On user-friendly interface construction for CACSD packages

Ravn, Ole

Published in:
IEEE Control Systems Society Workshop on Computer-Aided Control System Design

Link to article, DOI:
10.1109/CACSD.1989.69828

Publication date:
1989

Document Version
Publisher's PDF, also known as Version of record

ON USER-FRIENDLY INTERFACE CONSTRUCTION FOR CACSD PACKAGES

Ole Ravn, Assoc. Prof. M.Sc.E.E.
Institute of Automatic Control Systems,
Servolaboratoriet, Building 326,
Technical University of Denmark,
DK-2800 Lyngby
Denmark.

Abstract: The paper presents some ideas that are used in the development of an user-friendly interface for a CACSD package. The concepts presented are Integration and Extensibility through the use of object-oriented programming, man-machine interface and user support using direct manipulation, multiple views on objects and multiple actions on objects.

Introduction

CAD packages for control systems have been developed through a number of years. The commercially available state-of-the-art packages are mostly matrix oriented like MATLAB, but in the last years packages which integrate graphical input/output and simulation languages in an environment have emerged. [4]. Some work has also been done to analyze the design process itself for a better understanding of the iterative nature of most design problems. [2]. However, only little work has been done to combine these two approaches and to produce a package that supports the user better in solving the design problem.

Most of the available packages do not support the user in the iteration process of solving a design problem. They provide tools that can solve parts of the total problem, and the combination of these tools are up to the user. The tools are linear in the sense that they take a description of the system as input and produce an analysis result (f.ex. Bode plot) or a controller (f.ex. LQG-design) as output. It is up to the user to evaluate the results and decide what to do next. In most cases it is troublesome to redo the analysis with a small change in parameters to test out new ideas. A problem specified by constraints in several domains (f.ex. time and frequency) is not easily handled. Furthermore, the results of the analysis are normally presented in one domain and the users possibilities of interacting are also restricted to one domain.

The work reported in this paper is done as part of the cross-departmental project at the Technical University of Denmark in Automatic Control. The project involves 5 institutes in 4 different engineering sectors. One of the achievements of the project has been the provision of a common hardware and software platform for the participating institutes. This enables them to cooperate to a greater extent than before.

The project has at its disposal 5 Apollo workstations distributed at the participating institutes, connected via a network. As a common software platform for CAD of control systems PRO-MATLAB and ACSL have been purchased. Furthermore 13 subprojects involving the different institutes has been defined and a number of those has been started.

One of the subprojects is called 'User-Friendly Interface Construction' and it focuses on the construction of a prototype system to be used for testing a number of interface design concepts. The system should provide the user with better support in coping with the iterative nature of the design problem and enable him to get a better overview of the problem to be solved. The status of the work in this subproject at the Institute of Automatic Control Systems is presented in this paper.

The concepts described in this paper are:

- Extensibility and Integration through the use of objectoriented programming.
- Man-machine interface and user support using direct manipulation.
- Multiple views on objects in different domains.
- Multiple actions on objects in different domains.

Object-orientation

An object-oriented program is based on objects, which contain both data and the code to manipulate these data. A class is the basic data structure definition for an object, along with the functions used for manipulating instances of that object. The class is the template for instances of the object. The programmer can define classes and hierarchies of classes. A call of a member function can depend on the actual type of the object (even when the actual type of the object is unknown at compile time).

The concept of integration is based on the use of already commercially available programs for solving parts of the design problem. F.ex. PRO-MATLAB for analysis and synthesis and ACSL for simulating systems. Integration of these subsystems is done in such a way that the total system presents itself to the user in an uniform manner, but the user should still be allowed access to the different subsystems in a transparent way.

Extensibility of the system is seen as the ability to integrate new methods in to the system in an easy way. The user should be able to integrate his own methods effectively, maybe at different levels of complexity as in MATLAB.

The system could in a way be seen as growing by itself. The advantages are that system administration and maintenance could be minimized and that merging different user environments could be done by the users. However care must be taken to make some garbage collection from time to time.

