Requirements for user interaction support in future CACE environments

Ravn, Ole; Szymkat, M.

Published in:
Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design

Link to article, DOI:
10.1109/CACSD.1994.288902

Publication date:
1994

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

Requirements for user interaction support in future CACE environments

O. Ravn

Technical University of Denmark
Institute of Automatic Control Systems
Building 326
DK-2800 Lyngby, Denmark
or@sl.dth.dk

M.Szymkat

St.Staszic Technical University
Division of Control Engineering
al.Mickiewicza 30
30-059 Kraków, Poland
msz@earth.ia.agh.edu.pl

Abstract

Based on a review of user interaction modes and the specific needs of the CACE domain the paper describe requirements for user interaction in future CACE environments. Taking another look at the design process in CACE key areas in need of more user interaction support are pointed out. Three concepts are described through examples, dynamic data access, parallel evaluation and active documentation. The features of existing tools are summarized. The problem of how easily or 'naturally' the novel concepts are integrated, is stressed.
Keywords: User interface; Programming; Design Process; Active documents;

1 Introduction

Future CACE environments should provide the support for the next generation of styles of interaction between the designer and computer. The traditional concepts are based mainly on ideas adopted from universal programming languages (script files, function files, data objects). There is a need to include the designer as a cooperating factor in design process.

The objectives related to novel interaction styles depend on environment capabilities, and to some extent on the preferred type of user behaviour. In general the approach to building model descriptions should enhance systematic treatment of model representations through standardized programming techniques. Their functionality should support protection of the consistency of model descriptions, standardization of retrieval, updating and transformation methods, including construction of variant versions.

0-7803-1800-5/94/$3.00

Achieving these objectives requires standardization of model data modification procedures (replacing the traditional read-eval-display loop). In the specific context of CACE tools the requirements for user interaction support are closely related to the iterative nature of the design process. In particular the design decisions should be automatically logged to enhance reverting operations, variant development, multi-threaded dialogue, and active document features of the user created modules.

The crucial role in supporting user interaction is played by the user interface which should be able to implement certain general mechanisms as interleavability, concurrency, reversibility and repetitiveness. The paper includes a short overview of currently available techniques in interactively oriented packages.

The general model of user interface consists of the descriptions of several elementary phases following each other. These are [5]:

- user action (mouse cursor movements, keyboard text input, etc.),
- interface feedback (highlighting the selected area, opening the dialog box etc.),
- changing interface state (setting the values of interface control variables),
- initiating the computation tasks (executing assigned function callbacks).

The consecutive phases mentioned above involve only primitive elements of the the user interaction. Usually the dialog between the user and the interacting software tool is accomplished on-line (conversational interaction) or it may be pre-programmed (programmatic interaction), [6].

The *interaction modes* define the appropriate communication protocol for user-software information exchange. The most used ones in the today's CACE environments are combined modes including menus, forms, command languages and various forms of graphical manipulation. These heterogeneous solutions, typically implemented within one of the GUI standards, e.g. MS Windows or X-Window System, are often referred to as *window-based* interaction mode. It is important to stress here that only the most outer layer of the user interaction support, including the appearance (look-and-feel) of the interface devices is defined by the GUI. The essential part of the interface construction, related to the 'usability 'of the CACE software, relies on the bindings between modelling and computational services and the user interface itself. This area will be called the user interaction support.

The conventional user interfaces of currently available CACE tools are typically built around conversational and programmatic concepts which are well established in almost all types of existing application software, [6]. Most of the *user actions* supported by the interface involve [1], [2]:

- access to model data and computation results (via querying or browsing).
- manipulation on model data and computation routines (parameter setting, etc.),
- instantaneous configuration of the software (affecting overall working context),
- various consistency checks,
- program state retrieval and creation of execution reports (logging).

