. For example, the payroll maintenance program could be written to add entries to the appropriate history file that describe every record the user accesses and the before and after images of any fields that are modified. The end result would be a complete audit trail of every action impacting the payroll file.[4]

DISCUSSION OF THE APPROACH

Menu shells in general and the Mosaic approach in particular provide several advantages. First, shells create a level of data and program abstraction. This means that end users do not need to know where their programs or data are stored. In addition, since the command procedures that run the programs can perform logical assignments, sorts, and other housekeeping functions, users are freed from knowing the tedious technical details needed for program execution. Second, the Mosaic password module allows users to access only the programs and data that they require and users are never allowed to directly access the operating system. Thus, the shell enhances overall system security and integrity. Third, programmer responsiveness is improved because the history files capture the exact chain of events that lead to an execution error. This allows the programmer to recreate quickly the problem that caused the error. Finally, Mosaic makes system administration easier because the shell is dynamically extensible while users are active in the system.

On the negative side, Mosaic is currently written to work only with the VMS operating system. This limits its usefulness. Also, since the operating system is character based, Mosaic does not use a graphical desktop metaphor or the mouse. Finally, the shell is not designed to work in a client-server environment. The system assumes a traditional mainframe with attached dumb terminals. Despite these problems, the overall effect of Mosaic on the electronics corporation where it was developed was very positive.

[4]This requires that the executable programs be written to manipulate the history file independent from the menu shell. This feature was fully implemented, but never used by the electronics firm upon which this paper is based. It was decided that this level of security was not necessary.

Published by CSUSB ScholarWorks, 1995

FUTURE ENHANCEMENTS TO MOSAIC

The key contribution of the Mosaic shell is its basic design philosophy. The system solved a complex problem in a very elegant manner. The same philosophy can be applied to a modern computer environment. To begin, Mosaic should be modified to use a mouse and a graphical interface. This would improve end user productivity even more. Next, the original system assumed the COBOL and the VMS operating system performed all file input/output. This is a limitation. The system should be designed to work on a client server network with different database backends. Lastly, the shell should have built-in tools to analyze the history logs. Long history logs are difficult to use without adequate processing support.

SUMMARY

This paper discussed the development and use of a menu shell called Mosaic. Mosaic was designed to provide a powerful, flexible menu shell interface for the final users of an electronics corporation. The shell helped improve the productivity of end users, programmers, and the system administrator in the company. It did this by creating a level of data and program abstraction that allowed end users to concentrate more on their work and less on data processing details. It also provided administration, security, and debugging support for the technical staff. Mosaic is a good example of how end user computing can improve overall corporate productivity.

REFERENCES

Day, M. (1992, April 20). DOS menu packages fulfill varied needs. *LAN Times, 9*(7), 41.

Linton, M, Vlissides, J. & Calder, P. (1989, February). Composing user interfaces with InterViews. *Computer, 22*, 19.