Often cited benefits of using object-oriented programming are [6]:

TH0270-9/89/0000-0035$01.00

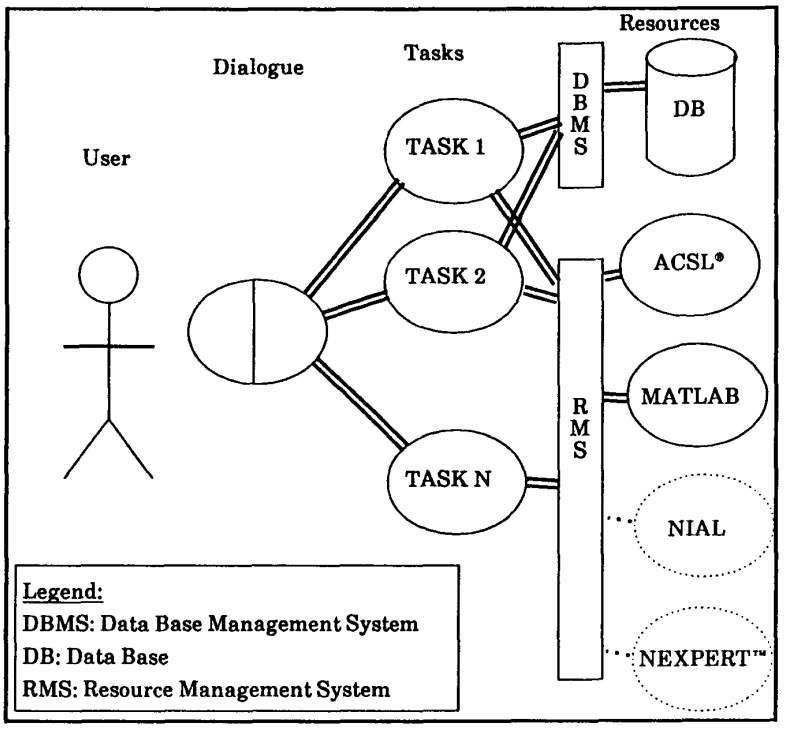

Figure 1.

- Programs can be developed more quickly and inexpensively than with standard programming techniques
- Programs are more modular and lend themselves well to structured design
- New objects can be easily defined and added to the system without impacting the established part of the system
- Testability of objects can be built into the objects themselves
- Integration of systems is easier and
- Objects and methods may be re-used by others.

Thereby object-oriented programming supports the concepts of extensibility and integration. Object in the system are transfer functions, matrices, collections of subsystems, etc. Each object has a content and some attributes. The content is the coefficients of a transfer function or the elements of a matrix. The attributes of an object is owner, creation date, graphical representation, etc.

Objects always know what to do themselves. If an instance of the class 'transfer function' is told to draw a Bode plot, it uses the bodeplot class function (which passes the coefficient to MATLAB and tells MATLAB to draw a Bode plot). If no class function for Bode plots exists an error message is produced. It is now easy to see how the modularity comes into play. If a new class called transfer function matrix is needed it may be derived from the transfer function class, and many of the class functions may be re-used or new may be written. This can be done without having to change the original system.

The prototype system is implemented using the compiler based C++ programming language in an UNIX environment on the Apollo workstation. Phaal [5] used Smalltalk on a SUN to develop an object-oriented system for Control System Design. However, Phaal finds that Smalltalk is too slow to make the design environment suitable for real work in its present form.

Resource management

On figure 1 a blockdiagram of the system is presented.

The user is in contact with the system through the man-machine interface shown to the left. This is described in more detail later. The user can set up various tasks to be performed, either once or continuously, f.ex. the monitoring of an object. The actual calculations are performed by so-called resources shown to the right in figure 1. The resources are different packages. MATLAB and ACSL are the central resources of the system, but other resources such as an expert system shell could easily be connected.

The easy integration of new resources is one of the key features of the system. This feature expands the lifetime of the system as new software can be incorperated when available. This easy extensibility is accomplished by viewing the resources as objects, and letting a resource manager supervise the use of resources. It should be mentioned that resources can be running on the same machine as the rest of the system or at another machine connected via the network using TCP/IP. This feature enhances the total flexibility of the system, regarding both computational load and economy, as licenses for all resources not necessarily have to be available for all machines running the system.

The resource manager handles the communication with the actual resources through UNIX-pipes, and performs the administration if several tasks need access to a single resource. A resource object has several class functions. A constructor function that checks for availability, sets up communication pipes and starts the resource. A destructor function that closes down operation of the resource. Get and put functions perform the actual communication. Furthermore, there are some utility functions such as statistics monitoring, error functions, etc.
At this moment no interpretation of the commands is done in the resource manager. It just sends the commands on to the resource, receives the answers and returns it to the requesting task. In the future some command interpretation should be put into the resource manager to enable it to perform new operations, such as adaptive load optimization, that is, the ability to start and manage several copies of the same resource if load demands it, and the ability to decide which resource should be used if several different resources can perform the same operation.
The database manager in the prototype is quite simple, but it could be extended to incorporate object orientation. Some work has been done in the area of databases for CACSD packages [3].

