Object Oriented Programming with the SAS/AF\textsuperscript{®} FRAME Entry:
CANDA Experiences

Barry R. Cohen, Planning Data Systems

ABSTRACT

The FRAME entry enables a sophisticated graphic user interface in applications by providing graphic screen objects that are manipulated using object oriented programming techniques. The GUI is briefly defined, including explanation for why it provides a superior interactive environment. The basic concepts of OOP are then presented, including a concrete OOP example. Explanation is given for why OOP is so effective for programming interactive applications with a GUI. The SAS/AF FRAME entry is then introduced, with comments about its use in CANDA system development. More broadly, this paper is about the value of SAS/AF with the FRAME entry to development of interactive applications in general. It is an outgrowth of my experience using the SAS/AF FRAME entry for CANDA development in the SAS for Windows environment.

INTRODUCTION

The SAS\textsuperscript{®} System, with the release of version 6.08, has made a major advancement in our ability to develop interactive systems within the SAS environment. This advancement comes from the introduction of the FRAME entry in the SAS/AF product. The FRAME entry enables a sophisticated graphical user interface (GUI) in applications by providing graphical screen elements that are manipulated using object oriented programming (OOP) techniques.

The FRAME entry is important to the development of any large, highly interactive system because:

\begin{itemize}
  \item Graphics allow for the best user interface and the FRAME entry is an entirely graphical way to develop screens.
  \item OOP is an efficient programming technique, especially for systems employing a GUI, and the FRAME entry is based upon an OOP approach.
\end{itemize}

The FRAME entry is also important to interactive system development for two other reasons:

\begin{itemize}
  \item First, many interactive systems must present data that has been analyzed in SAS data sets and the FRAME entry is a SAS/AF product. Thus, SAS/AF with the FRAME entry provides the important benefit of developing an interactive system entirely within the same environment where your data is.
  \item Second, large interactive systems require prototyping. The FRAME entry supports rapid and efficient prototyping of any interactive system.
\end{itemize}

This paper is an outgrowth of my particular experiences applying SAS/AF and the FRAME entry to CANDA development projects in the SAS for Windows\textsuperscript{™} environment. But a CANDA is only one example of large, highly interactive systems. So, more broadly, the subject of this paper is information for those who plan to develop any interactive system, who want to remain SAS-based for such a system, and who want the advantages of object oriented programming and a graphical user interface in their work.

This paper will discuss:

\begin{itemize}
  \item What is object oriented programming (OOP)?
  \item What is an OOP data class?
  \item What is an OOP method, and why is it an efficient way to program any system?
  \item What is an OOP object, and how does it relate to classes and methods?
  \item Why is OOP effective for systems employing a GUI?
  \item What is the SAS/AF FRAME entry, and how does it incorporate GUI and OOP?
  \item What are the challenges to effective use of the FRAME entry in interactive systems?
  \item What is the role of the FRAME entry in CANDA development?
\end{itemize}
Pharmaceutical Industry

BACKGROUND

Recent advances in hardware and software have lead to a significant increase in the use of interactive systems in many environments. New operating systems such as DOS with Windows, Windows NT®, and OS/2® are more interactive and more graphical than previous text-based operating systems. This increase in interaction with the user has provided more easy-to-use power than ever before at this level. New application development software tools are more interactive and more graphical as they interface with the developer, and they of course produce more interactive, graphical systems from the user’s perspective. Microsoft’s Visual Basic® is a common example of these tools.

Specific industries are also now seeing important interactive applications that provide users with better access to data and more analytical power than ever before. In particular, the pharmaceutical industry has seen introduction of the CANDA and related data review systems. A CANDA is a highly interactive system for data analysis and review. The acronym stands for Computer Assisted New Drug Application. CANDA systems are intended for use by the FDA to review pharmaceutical manufacturers’ submissions of data and analyses results for product marketing approval.

