Implementing Structured Design of Menu-Driven Systems with SAS/AF Software and the Macro Facility

John M. Rinehart
RMC Environmental Services

Abstract

Structured system analysis and design techniques have come to be commonly used in "traditional" data processing environments, with the goal of producing information systems characterized by modularity, top-down structure, clear functionality, and maintainability. The increasing use of "fourth generation languages," with their reduced coding, simplified file management, and quick prototyping, has tempted some system designers to side-step these recently learned and proven techniques and implement systems "on the fly." For Shame.

Structured analysis uses the Data Flow Diagram as a tool to describe the logical "what" of an information system. Several past SUGI presenters have noted the direct correspondence of SAS * Software datasets and DATA/PROC job steps to elements of the DFD. Structured design goes on to specify "how" the information system will accomplish its appointed tasks. Again, others have identified the ability of the macro facility to deliver the hierarchical function modules of a system structure chart.

This paper discusses the use of structure charts and structured design of interactive or "menu-driven" systems, wherein the design allows each incarnation of the system to be determined by the responses of the user. SAS/AF Software and the macro facility are shown to be complementary tools which are particularly well suited to building such systems under a structured design. Implementation techniques, tips, and pitfalls are presented.

Philosophy

In the early 1970's, about the same time that SAS* Software was first becoming available as a commercial product, some new ideas and new tools were being developed to improve the craft of the information systems designer. These ideas have now essentially matured into the disciplines known as structured systems analysis and design. In the same period of time, SAS Software has evolved and claimed a place among the "fourth-generation" of computer application development tools, database management systems have all gone "relational," and the "prototyping" of new "menu-driven" systems has come to be expected. In fact, structured analysis and design techniques have already been adopted by a broad variety of industries, and new "menu-driven" systems are increasingly common in the marketplace. Structured design goes on to specify "how" the information system will accomplish its tasks by using a Structure Chart. These modules are arranged in a hierarchy of control, each performing some task for its superior, and each in turn calling subordinate modules to perform more detailed tasks. Moreover, the information interfaces, or coupling, between modules will be completely specified and kept to a bare minimum.

The primary tools used in achieving these goals are rigorous graphic techniques which allow the designer to clearly and completely describe the function of the system. For example, in the systems analysis phase, the Data Flow Diagram is used to define "what" happens in the system, from the point of view of the data moving though it. Each arrow represents a type of data, and each circle represents a system process by which data are transformed from one type (or types) to another type (or types).

ANALYSIS (What): DATA FLOW DIAGRAMS

Similarly, the system design phase defines "how" the system will accomplish its tasks by using a Structure Chart. The data transforming processes of the data flow diagram are grouped into functional modules, represented by boxes on the structure chart. These modules are arranged in a control

1400
As each called hienu-chy, so that each module can call functions of modules beneath it, as represented by connecting lines. As each called module completes its task, it returns control to its "superior." In this way, each module needs to "know" only about its job and those beneath it. It need not know or care what its superior has to do next. The information interface, or coupling, between modules must be clearly specified. Small arrows in the structure chart define what data is needed as input for each module, as well as what data is generated or returned.

The wishes of the user are determined at appropriate points by system control modules which function as generic menus, whether they are implemented as pull-down windows, check-off lists, or simple enter-your-choice prompts. The menu module will then conditionally call some subordinate system function, according to the user's choice. Finally, when that subordinate function has completed its work and returned control to the menu module, the process will be iterated to see what to do next. A structure chart for such a system might look like this:

**Design (How): Structure Charts**

```
As a brief example, the functions outlined in this structure chart might be implemented in SAS like this:

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
A PROGRAM screen menu will allow the display of variable system information to the user, as well as provide the opportunity for obtaining secondary user responses in addition to the primary menu choice. For example, a "check off box" might be provided for the user to request "novice" or "expert" versions of the menu function choices. Additionally, if PROGRAM screens are used for obtaining user responses in other, non-menu functions of the system, then using PROGRAM screens also for the menus will provide a consistency of PFKey control throughout the system. PROGRAM screens use the END PFKey to continue; MENU screens use END to go back. Finally, use of the PROGRAM screen for system control modules give the designer a great flexibility in the calling of subordinate modules. Macro calls can execute macro-based modules, other PROGRAM screens may be called with PROC DISPLAY, or other SAS procedures may be called directly. Multiple subordinates may be called in succession, if so indicated by the design, and all subordinate calling can be controlled by the conditional logic provided by the PROGRAM screen and the macro facility. In contrast, MENU screens can only branch to another AF screen.

Pitfalls

I have identified (by experience) several potential pitfalls associated with the implementation of structured menu modules. The most serious problem has to do with the return of system control to a superior, or calling, module when the subordinate, or called, module has completed its assigned duties. The SAS/AF product documentation would have us use a MENU screen to conditionally call other AF screens, which would each re-call the original MENU as a last step:

```plaintext
PROC DISPLAY C=MENU1.MENU;
```

branches to:

```plaintext
(FUNCTION1.PROGRAM)
```

```plaintext
DATA . . .
```

```plaintext
PROC . . .
```

```plaintext
PROC DISPLAY C=MENU1.MENU;
```

The intent here is to create an iteration of the original menu module, according to our structured design:

**WHAT WE WANT:**

![Diagram](image)

Unfortunately, in a SAS implementation, the original menu module is now long gone, and what we actually get is an open-ended recursion which can go on indefinitely, is impossible to completely specify in the design, and can be the cause of numerous irritating bugs.

**WHAT WE GET:**

![Diagram](image)

For example, if FUNCTION 1 does not execute to completion (as it may well not if it is a PROGRAM screen and the user issues the CANCEL command), the re-call of MENU1 will never occur, and the user will suddenly be looking at the SAS or operating system prompt with a puzzled expression.

**PRE-MATURE EXITS:**

![Diagram](image)

As another example, the ability to re-use certain modules of particularly useful function in several places of the design is prohibited. We may want FUNCTION 2 to unconditionally execute FUNCTION 1 followed by FUNCTION 3. But FUNCTION 1 wants to re-call MENU1, and recursion problems of major proportion result.

**LACK OF RE-USE FLEXIBILITY:**

![Diagram](image)
This leads to my first rule of structured implementation, and its corollary:

**RULE 1:** NEVER SPECIFY A RETURN PATH IN A CALLED MODULE.

**COROLLARY 1.1:** THE CALLING MODULE IS ALWAYS RESPONSIBLE FOR THE NEXT FUNCTION (INCLUDING ITERATION).

Or stated yet another way, a subordinate module must never presume to know what its superior wants to do next. It should just quietly do its job and let its superior take the next step.

How do we implement this rule? If the calling module is a macro, the return of control is automatic, since the calling macro's referencing environment remains active until all macro statements have been processed. If the calling module is a SAS/AF MENU screen with its single branch capability, we are out of luck, and thus my suggestion that they not be used. On the other hand, when calling subordinates from SAS/AF PROGRAM screens, we can take advantage of SAS input stack, and the fact that all SAS code generated by a PROGRAM screen will remain on the stack and eventually be executed. When execution of a called subordinate module is complete, control will return to the next statement which was originally submitted by the calling module. Thus, a PROGRAM screen should generate SAS code to perform all the required subordinate functions, including self-iteration as the last instruction:

```sas
(MENU1.PROGRAM)
PROC DISPLAY C=FUNCTION1.PROGRAM; RUN;
%FUNCTION2;
PROC DISPLAY C=MENU1.PROGRAM; RUN;
```

This is still not a true iteration of the MENU 1 module, but a re-calling of it. Nevertheless, the fact that it is a self-call guarantees re-execution of the menu, independent of any subordinate actions (short of some fatal error). Recursion of the menu module will not be a problem, and is functionally equivalent to iteration, as long as the self-call is the very last instruction given.

**RULE 2:** SELF-ITERATION MUST BE THE VERY LAST INSTRUCTION IN A PROGRAM SCREEN MODULE.

The next temptation is to impose conditional calling of subordinate modules on this sample PROGRAM screen by making the whole thing an instream macro, followed immediately by the macro call:

```sas
(MENU1.PROGRAM)
%MACRO MENU1;
%IF &CHOICE=1 THEN %DO;
PROC DISPLAY C=FUNCTION1.PROGRAM;
%END;
%IF &CHOICE=2 THEN %DO;
%FUNCTION2;
%END;
...
PROC DISPLAY C=MENU1.PROGRAM; RUN;
%MEND MENU1;
%MENUX
```

The problem here is recursion of a different kind. The %MENU1 macro is defined in code generated by MENU1.PROGRAM, and yet the macro itself calls MENU1.PROGRAM. During the first execution of %MENU1 an attempt will be made to re-define the executing macro, and the system will stop noisily. In fact, care must be taken to avoid any possibility of code generated by an instream macro coming back around to the point of the macro definition. Thus to be safe:

**RULE 3:** INSTREAM MACROS SHOULD NOT CONTAIN ANY CALLS TO SUBORDINATE MODULES.

Recommendations

Finally, then, I come to my recommended model for using the SAS/AF PROGRAM screen as a system control menu module. This is a generic example which should obviously be modified for specific use. The user field area of the PROGRAM screen might look something like this:

**PROGRAM SCREEN MENU**

Enter menu choice here: &MCHOICE and press END.

1. Function 1
2. Function 2
...
(blank) to quit

Place X here _ for instructions

Current database: DATABASE

The user's menu choice would be entered in screen field MCHOICE, which would be associated with a macro variable of the same name. Valid options may be limited with the screen field attribute list. Other screen fields / macro variables might be used to request secondary responses (X for instructions) or to display additional system status information (database name). Pressing the END PFKey would turn control over to the program portion of the screen:

```sas
(THISMENU.PROGRAM)
###MMENU;
%MACRO MMENU;
%GLOBAL MCHOICE MFUNC;
%LET MFUNC=\$NULLMAC;
%IF &MCHOICE=1 THEN
%LET MFUNC=\$FUNCTION1;
%IF &MCHOICE=2 THEN
%LET MFUNC=\$FUNCTION2; Proc
DISPLAY C=\$FUNCTION2; Proc
DISPLAY C=\$FUNCTION2; Proc
DISPLAY C=\$FUNCTION2; Proc
```

There are several points of interest in this example. First, a ### screen macro is defined and used to translate the user's choice into a subordinate function call. The global macro variable &MCHOICE is
tested and used to conditionally assign a value to a global macro variable &MFUNC. The value assigned may contain any SAS code needed to execute a subordinate function, but it should begin with a macro name, as &MFUNC will eventually be resolved as a macro call. Thus &MFUNC is initialized to a "null value" of NULLMAC, where %NULLMAC is a macro defined to the system but containing only a null statement. If the desired subordinate function is a macro module, &MFUNC is set to its name, such as FUNCTION1 for the module %FUNCTION1. Alternatively, subordinate functions may be called as direct SAS code, assigned to &MFUNC behind "NULLMAC," as in the case of the PROC DISPLAY of the next menu.

Continued execution of this module is made conditional on the user response with the &MCHOICE test. If the user leaves this screen field blank, no further SAS code will be issued; this module has completed its task, and control will return to its superior. On the other hand, when the user enters a valid response for MCHOICE, the following code will be produced. The &MCHOICE macro variable will be cleared, a subordinate function will be called through symbolic substitution of &MFUNC in the %&MFUNC statement, and the self-iteration of this menu will be requested.

Surely there are other possible implementations of menu control functions. I can recommend this one as well tested, flexible in application, and conforming to the structured design model.

Design vs. prototyping?

Having talked at length under the banner of structured design, I need to add one more philosophical observation based on experience, and relating to the debate among system developers between design and prototyping. I believe that both approaches have merit, and could be used together to exploit the strengths of each.

Especially for interactive applications prototyping may well be indispensable. Indeed, for assessing design feasibility, a prototype can reveal whether our fourth-generation language can possibly do what we desire. We can't assume that any design can be built with any software tool! Prototypes also have value in quickly and cheaply assessing design functionality and user satisfaction.

Then, once a prototype is "working," use structured design tools to document the system. If you can't do it, there are probably some major design flaws which will have to be addressed. In my experience, the prototype system "worked" just fine, but a review from a structured design angle revealed some potential disasters. This step will also help to solidify the system module boundaries and interfaces, and provide a solid foundation for the move to production status and future system maintenance. Finally, don't forget or be afraid to re-code the prototype! It was just a first approximation, anyway.

Summary

Use structured design techniques, in connection with prototyping, to produce higher quality applications. Use the SAS autocall macro facility for system modularization and interface control. Use SAS/AF for interactive screen functions, but use the PROGRAM screen for menu modules, and build on the rules and examples given in this paper.

Questions and comments on this work are invited, addressed to the author at:

RMC Environmental Services
1921 River Road, P.O.B. 10
Drumore, PA 17518
(717) 548-2121

Acknowledgements

This work was supported by Philadelphia Electric Company and Pennsylvania Power & Light Company.

SAS and SAS/AF are registered trademarks of SAS Institute, Inc.

Bibliography


