CHALLENGE TO REACH THE END USER

Late Papers

SOUTH PACIFIC AREA REGIONAL CONFERENCE

MELBOURNE, AUSTRALIA

NOVEMBER 19-22, 1984

THE VICTORIAN H.P. 3000 USERS' GROUP

SOUTH PACIFIC AREA REGIONAL CONFERENCE

19-22 November 1984

MELBOURNE

AUSTRALIA

"DFS an Alternative to VPLUS"

J. Cybulski

P. Darbyshire

R.M.I.T.

DFS an Alternative to VPLUS

Abstract

DFS (Display Facility Software) is a screen handler developed at RMIT to allow on-line forms handling on a variety of terminals.

The package offers most of the standard facilities available on other screen (or form) handling systems, but it also possesses the unique capability of defining terminal characteristics. This permits the inclusion of new types of terminals in the system configuration, without compatibility constraints, allowing choice from a wider selection of terminal makes and prices.

The benchmarks of the DFS system proved it to be an efficient, flexible and robust tool; and these qualities are assured by screen-forms compilation, data compression, cursor movement optimization and table driven logic.

Certainly DFS is a cost-effective alternative to Hewlett-Packard's VPLUS which offers huge operational capabilities but which also ties users to the expensive HP terminals.

Philosophy of Writing Screen Handlers

Everybody uses screen handlers. Screen handlers are an important tool in writing fashionable software. User-psychology has been put on a pedestal in current Computing and therefore new techniques and methodologies in man-machine interfacing have appeared on the lucrative software market: natural language interfaces, voice analysers and synthesizers, graphics input and output devices like a light-pen, a mouse, digitising tablets, high resolution terminals with sophisticated software in form of windows, spreadsheets, forms, gadgets, knick-knacks, gadgets, gadgets. We call it computer education of the computer user, and we have tought him or her to demand high standards and to feel free and comfortable with computers.

Wherever you turn, you see the glamour of available software tools, shiny and polished packages, intelligent peripherals, all of which you would love to have in your computer business, all of which you would buy immediately if you could disregard their cost and your company's emptying pockets, if you could forget about the software conversions and hardware upgrades which are inevitable to support those little wonders.

On the other hand you know that the software market is a harsh battlefield, which offers but also demands high quality. You are already in a catch-22: you must produce those "user-friendly" toys to get a high profit, but to do so you must also make an investment in appropriate, usually expensive tools.

Ok, you cannot afford the fantastic Artificial Intelligence "Expert Systems" which would do whatever you request, neither you can buy the

wonderful fully integrated software development tools (same reason). You cannot even reach for those excellent, cheap and powerful form handlers which would give your software a modern look, because in conjunction with them you must also purchase the most intelligent and expensive computer terminals. You don't have too much choice, so you are forced to write all the necessary non-deliverable software including your own screen handler.

You look around to see what equipment you have, what terminals are available, what the user would want, what programmers demand; and you write one. You can see immediate advantages: you don't spend too much money, the screen handler works, it gives a pleasing look to your products, and you can also modify it whenever you want to - it's your piece of code! And you do change it, actually quite a lot - when you get a new terminal to support, or when the user does not like some of the form features, or when the new edition of the operating system is introduced, and soon an update is coming again. The system mushrooms, and then the worst happens - the irreplaceable super-programmer who wrote the screen handler retires and nobody else can modify your package anymore. I tell you what - you blew it...

Everybody uses screen handlers and everybody writes them.

Methodology of Writing Screen Handlers

So we realize that a screen handler is an ordinary software product and ordinary software engineering methodologies should apply to it - only then can it be cost-effective. The screen handler **must** then be user friendly (so that the user's psychological needs are satisfied), it should be flexible which manifests itself in peripheral independence (so that a wide range of terminals could be used with it), host independence (so that the product can be marketed for other types of computer installations) and user community independence (so that it could be optioned for different types of users). Extendability, portability and efficiency are other aspects of software production which are applicable to screen handlers as well.

User friendliness of software has two different aspects: ease-of-use by end-users and ease-of-use by programmers. The better the package looks the better your bank account feels; the longer it takes to implement, the higher development costs are. Both aspects are important. Let us consider the screen handler features which provide the maximum aesthetics for the user and the minimum cost to a software developer (● - indicates all features implemented in DFS, □ - indicates extras which have not been addressed by DFS):

- displaying forms on the screen,
  any form consists of different types of information,
  all types should be clearly distinguished on the screen:
  - ●● background, invariant information
  - ●● protected but variant fields
  - ●● unprotected, user-changeable fields
- full-screen editing of unprotected information

This is a bare minimum that we can get from any of screen handlers. Clearly we can also have super-duper facilities which are not provided by many available screen handling tools, for instance (● - implemented in DFS, □ - not implemented in DFS):

- □ multiple forms on the screen (or windows)
- □ scrolling forms (or spreadsheets)
- ● mixed block and text modes of processing on one screen
- □ function keys
- ● etc...