Man-Machine Interface and User Support

The man-machine interface is based on direct manipulation. Direct manipulation refers to interfaces having the following properties [1]:

- Continuous representation of the objects of interest.
- Physical actions or labeled button presses instead of complex syntax.
- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

The interaction is mainly done through graphics using the mouse as input media. The interface is specified using an interface design program available from Apollo Inc. called Dialogue. The screens shown in this paper are produced with the first prototype system that used Domain/Dialogue and the Display Manager from Apollo Inc. Since then Apollo has developed an new package called OpenDialogue which runs under the X Window System and is portable to other machines. The new version of the prototype system uses this interface design package.
Another aspect of using OpenDialogue and the X Window System is that since X is serverbased the client program can run on a different machine as X server. This adds another distribution level to the system: since resources can run on one machine, the system itself on a second and the interface graphics etc. on a third. This adds flexibility to the total system.
OpenDialogue is a declarative interface design program that uses events to trigger actions in the interface or in the application. The specification of the interface consists of two parts: an application oriented part and an user oriented part. The application oriented part contains the specification of the tasks which the application can handle. The user oriented part contains the specification of the graphical interface layout with menus, scrollbars, buttons etc. This last part can be changed without having to recompile the whole system. The advanced user can therefore specify his own favorite interface layout.
OpenDialogue has a number of build-in objects f.ex. menus, mouse sensitive buttons, graphics areas. These are known to the system and may be used directly in the interface design. However, user specification of interface objects is also possible. The user can extend the known interface objects by defining the new ones in C++ and integrating them in the OpenDialogue system.
A screendump from the system is shown in figure 2.
The system main window is show to the left. The model loaded into the system is a model reference adaptive system. The two top rows of icons are mousesensitive buttons that enables the user to perform different tasks. A button can be selected by pointing to it and pressing the mousekey.
Every function has a number of options defined for it. There are three levels of defaults for these options. The bottom level is the hard-coded defaults of the system. These are used if defaults at a higher level is not present. The middle level is user specific defaults. These are stored in a user specific file and loaded when the system is started. These defaults can be configured by the user. At the top level there are session specific default which can be used temporarily in a single session. These can also be loaded and stored in user accessible files. The three level default scheme gives the user a lot of flexibility, and enables the user to setup a favorite environment. The user hardly ever has to alter options for functions from the defaults. The options-popup is displayed when mousebutton 2 is pressed over a function icon. A helping text for the function is provided when mousebutton 3 is pressed.
In complex and changing environments the need arises for some kind of command-map that displays the connections between different command sequences and displays which commands are applicable at this state of the design process. This facility will be implemented as a resource in the future.
The objects in the blockdiagram can be opened by pressing a mousekey and the content (data) of the object can be examined and altered. The attributes of the object can also be viewed by pressing another mousekey.
The two windows to the right on the screen are normally not open. They are windows that display the communication with the connected resources, in this case the simulation language ACSL and the package PRO-MATLAB. The windows would normally just be icons like the three at the bottom right. These are a network mail facility, an on-line manual facility and a print facility. The X Window System clients