CANDA systems are now powerful and important tools in the industry’s drug development process, but they were significantly less powerful and efficient until the recent software advances concerning graphical user interfaces (GUI) and object oriented programming (OOP). A GUI provides a powerful, easy-to-use interface for interacting with users. OOP is an effective approach to developing software, and it is particularly applicable to development of graphically oriented interactive software.

The SAS System has now added GUI and OOP tools to its application development product, SAS/AF. These tools are provided in the new SAS/AF FRAME entry, making this product a very powerful and very contemporary tool for interactive system development.

WHAT IS A GUI?

Simply put, a GUI is a graphically oriented multi-window user application. The phrase "graphically oriented" comes from the fact that the elements in the display windows are graphical in nature instead of text. Graphic window elements include icons, pushbuttons, scroll bars, radio boxes, and bit-mapped graphics.

There is a full integration of graphics and text in a GUI system. These two components were kept separate in older interactive systems. Such systems were consequently both more cumbersome to program and use, and less elegant in appearance.

Finally, a GUI system usually includes a mouse interface. Users navigate through the application by pointing and clicking selections which are represented by the graphic elements.

BENEFITS OF GUI

GUI systems tend to be simpler and quicker to use compared to text-based systems. Selection by point and click is faster than by text entry. Point and click selections are also more accurate than text entries, which are subject to keystroke error. Selections are also more intuitive because the graphic window elements used for selection can depict the idea or task.

The first result of this is that users become productive on a given system sooner because they require less training and less documentation review. A second result is that users become more productive because an easier-to-use interface allows developers to develop more powerful, complex systems, which in turn gives users better access to their data and more things they can do with it.

WHAT IS OOP?

Two caveats are in order before the technique is described. First, explanations of OOP can range from brief to quite detailed. The explanation herein is somewhat brief so that focus can remain on the role of the SAS/AF FRAME entry in application development instead of shifting to a treatise on OOP. But it is detailed enough for you to understand how SAS has provided you with OOP tools starting in release 6.08. Second, explanations of OOP, whether brief or detailed, tend to initially be quite abstract. You are assured that a concrete example will follow to clarify the abstract.

OOP is a technique for writing software. The phrase "object oriented" refers to an approach where emphasis during design and coding is on the data and the operations performed on that data. The procedure, or program flow, is de-emphasized.
Design of an application using OOP begins with analysis of the data, in order to group them into similar categories upon which similar actions are performed. Similar data categories form classes, and actions performed on classes are methods. After the classes and their methods are identified, each occurrence (or instance) of each class is identified. Each instance of a class is an object. Finally, the program flow is designed. This flow designates the relationships among the particular objects of the application, and it describes processing the objects; that is, triggering or sensing activity on the objects and calling methods to handle that activity as needed.

Implementation of an OOP application begins by constructing storage locations for the classes and their methods, and tools to call upon them to build objects. This may be thought of as a tightly managed software library of some sort. Next, the methods for the data classes are coded and stored in this library. Finally, the application's procedure is coded. This includes building objects from the classes library as needed, defining the activity (or events in OOP) to occur among and between these objects, determining when these events have occurred, and calling the appropriate methods to handle the events.

OOP EXAMPLE

Consider the abbreviated application represented by the following display screen. This application solicits a userid and validates it, then solicits a password and validates it, and then allows the user to select a report generation menu. Help is always available, as is the Exit function. Let us look at how the procedural programming technique would be applied to designing and coding this application.

<table>
<thead>
<tr>
<th>Welcome to the XYZ System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter your userid now:</td>
</tr>
<tr>
<td>__________ (Userid invalid, re-enter)</td>
</tr>
<tr>
<td>Enter your password now:</td>
</tr>
<tr>
<td>__________ (Password invalid, re-enter)</td>
</tr>
<tr>
<td>Reports _ Help _ Exit</td>
</tr>
</tbody>
</table>

The procedural programmer would mock up the display screen and determine that the program must initially display a fancy text string that reads "Welcome to the XYZ System" in bold italics and at a point size of 15. This programmer would also determine that the program flow would also have to initially display the regular text string "Enter your userid now: ", a space for entering the userid, and the three menu choices at the bottom. The design specification would read something like this:

- Display the title, userid prompt, userid entry field, and three menu choices at the bottom.
- If Help or Exit is chosen, display the help screen or exit respectively.
- If Reports is chosen, display a message that it is not available until a valid userid and password are entered. If a userid is entered, validate it against the userid table. If it is valid, display the password prompt and its entry field. If it is not valid, display the invalid userid message . . . , etc., etc. etc.