The prevailing part of existing CACE tools supports the *sequential interaction style*, ie. a type of the dialog where the user actions have to be organized in a certain ordered manner. The navigation through the cascade of menus is a basic example, the execution of sequences of commands serves as another one. The repetitions or parallel dialog threads are often available, but not directly supported. The *asynchronous interaction style*, where many tasks are at the user disposal at the same time and sequencing within one task is independent of sequencing within the other is an alternative. In what follows we will try to precise what kind of interaction style would be preferred in the future CACE environments due to the specific properties of control systems design.

2 CACE specific requirements for the user interaction support

As pointed out in an earlier work [7], [8], [2], looking at the nature of the design process gives insight into the needs of the user of CACE systems. In order to determine potential focus points for future work on refining and enhancing user interaction support the design model in Figure 1 is revisited. It should also be pointed out here that the problems of user interaction relies on the existence of good and well proven numerical methods for the underlying analysis and design. These tools are to a large extent available today but the problems to use them efficiently. Here an enhanced user interaction support will benefit the overall efficiency of the CACE environment. Many of the problems and questions asked in the development of user interaction support are common to many domains of engineering, e.g., circuit design, digital filter design, machine engineering. Especially software engineering has a large common problem basis with CACE [8].

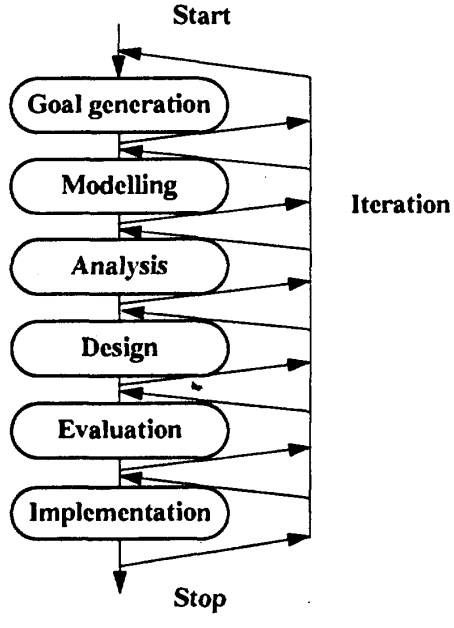

Figure 1: A simple view on the design process.

The design process model shown in Figure 1 is a very simple abstraction but still it provides a way of classifying user actions during the design. The model consists of 6 phases:

- *Goal Generation.* This phase initiates the design process. The problem and the desired features of the solution are determined. This phase is normally done in cooperation with the customer, other engineers etc. Normally no formalized tools or methods are used here.

- *Modelling.* The modelling phase is used to determine a model of the system to be controlled. This is normally a mathematical model which can be used by the tools of the following phases. Models of different complexity may be derived such as linear plant models for the design of linear controllers and then non-linear plant models in the evaluation phase. Many CACE tools exist for assisting the user during this phase.

- *Analysis.* The derived model is analyzed in order to gain an understanding of the system and the potential problems. The analysis results are used as a basis for choosing a controller structure. Not just the normal numerical tools are applicable in this phase; the potential benefits of using symbolic manipulation tools are becoming more and more evident and many of the numerical packages have built-in symbolic tools or interface to them.

- *Design.* A possible controller structure is selected and the parameters are chosen in order to match the design goals. It may be useful to consider more controller structures and compare their performance in parallel. Many tools for designing standard LQ, LQG etc controllers exist.

- *Evaluation.* The different controllers are considered in this phase and compared with respect to the features of the desired solution set up in the first phase of the design process. The degree of compliance with the goals is determined and the best controller selected. The evaluation phase may use simulation of the system or use partially the real-time interface in order to select the best controller. More models may be used in order to gain insight into what features of the system and the controller limit the performance.

- *Implementation.* The chosen mathematical description of the controller is implemented. More and more tools are emerging in this field. The standard packages have C-code generation tools and offer hardware which can be used for testing the controller in a laboratory environment. The main problem here is the balance between code efficiency, hardware dependency and the degree of automation of the phase.

Another element of the design process model is the *iteration* which is its fundamental property. The iteration can be performed manually, semi-automatically or automatically. The iterative nature of the design process is also an important element which we will return to.