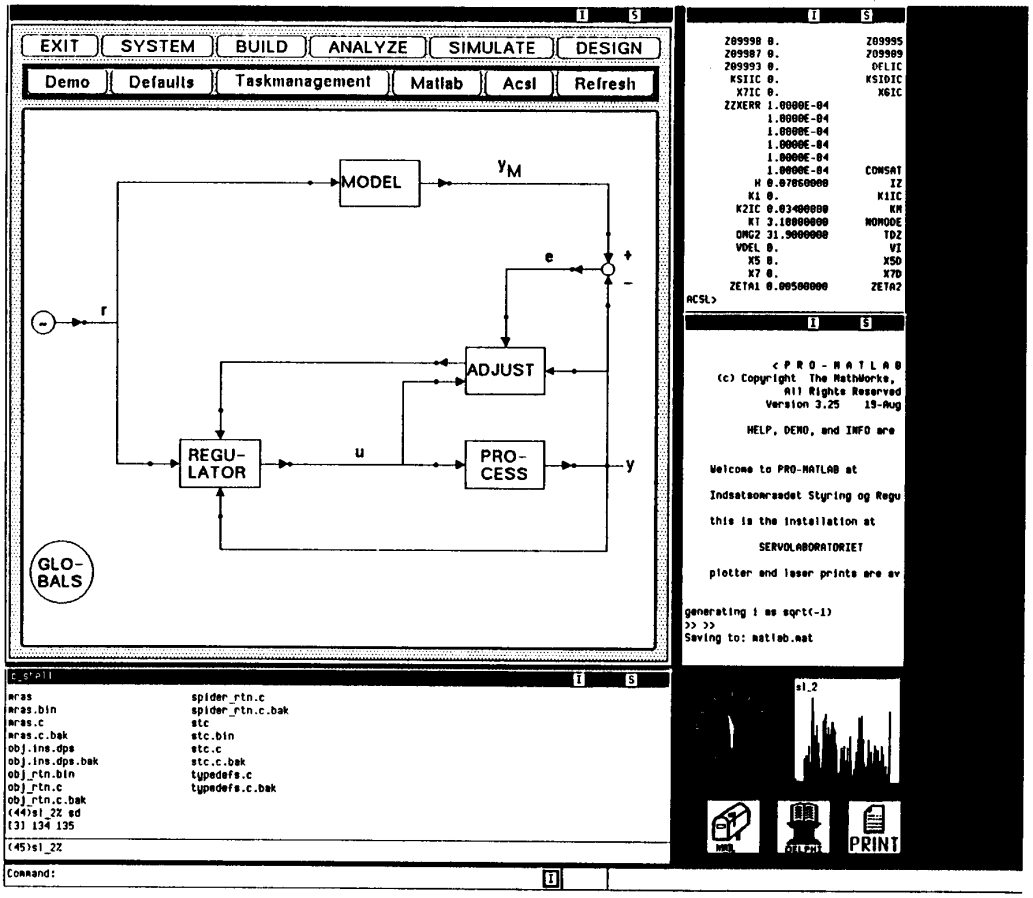

Figure 2.

xclock and xload are also shown. In this way the normal UNIX-operating system facilities are available in a similar way as the system facilities.

Multiple views

One of the main facilities of the system is the ability to define a view on an object. This view can be graphical or alphanumerical. For instance if the object is a transfer function then a Bode plot of that transfer function is a graphical view on that object. A display of the values for the poles and zeroes would be an alphanumerical view on the object. Views can be defined to be static or dynamical. The dynamical views can be updated at a regular interval or update can be triggered by some event in the system. The facility of using dynamical views offers the user good possibilities to check the action he performs in any domain he chooses, not necessarily in the domain in which the action is performed. The user can define multiple views on an object, that is, define views in different domains. These views can be updated each time the user makes an action in one of the domains.

An example of a design problem where this feature gives superior overview is the design of a PD-controller. In figure 3 three views are shown: to the right a closed loop stepresponse, below the polplacement of the closed loop system and to the left a Bode plot of the open loop system. The controller phase is also plotted. Two alphanumerical views are provided (but not shown in the figure), they are the gain and phase margins of the system. These views are just examples, many other views could be thought of. New views can be added or changed at any time during the design process. Each time the user makes a change to the controller parameters the views are updated and the user is able to monitor system performance in both the time-, frequency- and polplacement domains at the same time, without having to enter tedious command sequencies.

The concept of multiple dynamical views is one of the main features to be tested in the system. In earlier packages the user has to either write a command file with the commands to be repeated or re-enter the commands to produce the plot each time change to the controller parameters is made. In practice it will enhance both the users overview (as no superfluous commands need to be entered) and the time consumption for the total design process. This may be because, the time from a change is made to the result can be evaluated, is minimized.

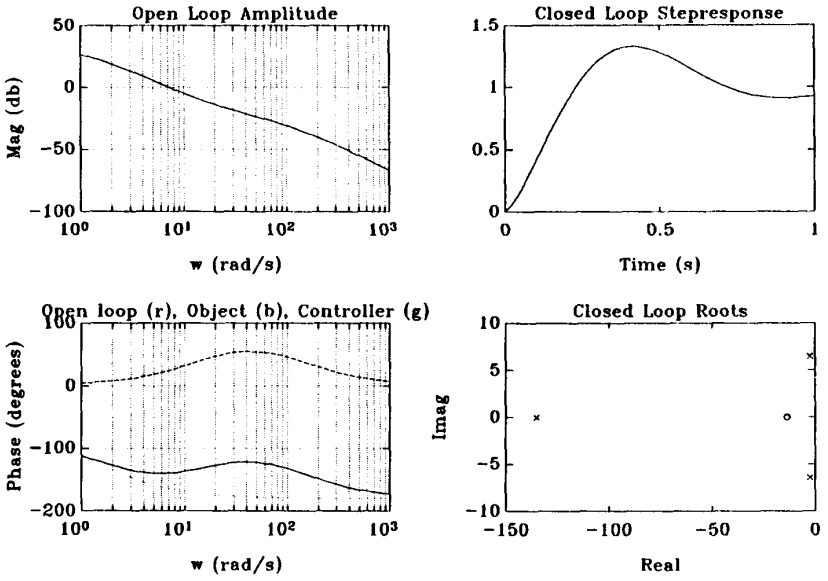

Figure 3.

