A Architectural Software - Pattern for the Construction of Adaptable GUIs for Network Based Control Systems

Dipl. Inform. S. Schreiber-Ehle, Dipl. Ing. B. Seifert[1]
Screen Paper Communication GmbH
D-53359 Rheinbach, Schumannstr. 6
049-2226-909616
info@screenpaper.de

Abstract

Software systems today are the most important tools in the process of command and control information management and planning. The users have to fulfill rapidly changing tasks and only have short training time. Therefore we claim that a Piece of Software is just as good as it's Graphical User Interface (GUI). The elements of a **GUI centered software architecture** for C2 systems have been identified and defined. A design pattern framework has been constructed to provide the ability of dynamic GUI adaptation. The basis of the concept is to define dialog components for subtasks that are exchangeable by means of using a object protocol for the communication between components. The design is based of the definition of independent components, that can be spread over a network, or work together like a web-client-server architecture. Componentware techniques and object bus technology were used to apply the architecture to a C3 system, by providing tools for the task of generating military situation reports with geographical referenced data. The geographic display tool was integrated into existing office environments, by the application of the proposed architecture.

1. **Problem description**

Software systems are the most important tools in managing and planing command and control information. But a piece of software is just as good as it's graphical user interface (GUI). So the GUI determines the usability of a software tool. If the user is unable to find the needed functions to fulfill his/her task, the software does not support to his/her work, but acts as an obstacle [Stary et al., 1997]. Because users have to perform different tasks requiring very different skills there is no chance to create the one and only optimal (ergonomic) interface.
One GUI can be appropriate for a special user performing a certain task. But it may be insufficient for the same user working on another task. This leads to the conclusion, that various interfaces are needed for different tasks and skill-levels. Behind this interfaces have to be tools available for all kind of tasks. This is the reason for the need to integrate existing commercial of the shelf products available at some time, especially after a system was designed and created.

Another issue is the need to keep track of the user's environment. He/she has to react on certain data input (e.g. sensor information) or incoming messages. This input has to be integrated into the current workspace, without preventing him/her to do the daily work. The most significant

[1] Many thanks to Prof. Dr. B. Döring, Institute of applied Sciences (FGAN), Werthhoven, Germany

problem of bad designed GUIs is that they do not support the users in performing their tasks, but lead to confusion. Thus frequently they are the reason for human errors.

These items are main criteria to be considered in designing command and control systems. Errors in this area can result in severe consequences for the system, even for the lives of people.

Additionally users of C2 systems are working in a very special situation and environment. They do not have much time to learn to work with the system, frequently very short time to react is available for them, and they carry high responsibility. Also in performing their tasks lots of information distributed on their computer network has to be considered.
Combined with confusing GUIs, this situation is hard to manage for users of command and control systems.

2. **Approach**

Because of the central meaning of the GUI for man-machine system performance, software ergonomic considerations should be the starting point and central issue in system design. An important issue is the adaptability of the system according the users needs and his situation. The GUI should be adaptable according to three parameters : task, user skill and environment.

This means, that the interface has to be changeable at runtime depending on the specific task of the user, the skills the actual user has, and the actual environmental situation. To technically solve this problem a software architecture will be defined that supports dynamic GUI creation in terms of a pattern framework.
Especially in the operational environment of C2 systems there are some circumstances that make dynamic GUI creation easier : The work of all people interacting with a C2 system is based on the same data, sometimes they just need different levels of abstraction (views) in working on a shared network. The main issue of network centered cooperative work is that similar tools can be used.
The tools for certain subtasks a GUI supports, are provided inside the network, we will call them "basic elements" or "components".
So different GUIs can be put together by using the same basic elements. State of the art in dealing with this problem of combining basic tools is the application of the componentware technique [Microsoft, 1997], [OMG, 1999].
Another advantage in using componentware techniques is the ability to integrate existing commercial and governmental software. So the need to provide tools for the varying tasks can be met by choosing appropriate existing products, or creating small extension tools.

The first step to such application is planning the user interface based on a task oriented analysis of the C2 system structure. After the tasks are identified and mapped to functions that are necessary to perform the tasks, object oriented analysis leads to an object model for the things involved. Once it is determined who does what, the main work developing the architecture starts.
No one would think he knows the architecture of a house just knowing which kind of materials are needed. The same way the object model itself does not describe the architecture of the system yet. The dynamic behavior is the essential of a systems face.

