Principles of CASE Tool design for Automation Control

Dr. Wolfgang Brendel
infoteam Software GmbH
D-91088 Bubenreuth, Am Bauhof 4, Germany
phone +49-9131-78 000, fax +49-9131-78 0050
e-mail 100024.1231@compuserve.com

Abstract

Twelve principles which should be obeyed by every designer of CASE-tools for control automation are postulated, in order to avoid that a post-modern computer scientist with fancy ideas neglects the requirements of users who are much more conservative regarding taste, attitude and practicability.

Much more importantly, PLC-programming has to be different from conventional software development because of the different kind of people who are involved in that task. The purpose of the following statements is to give some hints as to which principles should be obeyed in designing a PLC-Programming system.

Giving examples from the implementation of the Open Development Kit, an interactive PLC-programming support environment conform to IEC 1131-3, we also give guidelines as to how to implement these principles in a real world environment. We close with first experiences from several hundred users and give an outlook to further topics both in research and development for the near future.

Keywords

IEC 1131-3, PLC, programming, CASE-tools, OPEN DK

1 INTRODUCTION

To enhance the performance of the Software Engineering Process, it is required to shorten the development cycle of new automation control software. This aim is achieved by introducing engineering processes which apply different engineering methods in several phases of the overall design and the development cycle. This is accompanied with the development of CASE-Tools supporting these methods. Nowadays we have a huge bundle of design tools, graphical editors, interactive workbenches, incremental compilers and on-line debugging utilities with one major obstacle: the lack of common interfaces, ensuring a smooth transition from design down to operation and ensuring reliability of an automated manufacturing system.

That's why we now have to pursue two objectives:

- to reduce the number of tools involved as far as possible and
- to standardize the interfaces between these tools.

The first target can be reached by incorporating tasks like code generation in design tools used at an early stage of the engineering process, eliminating the need for the user to occupy himself with compilers and generated code in detail.

To reach the second aim - to standardize interfaces between heterogeneous tools - is much more difficult, because it requires the co-operation of tool suppliers who often are competitors. Nevertheless we know from all sectors of engineering that international standards are mandatory for technical progress. So there is no alternative but the standardization of interfaces.

2 CASE-TOOL ARCHITECTURE FOR CONTROL AUTOMATION

Which CASE-Tool design will meet these objectives? The ultimate judge will be the engineer who has to perform a most complicated task with a fixed deadline: the day the manufacturing system he designs has to be up and running!

In order to design a flexible architecture of software tools for control technology, infoteam made an analysis of user requirements in the field of PLC-programming and control automation. Based on these results, we found out that principles can be stated for the development of CASE-Tools which shall ensure that a software tool will match the requirements of the end-user and, at the same time, will fulfill the two overall objectives as mentioned above.

The key to an open architecture for control automation tools is to define a common framework in which heterogeneous tools can cooperate and to develop an Application Programmer Interface - the so called Open-Software-Link „OSL“ used by developers of tools for automation control programming.

The Software-Industry market is dominated by de facto industry standards in areas like operating systems and database technology as well as programming languages and compiler design. This eases exchange of information and technology.

Industrial Control Automation is one of the few exceptions: Until recently there was no world-wide standard at all. Each PLC-manufacturer had its own style of programming tools and even worse, depending on the country of the company each control system was programmed in a different programming language:

- in France, Spain and Italy using GRAFCET
- in the United Kingdom and the United States using Ladder Diagrams
- in Germany, Austria and Scandinavia using Function Block (FB) Diagrams
- and in all countries some dialects of Instruction List and other languages.

Principle 1:

*Don't try to convince the user of your favourite programming methodology or programming language. Support the **method or language** the user wants to use.*

Fortunately the emerging international standard IEC 1131-3 defines all five languages in a consistent way. Moreover, the user and vendor organization PLCopen defines compliance rules that have to be fulfilled by the corresponding tools implementing that standard. This is a prerequisite for the standardization of widely accepted interfaces between CASE-tools, interactive editors, compilers, MMI-tools and a common database definition all used throughout the life-cycle of an Industrial Control Automation Project.