The procedural programming technique can be summed up simply by saying that the primary focus is on the procedure that the program must follow, that is, its flow. Data are viewed as items that are input to, or output from, the program as the flow of events unfolds. Specifications are written from this perspective, and code is then written from this perspective.

In contrast, procedure is de-emphasized in the OOP technique. Consider the OOP programmer's mock-up of the same application's display, noting especially the programmer's annotations on the right side.

<table>
<thead>
<tr>
<th>Welcome to the XYZ System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter your userid now:</td>
</tr>
<tr>
<td>__________ (Userid invalid, re-enter)</td>
</tr>
<tr>
<td>Enter your password now:</td>
</tr>
<tr>
<td>__________ (Password invalid, re-enter)</td>
</tr>
<tr>
<td>Reports _ Help _ Exit</td>
</tr>
</tbody>
</table>

Classes/Objects

FancyText

Text

TextEntry, Text

Text

TextEntry, Text

PushButton
Pharmaceutical Industry

During design, the OOP programmer has looked at the same application from a different perspective, and has identified that this application's display is composed of these classes: FancyText, Text, TextEntry, and PushButton. Further, the OOP programmer notes that the following methods will be required for each class:

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>FancyText</td>
<td>vary text content, font, color, size, intensity</td>
</tr>
<tr>
<td>Text</td>
<td>vary text content and hide/show</td>
</tr>
<tr>
<td>TextEntry</td>
<td>sense entry, vary hide/show and protect/unprotect, validate entry</td>
</tr>
<tr>
<td>PushButton</td>
<td>sense selection, vary text content, active/inactive, program called</td>
</tr>
</tbody>
</table>

The OOP programmer will now approach coding the application as follows. First, the four classes will be defined as graphic screen elements, and their definitions will be stored in a library. Next, the methods will be programmed for each class and also stored in the library. Next, code will be written to access these classes to define objects from them as needed. With this up-front work completed, the coding now focuses on constructing the display screen:

Locate one object from the FancyText class. Call methods to set text to "Welcome to the XYZ System", set size to 15 point, set font to Helvetica Italic, set intensity to bold, and set color to cyan.

Locate one object from the Text class. Call methods to set text to "Enter your userid now:" and set hide/show to show.

Locate one object from the TextEntry class. Call methods to set protect/unprotect to protect and set hide/show to show.

Locate one object from the Text class. Call methods to set text to "(Userid invalid, re-enter)" and set hide/show to hide.

Locate one object from the Pushbutton class. Call methods to set text to "Reports", set active/inactive to inactive, and set program called to "Rptmenu.Program".

Locate one object from the Pushbutton class. Call methods to set text to "Help", set active/inactive to active, and set program called to "Help.Program".

Locate one object from the TextEntry class. Call methods to set protect/unprotect to protect and set hide/show to hide.

Locate one object from the Text class. Call methods to set text to "Enter your password now:" and set hide/show to hide.

Locate one object from the Text class. Call methods to set text to "(Password invalid, re-enter)" and set hide/show to hide.

Locate one object from the Pushbutton class. Call methods to set text to "Help", set active/inactive to active, and set program called to "Help.Program".

Locate one object from the Pushbutton class.

... etc., etc., etc.

At this point, the code becomes procedural and follows about the same flow that the procedural programmer's does. But, as events occur for objects (e.g., the user enters a userid in the Userid TextEntry object), the code includes calling other OOP methods. So interwoven into the procedural code is a fair amount of calling OOP methods for OOP objects as needed to get the job done. Here is an example:

The program flow will, for the Password TextEntry object, initially call methods to set the object to hide status and to protect status. Then the program will do all the required processing for the Userid. Then, when the Userid is valid, the program will call methods for the Password TextEntry object to set its status to show and unprotect, and later call additional methods to sense entry and call the validation routine.