The adaptation of a GUI to the skill level of a user or environmental requests can be set up in three different ways.
First , the user himself or some operator can manually adapt the GUI, secondly it can be done inside the system by GUI agents watching the users actions, third it can be done automatically by some environmental system, that keeps track of e.g. certain sensor data.
We will now consider the basics of how to put up the system, by defining the exchangeability of dialogs in a mathematical way. After that we come to the conclusion, what criteria have to be fulfilled to build a system of exchangeable parts.

3. **Theoretical considerations – exchangeability of dialogs**

To describe the function of the system we define a task *T* consisting of *n* subtasks. The task transforms the input data into the output data. The data can be described as words *w* of a formal language based on an alphabet *S* consisting of alphanumeric characters.

$\Sigma = \{\text{alphanumeric characters}\}, \quad w \in \{(c_1,...,c_m) | \; c_i \in \Sigma\} \quad \text{where } m,i \in N$

(1) $T(w_0) = w_n$
The result of applying task T to $w_{0,}$ is $w_{n,}$ where T consists of subtasks:
(2) $T(w_0) = T_1 \circ T_2 \circ ... \circ T_n (w_o)$

Each task has a finite set of r Dialogs. These Dialogs are adapted to e.g. the users skill.
$M_D = \{D_0, ... , D_r\} \quad \text{where } r \in N$
For example : the task may be to order books, and the subtasks are to give addresses for delivering and billing and providing a list of books. There are dialog-elements for each subtask e.g. adress input.

Performing a sub task means working on one of these Dialogs:

(3) $T_i (w_{i-1}) = D_{ij} (w_{i-1}) = w_i$

It is important that each Dialog for a certain subtask returns the same result : the resulting data word is always the same.

(4) $D_{i1} (w_{i-1}) = D_{ij} (w_{i-1}) ..... = D_{ik} (w_{i-1}) = w_i$ where $i = 0, ... ,n$ subtasks
where $1 < j < k < r$ Dialogs

For example : this means, that there can be different dialog elements for the same task : “billing address input”. But the result of each dialog is the same as produced by other “billing address input” dialog elements inside the system. Therefore they are exchangeable.

The following figure shows how the Dialogs can be ordered in a matrix, where the first index represents the number of the subtask and the second index represents the corresponding dialog element for this subtask :

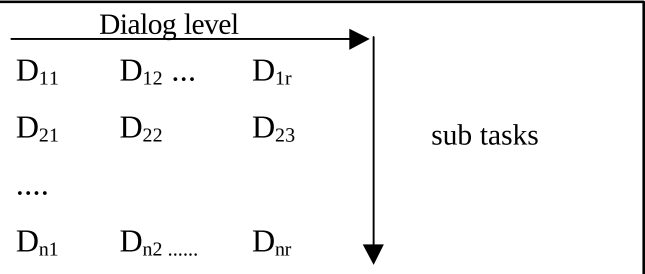

Dialog level
D_{11} D_{12} ... D_{1r}
D_{21} D_{22} D_{23}
....
D_{n1} D_{n2} ...... D_{nr}
sub tasks

Figure 1 : Dialog levels and Subtasks

For example : if we have three subtasks and three dialog levels (which means that there exist three dialogs, for example one for novice users, one for occasional and one for expert users)

$$\begin{array}{lll} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{array}$$

Equation (4) leads to the conclusion that various combinations of the dialogs are possible:

(5) $D_{ij}(w_{i-1}) \circ D_{i-1\,j}(w_{i-2}) = D_{ij}(w_{i-1}) \circ D_{i-1\,k}(w_i)$

For our “book order” example this means, that it is possible to use first a novice level dialog to give the billing address ($D_{11}(w_0)$), and in the second step the user can choose the expert-level dialog ($D_{23}(w_1)$) or the novice-level dialog ($D_{21}(w_1)$) to give the book-list. In both cases, the resulting word and therefor the result is the same.
$D_{11}(w_0) \circ D_{21}(w_1) = D_{11}(w_0) \circ D_{23}(w_1)$

The words made up of Σ can be the elements of an object bus protocol. So the results of each dialog step (element) can be transported to all software pieces, that are connected to the system by means of an adapter mechanism, defined by the architecture.

To perform the whole task a user works on the subtasks by selecting his Dialog for each task.
The intermediate results (data words) can be described as states the user reaches on his way through the task sequence. The selected Dialogs represents the way from one state to another. These transitions between states can be understood as part of a finite state automaton, where one state is transformed into another by a letter of the automaton’s input alphabet [Balzert, 1996]. Obviously the letters of the input alphabet are the Dialogs.