An overall evaluation of the design phases indicates that most CACE tools are available for the Modelling, Analysis and Design phases. Some tools are also available for the Implementation phase. However there is a lack of tools for the rest of the phases and the iteration. Some environments being developed at universities, [4], support the iteration but these in turn are not generally available.

In Figure 2 another view of the design process is shown [3]. Here the mathematical abstraction level is more apparent. The modelling and implementation phases represent transitions between the physical structure and the model structure level. The analysis and design phases appear as transitions between the model structure and the control structure level. CACE tools are generally available for the more abstract levels and not so much for the lower levels.

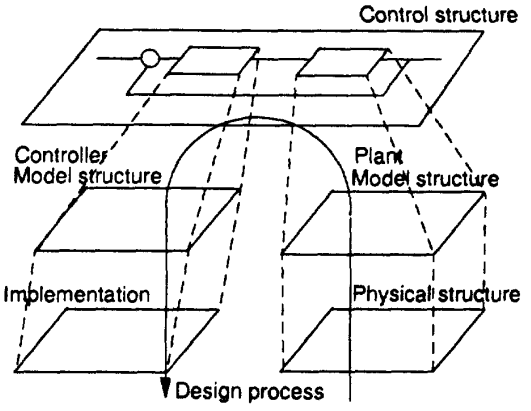

Figure 2: Another view on the design process.

Looking at the design process indicates that there is a strong need for user interaction support in the more 'soft' areas. These are Goal generation, Evaluation and iteration. The iterative nature of the design process gives rise to more explicit automation of certain *design loops*. On the other hand it may be difficult to decide beforehand which decision on certain stage leads to the successful design. This would require a support of parallel or variant design procedures.

It is important to define the qualities required from the interaction support which are needed to meet the needs of the control system designer. We will concentrate on the following ones:

- *interleavability* - meaning that the user can interrupt one task, skip to another task, come back to the first one, and so on, with possibly many other tasks,
- *concurrency* - meaning that several tasks may be simultaneously (or virtually simultaneously) executed,
- *repetitiveness* - meaning that certain tasks may be recorded and replayed when needed by the user,
- *reversibility* - meaning that the state before the task execution may be fully restored, independent of the results.

All the requirements specified above are directly related to the support of variant design ('what if' scenarios, etc.). In general, we expect that future CACE environments will be able to perform simultaneous visualization of the results of the same design parameter change in alternative control system structures.

In what follows we will consider also even more demanding requirements related to the *intelligent interface* definition. These requirements involve probably more the user interaction support as a whole than the user interface itself. They go much deeper into data representation and the application domain. We will recall here the 'spreadsheet metaphor'. In the automatic recalculation mode the manipulation on specific cells produces 'immediate' adjustments in row or column sums. In general, this concept complies withe the idea *active document*. In fact the user defines certain relations between the manipulated objects, and the software is able to update the context automatically.

In the case of control system design a similar situation appears whenever we have parameterized design schemes. Let us specify the requirement concerning the user action support related to this kind of software behaviour as *meta-programming*. The explanation of this term is as follows. In order to obtain the desired effect the user should be able to create his own two-way bindings between the interface, data management and computation layers. This may be done explicitly or it may be deduced from the usage context, provided that general rules of meta-programming are known. The simplest example is the command initiating re-execution of a certain set of previous commands in another workspace context.

3 Examples

Three examples of novel user interaction elements are described below. These elements can be implemented with the current standard packages, but the key issue is the ease with which they can be used by even less experienced users. The availability and the degree of integration is also of paramount importance when evaluating the usefulness of the suggested elements. The following examples implemented using current tools were intended to illustrate possible benefits that should be easily accessible in future CACE environments.

3.1 Dynamic Data Access

The concepts of control systems as objects, and views and actions on them have been described earlier [7]. Here the main ideas are recaptured in Figure 3.

