The separation of Graphical User Interfaces and Applications in software development within the HORACE project.

D. S. Akeroyd
Systems Development Branch
The Met. Office
Bracknell, UK

Summary: This paper describes the method of software development used by the HORACE software development team and how the separation of Graphical User Interfaces (GUIs) from applications has led to a re-appraisal of the software design and production process.

1. INTRODUCTION

1.1 Since the introduction of Graphical User Interfaces (GUIs), it has been common practice to incorporate the GUI within the code of the application. With the advent of the X Window System the application code has become a subordinate part of the code defining the GUI. The application code now comprising of modules that are executed when some form of user action takes place, such as clicking a mouse button when the cursor is over a widget. Whatever the case, the GUI and the application are bound together as one single executable task.

1.2 Before the HORACE project began, GUIs for operational systems within the Met. Office had been created using GKS type primitives. Each representation of a real control object such as a push button, toggle, slider etc. had to be constructed from scratch. Obviously the code that made up common devices such as these were incorporated into libraries for re-use by other software developers, but these home made devices were complicated, difficult to maintain and definitely none standard.

1.3 With the introduction of workstations into the Met. Office it was obvious that the best thing to do would be to use the native GUI standard, that of the X window System and in particular the Motif toolkit. Although this was a good idea in principle there were several disadvantages, particularly to the HORACE project.

- Even using the MOTIF toolkit, creating a GUI uses many more lines of code than even a home grown GUI.

- GUIs written using MOTIF generally require a good knowledge of the C programming language, and all our software developers use FORTRAN.

- There are many hundreds of person years of software already written in

FORTRAN within the Met. Office, chart plotting and contouring packages for instance, which would be difficult to incorporate as subordinate modules of a C encoded MOTIF GUI.

1.4 The answer to the first problem was obvious, buy in a GUI builder package to alleviate the problem of having to hand encode the GUIs, leaving only the functionality (callbacks) to be developed.

1.5 However, this still left the remaining problems of staff training and software re-use. It became apparent that it would be better to give a few people a greater depth of understanding of C and MOTIF and leave the application developers to work in FORTRAN. But how to incorporate the FORTRAN code of the application developers with the C and MOTIF code of the GUI, and what about the re-use problem. It became clear that the neatest solution would be to divorce the GUI from the application completely and have them run as two separate processes communicating with one another.

2. THE ADVANTAGES OF THE SEPARATION OF GUIs FROM APPLICATIONS.

2.1 The GUI may be prototyped in isolation from the application, allowing potential users to try the look and feel of the application without the application being written, or only part written. This allows the developers to check that the software being developed will fulfil the functional requirement. This is facilitated by the use of a GUI builder but could be done without one.

2.2 Re-use of existing applications without major re-structuring. There are two ways of doing this, either add modules that can communicate with the GUI, or communicate with an existing application by the GUI sending data to and receiving data from the GUI as if the GUI were talking to a standard alphanumeric terminal using the Unix interprocess communication facility of pipes.

2.3 Splitting the GUI and application completely means that two developers can work in parallel but asynchronously, thus shortening the development elapse times.

2.4 Updates to X-windows and MOTIF only require the re-building of the GUIs with no impact on the application. Obviously the same applies if the application needs amending.

2.5 As previously stated, GUI expertise can be concentrated into a small team thereby giving them a greater depth of knowledge, rather than having it spread more thinly throughout the development team. This means that anything out of the ordinary is less likely to stall the development.

2.6 Development of the GUI by a small group of people makes for easier enforcement of consistency of the GUI across the project.

2.7 The fact that the GUI is separate and can be produced easily mean that so long as the underlying functionality of the GUI is unchanged then cosmetic changes to the GUI in response to user criticism can be easily achieved.

2.8 Allows more than one application per GUI if desired, see section 3.

2.9 Easing of the critical path, because parallel development is possible although both the application and the GUI must be finished before the functionality is satisfied. If one is finished before the other, it releases that developer to move on to other things.

3. WHICH SHOULD BE THE CLIENT AND WHICH THE SERVER?

3.1 During the analysis phase of a project, one attempts to model the functionality of the system and the data flows within it. This then leads on to the design of tasks to fulfil the required functionality. These task designs are then developed into applications. If the applications are to have GUIs then the GUIs must also be modelled in the designs, this works if the GUI and the application are one, but leads to very complex designs.

3.2 If the application and GUI are separated into two tasks within the design then these should be modelled separately, but just how do you model a GUI? It soon became apparent that if you use a GUI builder then in fact you don't, you use the prototype GUI as the design, and with the builder used in the HORACE project this design can be documented. But what information do you use to design the prototype? Well if the application has been designed then its data flows into and out of the application are defined. These flows can be used as the basis for the design of the GUI, the GUI is then prototyped, checked against the analysis and demonstrated to the users. In this approach the application can be viewed as the client and the GUI as the server of the application.

3.3 This approach works but there are problems, the first is that although the prototype GUI can be checked without any functionality in place, the application has to be written before it can be checked. If the application is viewed in isolation from the GUI, then if the analysis and design are done correctly then the application should produce the required functionality. The problem is that as far as the user is concerned the application is driven by the GUI where as in fact using this approach the GUI is driven by the requirements of the application. It soon becomes obvious that there is conflict here and this can only be resolved when the GUI and application are hooked together. If this produces difficulties then it may be that either the GUI or the application or both have to be altered.

3.4 The solution to this would appear to lie in the fact that the use of MOTIF means that the application is naturally subordinate to the GUI, thus it is the GUI that is the client whose request for function-

ality is served by the application. In this method the GUI is prototyped, using the GUI builder, before the application is designed. This can then be checked against the analysis and demonstrated to the users. If this checks out then data flows from the GUI to the application can be specified and the application designed to satisfy the demand of the GUI. The GUI can now be built and the application developers can go ahead and design and build the application. When they are linked together there should be less conflict as the application is now satisfy the requirements of the GUI and hence the user. The chances of either of them, particularly the GUI, being in need of amendment are significantly reduced.

3.5 In fact this method of production actually means that the GUI is in effect a part of the analysis process, and that the design of the application should not be started until the GUI has been signed off.

3.6 Finally, making the application the client makes it easier to identify where it would be useful to have more than one application per GUI.

4. THE IMPACT ON ANALYSIS AND DESIGN.

4.1 The stated prime reason for using Structured Analysis and Design Methodology is to produce more robust and maintainable code. One way that this is done is by the method enforcing modularity.

4.2 The fact that the functionality of an X-Window GUI is in the callbacks, means that the code is forced to be modular. In fact if a Structure chart were attempted for a GUI, it would turn out that the lowest level of decomposition would be a callback.

4.3 It also tends to be the case that non-object orientated methods, such as the YOURDON tool set used by the HORACE project, favour a deterministic approach. The fact that GUIs are event driven means that they are by their very nature non-deterministic, and thus tend not to be suitable for our current methodology.

4.3 The method of GUI production used by the HORACE project does not use any real SADM at all. The only place that the GUIs feature in the method are in the task model, with one process bubble for each GUI, and in the use of Data Structure Diagrams to illustrate the structure of the Inter-Process Communications between the GUI and the application.

4.4 Using a GUI builder means that the design of a GUI is self documenting. Maintenance is still catered for because the structure of the GUI can be examined using the builder. The tool also allows identification of which callbacks are attached to which widgets.

4.5 Robustness of the code can be enhanced by using a variety of Quality Assurance tools before the

code is released.

4.6 Maintenance of GUI code can be further enhanced by using reverse engineering tools, as and when maintenance is required.