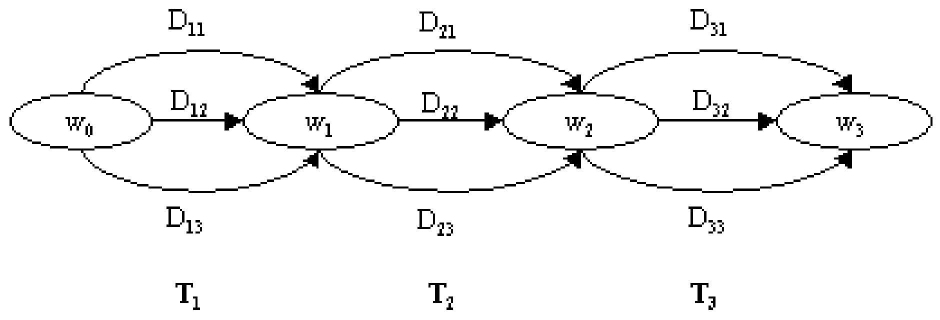
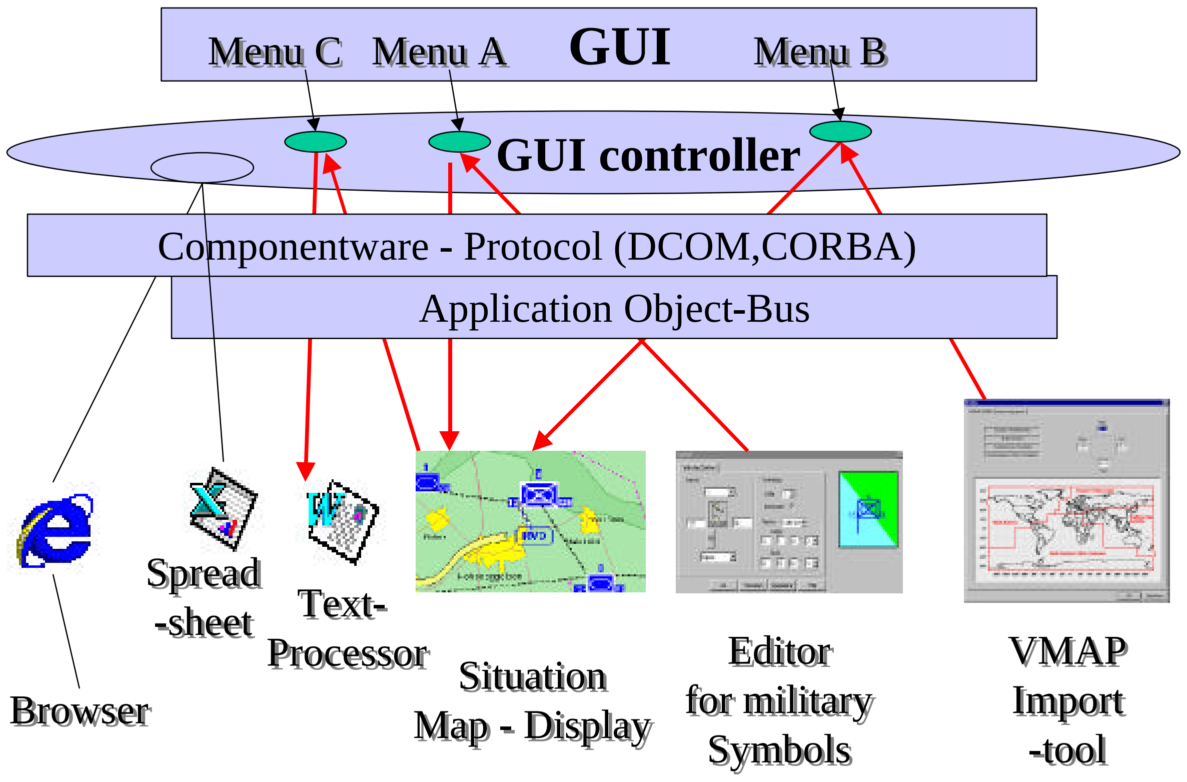


Figure 2 : States of the automaton

Imagine a certain task, that can be done by subtasks T1, T2, T3. According to these Subtasks dialogs adapted to three skill levels are defined to deal with the user interaction . The names of these dialogs are D11,..., D33.
The finite state automaton shows, that we can choose combinations of the dialogs as a sequence to work on Task T. We can choose one of the dialogs for the subtask we are working on, we only have to keep the sequence of the subtasks. The result of each subtask This is the same for any Dialog Dij ($i,j \in N$, $1 < i < 3$ and $1 < j < 3$).
For our example this means, that we can choose any skill-level-dialog element to perform the task of book ordering.
After modeling a system with such multiple adapted dialogs, we can make up the whole system by a collection of dialogs for each subtask, where the appropriate type of dialog is chosen at runtime by the user, or by some GUI agent examining the users interactions.
This theoretical consideration in fact is the application of a design pattern separating the tools (called controllers) and the data (called model).