So, in summary, first we have programmed the definition of the TextEntry class, then we programmed the methods for this class, and then we wrote procedural code which displayed an object from this class for the password entry, and then we called methods as needed as the object was used during program execution.

SOME COMMENTS ABOUT OOP

Is it Different?

You might feel at this point that the OOP technique is not really so new and so different. It may seem to you
to just be a higher form of modularized code. In essence you are correct to feel this way. The OOP technique of focusing on data classes and the actions upon them is an effective way to maximize the modularization of the code. OOP proponents will speak of this approach as different, using terms such as "a paradigm shift" to describe it. To a certain degree they are correct. But at some point the differences become semantic and the results of the OOP technique can reasonably be thought of as just a higher form of modularized code. What is particularly different is the perspective taken on the design, which facilitates optimal modularization. But even operating from this perspective still yields just modularized code, albeit to an optimal degree.

Were the focus of this paper a thorough analysis of the OOP technique, I would delve deeply into the differences between the two approaches instead of leaving it here. But the focus herein is about how you can gain the advantages of a GUI and OOP while building interactive systems with SAS/AP and the FRAME entry. For this purpose, it is sufficient and not misleading to view OOP from the viewpoint of one of its primary results, highly modularized code.

Hence, I will close out this comparison by simply noting that some of the important benefits of the OOP technique do stem directly from the code modularization, and note that these same benefits can be obtained from non-OOP modularized code, albeit probably to a lesser degree. They include efficiency through re-usable code, design consistency through re-usable code, ease of debugging, and self-documentation.

Well Suited to GUI Applications

Note that the sample application used herein to present OOP is an interactive one with a GUI. But OOP techniques can be applied to the design of any application . . . interactive, GUI, or otherwise. It is interesting, though, that OOP is particularly well suited to GUI interactive applications. This is so for two reasons. First, the elements on a graphics display screen can be readily viewed as objects with methods during the design phase. There is not a lot of time required, as with other applications, to determine what ought to be the object classes and what ought to be the set of methods for each class. Second, GUI object classes tend to be reused often because interactive applications tend to have many screens which re-use the same screen elements. Hence, once the up-front work of defining them and building a library for them is done, there will be a payoff for this effort stemming from their large amount of re-use.

The work of identifying object classes and methods is not always as straightforward in other types of applications, or in the background components of interactive applications. By background components, I mean, for example, the database access and number crunching activity that occurs behind the scenes once a user has selected among choices on the video display.

A Blend of Approaches Today

Another point concerns that fact that many OOP development tools today may require procedural code, too. That is, they may not produce systems that are exclusively a succession of calls to pre-coded methods. Another way to say this is that the object classes and methods identified and built for use in developing applications may not be so powerful and comprehensive as to cover all conditions in all the applications they are used for. Customized procedural code may be needed to fill in the gaps between what the object classes and methods can do in a given application. In many applications today, as was just mentioned above, the user interface is a GUI which is built using the OOP approach, while the background processing is coded without any OOP techniques at all. And in places where the OOP technique is applied, such as the GUI, it is often blended with custom procedural code. The following discussion of our sample application herein explains why blending of the approaches occurs.

Consider the TextEntry class we have defined. It is somewhat general in that it is used for both the userid entry and the password entry. Nothing in its definition or methods says it is for entry of one of these two pieces of data. Thus, custom procedural logic must handle each particular context of its use. So, after the userid is entered, custom program logic must cause the Text and TextEntry objects to be shown for password entry. And after password entry, custom procedural logic must activate the PushButton object for Reports. These tasks cannot be methods of the TextEntry class that would automatically occur after entry because they do not apply in all contexts.