From programmers' point of view the following utilities are of a great value (● - DFS features, □ - not implemented in DFS):

- ● remote form design (terminal independent, eg.: using an editor)
- ● full-screen form design (terminal dependent)
- ● form testing facility (form display, fields input and output)
- ● form compilation for a particular terminal (for efficiency)
- ● library procedures for programmable form handling
- ● libraries of forms (source and compiled versions)
- ● setting terminal characteristics (for terminal independence)
- ● form handling
  - ●● basic form handling like displaying, filling with data ...
  - □□ accessing individual fields
  - □□ form and field language (eg.: assertions, types, etc...)
  - □□ dynamic changing of field/form attributes
- □ programming function keys
- □ specialized form manipulation languages
- □ other super-extras like mouse interfaces, etc...

After a closer look at these characteristics and the facilities offered by the peripheral equipment available on the market we must come to the conclusion that some of the goals of user/programmer friendliness are

incompatible with other goals of software engineering, eg.: hardware independence. For example windows and spreadsheets would be possible only on graphics terminals. Obviously there are always attempts to do the impossible, as in some instances of the software on the market, but this leads to lengthy screen redrawing, very slow operation and therefore to irritated users.

Thus instead of simulating the features we do not have, let us design a package which will offer facilities achieveable on currently available equipment, let us make it as attractive as it is possible. We will then gain the user's (facilities and cost) and the programmer's (tools) acceptance without loosing efficiency. Such a package - DFS (Display Facility Software) is available and has been developed at the Royal Melbourne Institute of Technology. DFS offers a wide range of desirable facilities (see all • features mentioned above, see also figure 1), but it also excludes some features that are found in other screen/form handlers (see all □ features mentioned above); this situation has been partly explained here.

DFS is written in standard ANSI-PASCAL using the modular software development methodologies and therefore can be easily ported to other non-HP computer installations; furthermore the modular structure of the package permits easy extensions later on; and it is a table-driven software, thus enabling easy adjustment of its features without modifying the actual code. This approach is especially useful in setting the terminal characteristics tables, so that new terminal types can be easily added to the system. Practical experience with the package shows that the DFS's features make it a really flexible product (terminal, host, user