The lack of standardized interfaces and common data definitions led to the circumstance that the wheel is reinvented more than once in such an automation project. But even worse, mistakes are introduced and lead to poor quality and schedules which are overdue in most automation projects. Because there is no way to cooperate between tools from different vendors each company invented their own programming support environment.

Users are therefore adamant about the use of standard PCs for programming and hail the standardization of the programming languages by IEC 1131-3 as the breakthrough they have been waiting for.

The logical consequence is the demand for new manufacturer-independent programming tools in accordance with IEC 1131-3, which can be used on any PC to produce machine code for (almost) every PLC and put it into operation.

Working on this assumption and years of experience, the Open DK portable programming system was developed following the principles:

- Using standardized components, such as editors, compilers and management tools
- Adopting components where different requirements ask for different solutions
- Filling in missing pieces by developing special tools or enhancing already existing ones.

There are clearly defined interfaces between all parts of the software, so that they can easily be exchanged for new versions because the interface specifications are upward compatible. In this way, new applications can be added to the system at any time.

Figure 1: Architecture of Open Development Kit

The majority of PLC systems today, however, provide little or no support for the features of the IEC 1131-3 standard. Tools like those of the Open Development Kit can therefore only be ported to a particular family of controllers by adapting or extending their functionality.

Principle 2:

Provide for methods or options to adopt the system to the users needs.

Even manufacturer-independent tools like Open DK need manufacturer-specific code generators and ON-LINE drivers to adapt them to suit the conditions in existing systems. Since this customizing process always affects the manufacturers' own interfaces, porting is only practicable with their co-operation. The advantage is that the independent tools are exactly matched to the control system and perfect functioning is guaranteed by the manufacturer.

3 SUPPORTING THE REUSE OF EXISTING SOLUTIONS

In the past, the various controller families from each manufacturer all had their own programming units with specially developed programming tools. Changing to a different controller meant investing in new software development equipment each time and programmers had to spend their weekends poring over a new manual and getting to grips with new terminology and worse still a new programming language or at least a new dialect.

In switching over to a different supplier, the user has to rewrite large parts of the application. This and the fact that there is no common education in using these tools separated the industry even more.

Principle 3:

*Preserve **upward compatibility** with existing user source code. If this is not possible, supply tools for porting application software to the new environment without loss of information.*

The essence of mechanical or process engineering technology are the algorithms used for solving frequently recurring problems. To enable the engineer to employ the same method repeatedly within an application, IEC 1131-3 makes provision for the creation of instances.

The declaration of an instance of a function block must not be confused with the invocation of a function block. Creating an instance of a controller FB is the same as using several hardware modules of the same type in an automation solution.

Defining an instance is simply the creation of duplicates with identical functions. The introduction of the standard has vastly improved the conditions for extending the development of reusable application software to the PLC sphere.

Principle 4:

*Support at most the **reuseability of existing applications** to an extent as far as possible. The productivity of programmers depends to a large extent on the degree to which he has access to solutions already developed.*

A program organisation unit (POU) should be declared as a function block if it is a program part which is frequently needed and required to be reusable. An FB can have several input and output parameters and - unlike a function - can also store internal data. The values of the output parameters and internal variables are retained when an FB is called. For this reason, invocation of an FB with the same input parameters does not necessarily always yield the same output parameters.

Direct access to the inputs and outputs of a PLC is not possible within an FB. This makes FBs hardware-independent. Any FB that has already been declared can be used in the declaration of another FB or program.

An equally important feature compared to function blocks is the concept of functions! The first parameter of a function is the intermediate result of the current calculation and it always yields a new intermediate result. If the function has exactly one parameter, it cannot be distinguished in practice from an instruction because IEC 1131-3 does not stipulate parentheses for the parameters of functions.

This facility for apparently extending the instruction set of the controller is available not only to the manufacturer but also to the user. Therefore the reusability of solutions may be enhanced defining a functional substitute for a missing instructions on a target systems which doesn't support the full functionality despite the fact that there might be a slight loss in performance.

Clearly, the most important task of the Project Manager is to help the user manage configurations, resources, programs, function blocks and functions. But what about all the other information which is generated during editing, compiling, generation of target code and downloading?