We could have instead defined a UserID-TextEntry class that had a method to show a Password-TextEntry object when a userid was successfully entered. And the Password-TextEntry class would, in turn, have a method to activate a Report PushButton after a password was
Pharmaceutical Industry

successfully entered. This way, we would have avoided the need for any custom procedural logic and our application would be comprised of purely a series of calls to pre-programmed methods. But there is a tradeoff involved here between doing more work upfront in an OOP development environment to define and build more customized classes, and doing more work later as a given application is developed to write custom procedural code to fill in the gaps between what can be accomplished with fewer, less customized object classes and methods. In today's development environment a balance is sometimes struck between the two approaches. As time goes on and developers have more opportunity to identify, build, and refine classes and methods, application development will move more and more toward the pure OOP approach.

One new feature of OOP development environments is hastening our march towards a pure OOP approach. It is the use of graphically-oriented code-generating development tools which allow developers to engage objects and to graphically identify the relationships between them. These development tools then automatically generate the custom code required to fit the objects together in a given application. This achieves the requirement of customizing the application which is built using generalized objects, and it does so without putting a burden on the developer to write customized procedural code.

GUI, OOP, and SAS

One thing that should be clear at this point about the OOP technique for application development is that it tends to have a higher startup cost than procedural programming, but a concomitant higher long-run payoff. The startup cost stems from the effort needed to identify classes and methods during the design phase, and to then write code to build a library for them, define them, store them in the library, and retrieve them to engage objects as needed. The long-run payoff stems from the fact that as objects are re-used, no additional code need be written for them. The only new code will be the procedural code which is interwoven around use of the objects. And in OOP development tools with code-generation, even this is avoided.

But what if someone could save you much of the startup cost of using OOP in a GUI application? That would probably be an attractive proposition. This is what the SAS System now offers with SAS/AF and its FRAME entry. SAS/AF has always been a tool for interactive system development. Now, with the FRAME entry, the SAS System provides GUI and OOP tools for interactive system development.

The SAS/AF FRAME entry provides pre-defined OOP classes for the graphic screen elements of a GUI. Their definition and the library for handling them is there. You need only call upon them to engage the objects you need for your GUI applications. Further, each GUI class comes with a variety of pre-coded methods. These are built-in software modules for use in your applications. As you engage objects in your application, you simply make calls to these methods without writing method code yourself. Finally, the FRAME entry also supports the definition and use of custom classes, objects, and methods if you need them in an application.

The FRAME entry provides 18 pre-defined GUI classes. Below is a sample list to give you a flavor for them:

<table>
<thead>
<tr>
<th>Class</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushButton</td>
<td>choice selection (e.g., menus)</td>
</tr>
<tr>
<td>Icon</td>
<td>PushButton with iconic representation</td>
</tr>
<tr>
<td>Radio Box</td>
<td>label and marker for selecting mutually exclusive choices</td>
</tr>
<tr>
<td>ListBox</td>
<td>scrollable box for listing and selecting</td>
</tr>
<tr>
<td>GraphicText</td>
<td>text displayed with graphic software</td>
</tr>
<tr>
<td>ScrollBar</td>
<td>scrolling values into view</td>
</tr>
</tbody>
</table>

And here is a sample from the 13 pre-coded methods available for one class, the RadioBox class:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>activate</em></td>
<td>activate or gray the specified radiobox item</td>
</tr>
<tr>
<td><em>get_value</em></td>
<td>numeric return value of the active station</td>
</tr>
<tr>
<td><em>is sel</em></td>
<td>reports if specified item is selected</td>
</tr>
<tr>
<td><em>set_label</em></td>
<td>assigns a label to a specified item</td>
</tr>
</tbody>
</table>
SAS/AF FRAME and CANDA's

A CANDA is a highly interactive system that benefits greatly from a GUI. And, as stated above, a GUI application benefits greatly from an OOP approach to development. Hence, SAS/AF with its FRAME entry which uses a GUI and OOP, is an effective tool for CANDA development. I have found this assumption to be true from my recent CANDA development experiences using the SAS System.