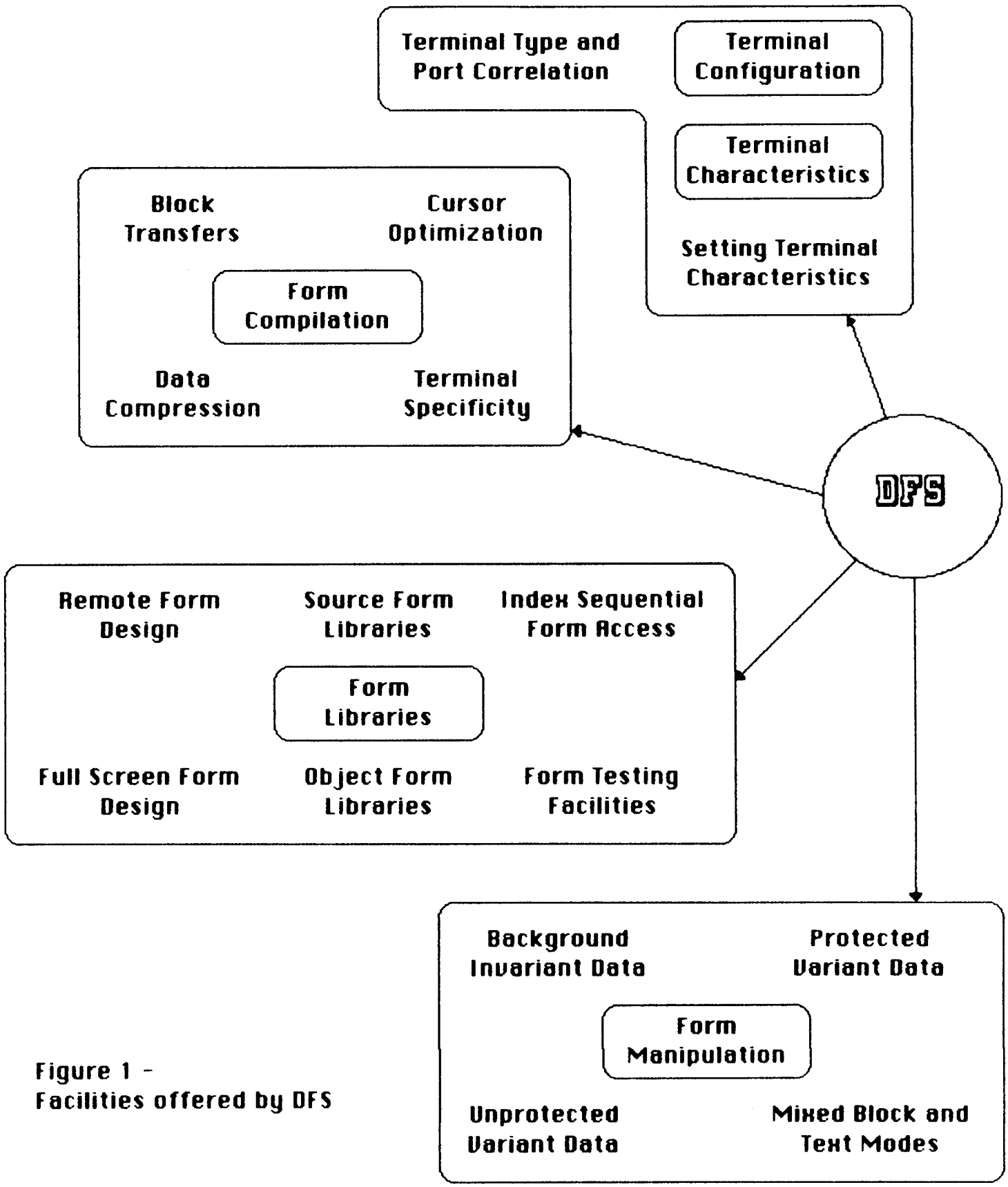


Figure 1 -
Facilities offered by DFS

independence).

Achieving user/programmer friendliness and software flexibility usually leads to a loss of operational efficiency. This trade-off has been considered during the package design, and a considerable effort has been put into avoiding software inferiority in speed and performance. Special optimization techniques have been applied to eliminate all possible deficiencies. Hence:

a. form pre-compilation is used to avoid terminal-specific computation during the application run;

b. the use of block I/O wherever possible leads to I/O time optimization;

c. index-sequential form library organization speeds up a precompiled form access;

d. data compression during the form compilation process minimizes later form transmission time;

e. cursor movement optimization techniques are used to shorten the local-terminal operation time.

The discussions so far indicate that DFS achieves all those software engineering goals (user/programmer friendliness, flexibility, peripheral independence, host independence, user community independence, extendability, portability and efficiency at the same time) which cannot be achieved by many other screen/form handlers, for example VPLUS. Also all the benchmarks show DFS is more efficient than another RMIT package it has superseded - DEL.

The Architecture of Display Facility Software (DFS)

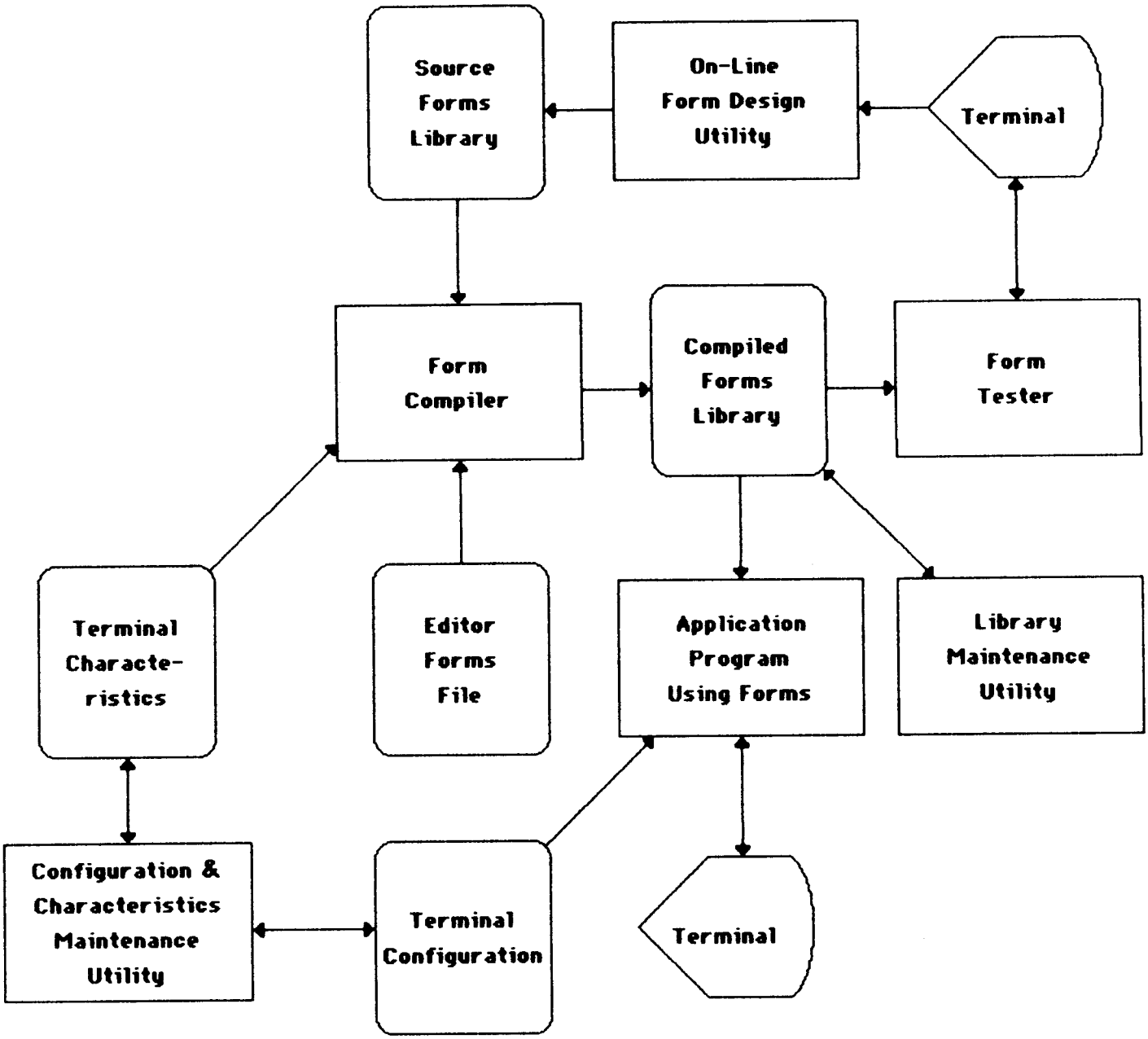
Its architecture is shown in figure 2. It consists of six subsystems:

- Form Designing Utility.
  It allows full-screen design of forms and enables maintaining terminal independent, source form libraries. This utility is combined with the Form Compiler and Tester creating a fully interactive form-design environment. It must also be mentioned that an alternative way to designing forms is with the use of editors.

- Form Compilation Utility.
  The utility is integrated with the Form Tester. It enables compilation of selected forms for use with specific types of terminals. The compiled forms utilize the full capabilities of these terminals and enable cursor-movement optimization and data compression, which significantly shorten the form manipulation and transmission times.

- Form Testing Utilities.
  This utility is especially useful for testing forms designed with the help of an editor in which case the viewing of designed forms would not be otherwise possible. The form tester is also used to test correctness of terminal characteristics set-up.

- Form Manipulation Routines.
  This is a set of subroutines to be used with application programs which require screen handling. The subroutines enable displaying forms on the screen, filling them with data (protected and unprotected fields), clearing forms, and all the other necessary operations (for details see the DFS user guide). The set of operations available in this object-library is fully compatible with another, older version of a screen handler developed at RMIT - DEL.



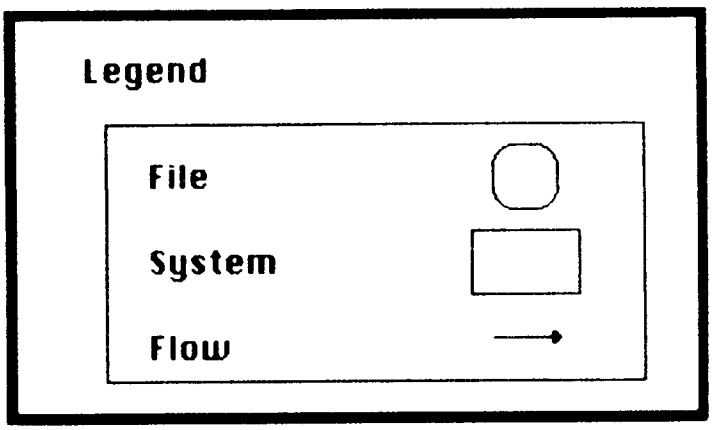


Figure 2 -
DFS System Architecture

- Compiled Form Library Utility.
  This subsystem enables easy maintenance of form libraries. Addition, deletion, copying and listing the compiled forms gathered in any particular form library is possible. There is the capability of creating different libraries for different groups of users, which permits viewing the same form in a variety of ways depending on user requirements (eg.: a group of RMIT users refuses to accept any forms with inverse video fields, whereas the programmers group demands such a facility on display arguing that this enables a better identification of the form features).

- Terminal Characteristics Utility &
  Terminal Configuration Utility
  These two utilities (integrated) maintain the heart of the DFS system. The first enables entering descriptions of new terminal types installed on the system, the second permits identifying the types of terminals connected to different system ports. The terminal characteristics table (of type descriptions) is used primarily in the form compilation for a particular type of terminals, the configuration table is used by form-utilizing routines to find out which of the binary (compiled) forms should be selected from the form libraries depending on the type of currently used terminal (port indicates the type).

We can see that all the utilities offered by the DFS package constitute a powerful screen (form?) handling system.