4. **Results**

The elements of a GUI centered software architecture for C2 systems have been identified and defined. A pattern framework has been constructed to provide the ability of dynamic GUI creation and therefore adaptation. The current GUI is determined by configuration data that can be manipulated manually or automatically. The main architectural elements are the separated GUI, with the GUI controller, which connects components that communicate to each other via the Application protocol. This Protocol is designed as an object bus, transmitted by means of component object busses e.g. DCOM or CORBA [Plasil et al.,1998].
Two components are connected to each other as a communicating pair [Schreiber - Ehle, 1999]. The communication is initiated by the user choosing the attached menu.



The components exchange data with each other while the user is working with the tools they implement. For example : if an user wants to import data, he chooses the “VMAP – Import – tool” to read geographic data from the VMAP[2] CD´s and generate input objects that are sent to the “Situation – Map- Display”, which displays the geographic data as a map.

The user’s tasks are being supported by commercial and military software tools. Their configuration, especially the information about communicating pairs, is stored in a three dimensional GUI matrix. The matrix is exactly the one defined in chapter 3. A dynamic interface mechanism working with the GUI – matrix, controls the system by directing communications via an object bus, that serves as a communications protocol. Another design pattern that was be used in this framework is the model-view-controller pattern [Gamma et al., 1995]. Components work as one of the three elements : 1. as a data storage (= model) 2. as a data manipulating tool (= controller) 3. as a display tool (= view). Each component of the framework ideally has to implement only one of this elements. Sometimes it seems to be useful to build a component as a pair of view and attached controller, because the things a user manipulates have to be visible for him/her to make sensible changes.

A c2 - system has been designed and implemented according to this pattern framework. The Implementations use DCOM and CORBA for the network communications. One implementation

[2] VMAP = Vector Smart Map, geographic vector data by National Imagery and Mapping Agency (NIMA)

(Implementation 1) uses a Microsoft-Office environment to create a GUI for a c2-system by using the Microsoft-Products Word, Excel and Internet Explorer to give Word-Processor-, Spreadsheet- and Webaccess –tools. Database tools based on ORACLE are used to provide a consistent way of persistent storage. These essential tools are completed by a military situation editor tool-set and some geographic information system (GIS) functions. The user can integrate the military situation map as part of his office documents and applications.
Another implementation (Implementation 2) uses a database tool for c3 processes in the ATCCIS [3] environment. The military situation editor in integrated as a CORBA component, displaying the map by means of a JAVA -Applet.
The principle of the implementations is similar, the essential tools are covered by existing commercial of the shelf products. Additional to the integrated commercial products, special tools matching military needs (e.g. a military situation editor) are developed as far as needed and integrated.
"Implementation 1" is being used by German governmental institutions and will serve as an integration platform for wider military applications now (e.g. situation analysis, distributed databases and replication as specified in the ATCCIS standard).
At the moment a tool for electronic radar systems is connected to the military situation map display. In the early implementations the configuration is controlled manually be some operator, preparing the working place for some user. The next step is to define a framework for agents keeping track of environment situation and user skills. Based on this information the GUI can be adapted to new situational requirements and skill levels of the user.

5. **References**

[Balzert, 1996], Helmut Balzert. *Lehrbuch der Softwaretechnik.* Heidelberg , Spektrum.
[Gamma et al., 1995] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns*. Addison Wesley, 1995
[Microsoft, 1997] Microsoft Corporation. *The Component Object Model* http://www.microsoft.com/com, 1997
[OMG, 1999] Object management group. *CORBA-Components :* http://www.omg.org/library/schedule/CORBA_Component_Model_RFP.htm, 1999
[Plasil et al.,1998] F. Plasil and M . Stal. *An Architectural View of Distrubuted Objects and Components in CORBA, Java RMI, and COM/DCOM.* Software Concepts & Tools 1998(66), Springer, 1998.
[Schreiber-Ehle, 1999], Sabine Schreier-Ehle. *Adaptable GUIs based on the Componentware Technique,* Human Computer Interaction ´99, 1999
[Stary et al., 1997] Stary,Riesenecker-Caba, Kalkhofer, Flecker *EU-CON- Ein Verfahren zur EU-konformen Software-ergonomischen Bewertung und Gestaltung von Bildschirmarbeit,* Zürich, vdf, 1997

[3] ATCCIS = Allied Tactical Command and Control Information System