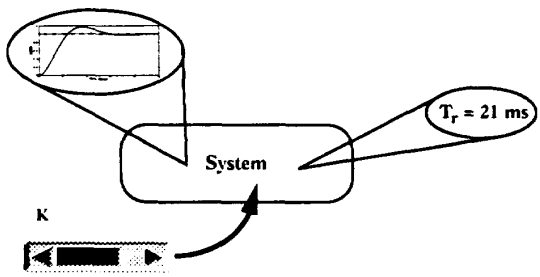

Figure 3: Dynamic views and actions.

The control system is an object in the CACE environment, and multiple views can be setup relating to it. These views can be graphical or alphanumeric and contain analysis results such as a step response and the rise time. The updates of the views are triggered by events, either explicitly as a request for update or as an action being made. The action on the system (object) can be, e.g., changing the gain of the controller graphically. When the action and the update of the views are linked the operation of the overall CACE environment is like direct manipulation. For complex systems and many views the currently available computer resources are likely to limit the performance. In that case an asynchronous mode should be used as even a small time delay in the direct manipulation limits the usefulness. Some aspects of the above ideas can be found

in SIMULINK and ANDECS [4]. The idea of this concept is to support the user in the iteration of the design process in order to facilitate extensive experimentation to help get a better feeling of the system. Some optimization tools coan be coupled with the iteration in order to achieve the design goal automatically or semi-automatically [4]. A key point of the concept is the ease with which the views and actions can be setup and modified, there is a great need for some form of maybe graphical meta-programming. In any case such an implementation should give better interaction possibilities to the iteration than a conventional script file.

3.2 Parallel evaluation

Another phase of the design process where there is a need of user interaction support is the evaluation phase. Experimenting with more controllers in parallel will ease the evaluation. The performance of the controllers can be viewed at the same time thus making it easier to see the benefits of, e.g., an LQ controller versus a PID controller for the same system. Figure 4 illustrates the parallel evaluation concept for different controllers and Figure 5 the same idea for system models of different complexity.

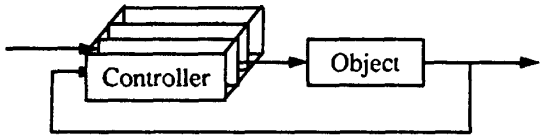

Figure 4: Parallel evaluation of different controllers.

With such a feature in the CACE environment the effect of, e.g., a limit on the control signal would be easily found in one experiment. Figure 6 shows the control and output signals of the system with and without the limitation.

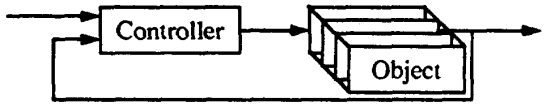

Figure 5: Parallel evaluation of different model representations.

Again the key issue is the degree of integration of the concept into the CACE environment.

3.3 Active document

A concept which finds some use in text processing is active document. A text processing system such as FrameMaker has built-in variables and an interface enabling the invocation of external programs. The usefulness of active documents in user interaction support is best shown by an example. A test robot has been built in the laboratory, the controllers have been designed and implemented. For some reason the gear ratio in one of the axes should be changed and the controller redesigned. In some current CACE system [4], [9] there is a database storing the history of projects. From this database the design calculation could be reconstructed. However if the documentation of the robot and the controller were written as an active document and given an engine for updating the document (similar to a spreadsheet) the redesign could be done automatically after the value of the gear ratio had been changed.

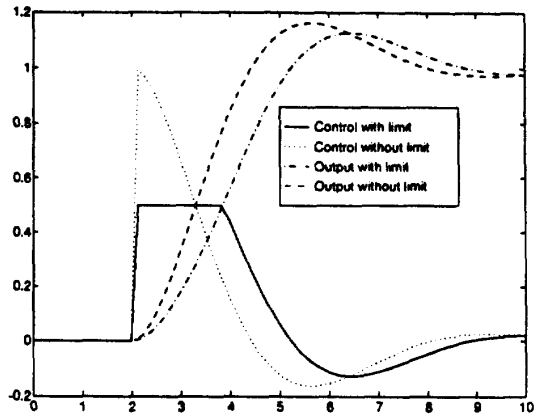

Figure 6: Control signal and output signal for the parallelly evaluated system.