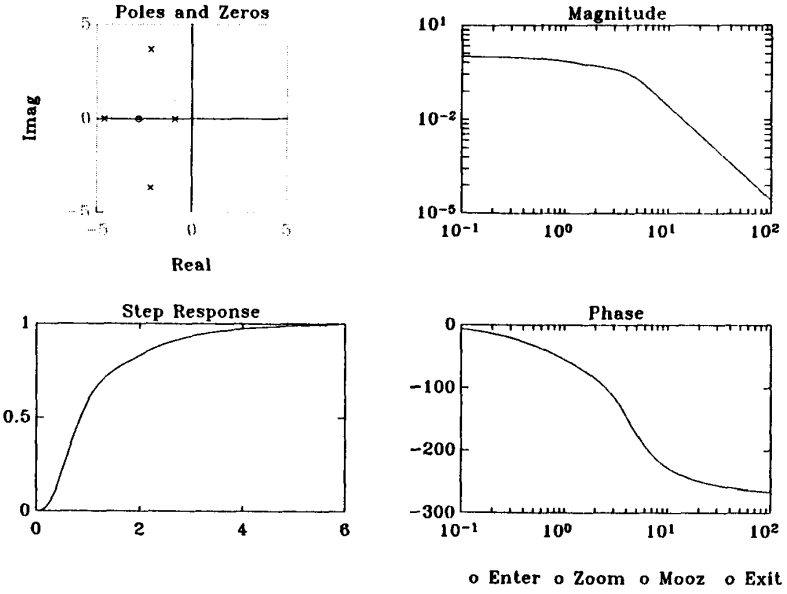

Figure 4.

Multiple actions

Actions are the users possibility to change the system. Actions may be graphical or alphanumerical. The graphical actions are the most efficient when it comes to testing ideas quickly. In the example with the PD-controller the user can graphically change the gain of the controller by pressing the mousekey and pointing to the point in the amplitude plot where the open loop gain should be. The position of the zero in the controller can also be specified graphically. The differentiation bandwidth or the polplacement may be entered alphanumerically. The ability to make key-actions graphically greatly enhances performance of the system.

Another important feature of the system is the possibility to make multiple actions. That is change parameters in different domains at the same time. As an example of how this feature works a graphical transfer function specification is presented. Three views on the system can be seen on figure 4. The pole-zeroes in the complex plane, a stepresponse and a Bode plot.

Actions can be made in either the pole-zero plot or in the Bode plot. The user can add or delete poles and zeroes in the complex plane, or change gain or poleplacements in the Bode plot. This example is used to train students in seeing how the positions of poles and

zeroes changes a systems stepresponse. Especially in showing of the effect of the positions of the zeroes a system like this is helpful.

Conclusion

The paper present a system for testing different concepts in CACSD. Especially the use of multiple views and actions in combinations with graphics is seen to enhance the users ability to get an overview of the system to be designed. Good support for iteration is provided and the short time between action and presentation allows the user to evaluate actions quickly. Object-oriented programming has been used to provide modularity and encapsulation.
This paper has presented the current status of the project and some ideas for future enhancements.

Acknowledgements

The author wishes to acknowledge the work done by M.Sc.E.E. Jesper Pedersen to implement the first prototype version of the system in his master thesis. The screendump presented in this paper are produced from this system. Acknowledgements are also given to Jesper Heurlen for his work with the resource management system and for the example using multiple actions. This work is also done in his master thesis.

References

[1] Hutchins, E. L., Hollan, J. D., Norman, D. A. (1986). Direct Manipulation Interfaces. User Centered System Design.

[2] MacFarlane, A. G. J., Gruebel, G., Ackermann, J. (1987). Future Design Environments for Control Engineering. Proc. of 10'th World Congress on Automatic Control, Munich, FRG.

[3] Maciejowski, J. M. (1988). Data Structures and Software Tools for the Computer Aided Design of Control Systems: A Survey. Proc. of 4'th IFAC Symposium on Computer Aided Design in Control Systems, Beijing, P.R. China.

[4] Munro, N. (1988). ECSTASY - An Environment for Control System Theory, Analysis, and Synthesis. Proc. of 4'th IFAC Symposium on Computer Aided Design in Control Systems, Beijing, P.R.China.

[5] Phaal, P. (1987). An Object-Oriented Environment for Control System Design. Ph.D. Thesis, Cambridge.

[6] Witten, S. (1987). Object-Oriented Programming. TC Interface, Nov/Dec 1987.