But it is important to note that interactive systems such as CANDA's are not completely comprised of user interaction. Such systems have two major components. The first, of course, is the user interface. This component concerns all the solicitation of information from the user regarding what tasks are to be done, and the presentation of the results of doing these tasks. The second component concerns the execution of these tasks behind the scenes. So, for example, in a CANDA, the interface component will solicit information about which dataset is to be queried, what the query is, and how the results are to be displayed. And the behind-the-scenes component will involve inputting the data, performing the number crunching, and preparing the results for display.

It is possible to use the OOP technique for design and coding of both major components of an interactive system, or to limit its use to one component or the other. If you develop an interactive system such as a CANDA using the SAS System, you will find it reasonably easy to use the OOP approach to build the interface because, in essence, the SAS Institute has done much of the up-front work for you with the SAS/AF FRAME entry.

The same is not true of the behind-the-scenes component and, thus, you are likely to still design and code this component more or less from a procedural perspective. This is because the up-front work for an OOP implementation has not been done for you, and because, overall, the programming tools provided with SAS, specifically the Base SAS language, the Screen Control Language (SCL), and to some extent the statistical procedures, have not been designed as OOP development tools.

I say "more or less" from a procedural perspective because these SAS programming tools can be used to a certain extent to implement an OOP design, and once you become familiar with the OOP technique, you may indeed incorporate some of it in the behind-the-scenes component of your interactive systems built with the SAS System. But given the large up-front cost for an OOP application development, and given that the SAS System does not save you any of this effort for the behind-the-scenes component, it is likely that your use of OOP will be limited in this component.

This should not be seen as a determining reason to not use the SAS System for development of interactive applications such as CANDA's. Few, if any, application development tools available today are strong enough on both the user interface and behind-the-scenes sides to allow application development completely using OOP techniques. Further, not only does the SAS System effectively provide a GUI and OOP for the user interface component of such systems, but its procedural languages (Base SAS and SCL) are reasonably strong tools both for the behind-the-scenes component, and the procedural aspects of the user interface component.

There are other GUI/OOP application development tools and languages available on the market today. One popular tool is Visual Basic which uses an upgraded version of the Basic language. Another is Visual C++ which uses the C++ language, and a third is Parts which is based on the SmallTalk V language. These items may provide more OOP support for the behind-the-scenes component of interactive systems such as CANDA's.

But the choice of software environment for a CANDA is not as simple as the pure GUI and OOP performance of the various software tools available. A CANDA is a major system, and as such, there are many dimensions to consider before selecting the software environment. The choice of a software environment for a CANDA is too large a subject to cover in any detail in this paper; however, I can make several comments about it, based upon my recent SAS-based CANDA development experiences:

- SAS/AF with its FRAME entry is about equally functional to other products for the GUI component of a CANDA.
- CANDA's developed with the SAS System will have a similar, but not exact, look and feel as systems developed to run in the Windows environment with Microsoft tools such as Visual Basic. This is particularly true with the release of version 6.10 of the SAS System.
Pharmaceutical Industry

- Some of the latest desktop GUI development tools can still produce an interface that appears slicker than that produced using SAS/AF and its FRAME entry. But the greatest challenge and greatest payoff in a CANDA does not come from producing a slick appearing interface. Rather, it comes from developing an intelligent interface that is intuitive to use, visually appealing, and appropriately leads the reviewer along in the review without he or she requiring a complete, in-depth knowledge of the database before the review begins. The SAS GUI is an effective tool for doing this, and any belief that the slickest appearing interface is required to produce the most effective CANDA is false.

- When the source of your data for analysis and review is SAS data sets, as is often the case with clinical trials data, then a SAS-based CANDA will save you the difficult and costly requirement of migrating that data to another database environment for your CANDA.

- CANDA's can have complex data relationships. Thus, their ad hoc data analysis tools, which are one of their key components, must be based upon strictly defined relationships between data tables (such as in a relational database model) to avoid both excessive processing time and overly large output tables. This is possible with the SAS System, which provides the ability to perform table join processing with either Base SAS code or the SQL procedure.