Active documents and hypertext are used with success in other domains such as network communication e.g. the 'Mosaic' client for the Wide World Web where data is retrieved and displayed in correct format by clicking on a hypertext area. The user does not have to know the actual site where the data is stored, to connect or unpack it. This gives the user superior interaction possibilities and a good overview of the data accessible in the system.

4 Currently available solutions

The popular GUI environments such as X-Window System or MS Windows offer a variety of tools and mechanisms for developing user interfaces, e.g., X-

toolkits, Motif, Openlook, Visual C++ and many others, all at low to medium levels of implementation. There are also available more high-level oriented development systems such as Bricklin's Demo, HyperCard, SmithersBarnes Prototyper, [5]. In fact due to their generality, which is by no means a deficiency, their are loosely related CACE.

The need for the more powerful domain-specific user interaction support tools has already resulted in certain evolution of existing CACE environments, just to mention: Matlab v.4.0 - Handle Graphics, Simulink's - interactive simulation concept and meta-programming features, configurable GUI of Xmath or Mathematica's notebooks. We do hope that some of the ideas presented in this paper may influence the future developments in this area.

The general remark which applies here is that most of the tools give only low level interaction support. This seems to be sufficient for software developers implementing the CACE tools rather, than to application domain oriented users. On the other hand the latter category seems to be able to verify the usablity of the user action support.

Another important issue which should be addressed here is the general problem of the standardization of the user interaction support which would require a serious collective effort of the CACE community.

5 Conclusion

Much of what has been presented here is available using existing CACE. The real problem is how easily or 'naturally' it is achieved. Sometimes it is just a matter of interaction or programming style within the given tool. In many cases external user interface management systems would be useful. The worst thing (for the user) would be to redesign an existing tool. As is generally agreed the nature of computer software is evolutionary. One may hope that some of the requirements proposed here will be accepted by the developers of future CACE software.

Acknowledgment

M. Szymkat acknowledges the support he received under the grant KBN-8-85289102: Computer systems of control and decision making - theory, formal tools and computer aids.

References

[1] H.A.Barker, M.Chen, P.W. Grant, C.P. Jobling, P.Townsend: An open archtecture for Computer-Assisted Control Engineering. *IEEE CSS Workshop on Computer Aided Control System Design - CACSD'92*, Napa, USA. March 17-19, 1992.

[2] H.A.Barker, C.P.Jobling, O.Ravn, M.Szymkat: A requirements analysis of future environments for computer-aided control engineering. *12th IFAC World Congress*, Sydney, Australia, 19-23 July 1993.

[3] A. Christensen: Models of Control Design – Modelling and Validation in Ship Control. Ph.D. Thesis. Technical University of Denmark,1992.

[4] G. Grübel, H-D. Joos, R. Finsterwalder, M. Otter: The ANDECS design environment for control engineering. *Preprints 12th IFAC World Congress*, Sydney, Australia, 1993.

[5] D.Hix, H.Rex Hartson: Developing user interfaces - ensuring the usability through product and process. J.Wiley, New York 1993.

[6] G.Kappel, A.M.Tjoa: Graphical user interfaces for object-oriented data base systems. *Cybernetics and Systems Research 1992*, Vienna, Austria, pp.1247-1254.

[7] O.Ravn: On user friendly interface construction. *Proceedings of the IEEE CSS Workshop on Computer Aided Control System Design - CACSD'89*, Tampa, USA. 1989, pp.35-40.

[8] O.Ravn, M.Szymkat: The evolution of CACSD tools - a software engineering perspective. *Proceedings of the IEEE CSS Workshop on Computer Aided Control System Design - CACSD'92*, Napa, USA. March 17-19, 1992, pp.225-231.

[9] J.H. Taylor, M. Rimvall, H. A. Sutherland: Future developments in modern environments for CADCS. *Proceedings of the IFAC Symposium on Computer Aided Design in Control Systems*, Swansea, UK, July 15-17, 1991, pp.51-62.