- Important value in a CANDA stems from statistical analysis that can be conducted on data subsets created from ad hoc querying. A SAS-based CANDA provides the full statistical power of SAS to achieve this value. Other application development tools either do not provide this statistical capability, or do so only in a limited manner. You may, thus, be forced to export your data for processing outside your CANDA system, such as with a spreadsheet program. This may mean less data summarization power, or less system control, or both.

- Table joins will produce normalized tables when the tables are joined according to strictly defined relationships; however, such joins can still produce repeated data values in a given table column that must be normalized before accurate analysis (counts, means, etc.) can be performed on that column. Failure to normalize joined data which exhibits column-specific repeated values will lead to erroneous analytic results. SAS-based CANDA's will provide maximum flexibility compared to other software products to accomplish the normalization. In contrast, other application development tools either do not provide this capability, or do so only with great difficulty for you.

- CANDA's developed with the SAS System will run on multiple platforms, just as the SAS System itself does. At this point, there is a large number of platforms to choose from.

- CANDA's developed with the SAS System will avoid the introduction of an additional application development environment. This, in turn, will save any SAS-trained organization the costs of developing and maintaining software skills beyond those presently kept in-house (except for the addition of FRAME skills to your present SAS/AF programming staff).

**SUMMARY**

Recent advances in hardware and software have lead to significantly increased activity in interactive system development. The hardware advances have concerned graphics display devices, and the software advances have concerned GUI software and the OOP approach to system development.

Prime examples of this increase in interactive systems include OS/2, DOS with Windows, and Windows NT at the operating system level, Visual Basic at the application development tool level, and CANDA's as an interactive application familiar to the pharmaceutical industry.

The GUI is well suited to interactive systems because it provides a superior interface to the text-based interface, and OOP is particularly effective for developing GUI interactive applications because graphic screen elements so readily lend themselves to being viewed from the perspective of OOP classes with methods.

The SAS System now offers the advantages of GUI and OOP in its interactive system development tool, SAS/AF. They come in the form of the new SAS/AF FRAME entry.

Since CANDA's are highly interactive systems, and since the SAS System now offers strong GUI and OOP
support for interactive system development, it follows that the SAS System is now an effective environment for CANDA development. This assumption has been borne out by my recent CANDA development experiences entirely within the SAS System.

There are other software tools available today for interactive system development that provide the advantages of GUI and OOP. The performance of each will probably exhibit strengths and weaknesses vis-à-vis the SAS System. However, a CANDA is a major system development project and, as such, the decision about the software environment must consider dimensions that go beyond the GUI and OOP performance of the software.

I have not done a critical evaluation of all those dimensions; however, organizations will probably find it valuable that a SAS-based CANDA will: (1) have a contemporary GUI development tool; (2) have a similar look and feel to Microsoft Windows; (3) is a very effective tool for designing an intelligent and therefore effective interface; (4) provide easy access to your study data and avoid costly database migration when the data resides in SAS data sets before the CANDA is built; (5) allow strict definition of table relationships, as in a relational data model; (6) support statistical analysis on data subsets generated from ad hoc querying; (7) offer more flexibility and capability than most other system development environments to normalize data resulting from table joining; (8) be multi-platform; (9) avoid introduction of another software environment with its requirements for developing and maintaining new software skills.

REFERENCES


SAS and SAS/AF are registered trademarks of SAS Institute, Inc. in the USA and other countries. OS/2 is a registered trademark of International Business Machines Corporation. Windows, Windows NT, and Visual Basic are trademarks of Microsoft Corporation. * indicates USA registration.

For more information about this paper, contact the author:

Barry R. Cohen
Planning Data Systems
PO Box 666
Ardmore, PA 19003
610-649-8701

Mr. Cohen is an independent information systems consultant, specializing in interactive application development and object oriented programming. He has been using SAS software since 1980 in a wide variety of industry applications, including a focus on the pharmaceutical industry.