We have the philosophy that the casual user should not be burdened with worrying about where to store those files. This task should be devoted to the Project Manager utility. On the other hand, the experienced user, who has several incompatible PLCs to program will use the same FBs and functions and as far as possible, only generating code for different targets out of the same sources. This user needs a sophisticated tool to support his task.

Furthermore with the emerging standard there will be numerous suppliers of function block libraries which are used as predefined functional units already tested and approved. Using not only its own source but also the geniality of numerous engineers will greatly enhance the productivity.

Principle 5:

*Provide for the possibility of **third party FunctionBlock-Library** support.*

The project manager is a central graphical tool which provides an extremely user-friendly, efficient interface for managing the programs, functions and function blocks of a project. Using the project manager the user can call individual blocks for editing, print them or transfer them to the controller. It also assumes central control of access to these structures by other tools.

Figure 2: Enhancing reusability through a Project Manager

One of the major tasks of the Project Manager is to support the user by structuring his project and to manage program organization units in an efficient way. Project data can be accessed via an open interface, which can also be used by other programs.

The project manager is used for creating and structuring projects. The actions performed with the project manager bear an outward resemblance to actions which can also be performed at operating system level or using a file management program. When you create a project containing branches, a directory with sub-directories is created.

The entire compiling and linking procedure for a complete project can also be controlled using the project manager.

4 DESIGNING THE USER INTERFACE

Surely five programming languages are enough for one subject? Five languages probably, but five editors are definitely not sufficient. In fact, different types of applications call for different versions of the editors, tailored to the special requirements.

This is the reason why we offer two editors for FB-Programming: A configuration oriented CFC-editor and the FBD-editor which was constructed for highly interactive programming.

What is common to all implementations of interactive editors is that all are designed with the highest possible comfort for the user in mind. Comfort means that each task the user wants to perform has to be implemented in a way that user interactions are minimised as far as possible. To phrase it more generally:

Principle 6:

Design for easy modification of designs and programs because 80-90% of tasks are changing already written programs and only 10-20% is writing new code or designing new programs.

Figure 3: Moving function blocks with autoplacement and autorouting

The solution the developer provides to the end user can be easily measured: The cost function is defined as total number of keyboard hits and mouse movements necessary to perform a

given set of alterations on an already existing program. The best implementation minimises this cost function.

Studies of PLC programs in industrial engineering have revealed that only between 10 and 20% of all control programs are developed from scratch. The majority (80 to 90%) are at most updated. This means that most of the time the editors of a programming system are not used for writing or developing programs, but merely for updating existing programs.

Just compare the systems currently available! How many user actions does it take to move the function in the example above to different position? Only three or four user actions are necessary. Very few of the CAD-oriented systems available on the market can manage with less than 20 or 30 actions. Unfortunately, it is not possible to assess the quality of a programming system at first glance.

Much harder to achieve and even more complicated to judge is the requirement of designing sophisticated engineering tools for the casual user or sometimes untrained personnel.

Principle 7:

*Design a system with a **predictable behaviour**. Assure the user that each action that he takes will be checked as soon as possible. Make sure that he can rely on the fact that the system will either accept his instructions or reject them immediately.*

Figure 4: Automatic checked graphics in ladder diagrams

The method of graphical program development by means of repeated transformation of valid, correct graphical structures has been used successfully since the beginning of the 80s. Why should users settle for less today?

The graphical ladder diagram editor only accepts input operations which result in correct networks. User entries are immediately checked as to whether they can be represented in graphical form and unacceptable entries are rejected. This ensures that the current network is always graphically correct.

Programs and blocks entered in the graphical languages can also be displayed and tested in IL. Conversely, programs written in IL can also be represented in a graphical format as far as the graphical facilities allow. This fact is due to the incremental compiling method by which all actions of the user are compiled in transformations on a language-specific working set. The effect of this transformation is shown to the user by the visualisation module.

Figure 5: Graphical transformation using incremental compiling

The same goes for the introduction of the mouse for programming. Users demand from experience that all tool operations should be possible from the keyboard.

Principle 8:

*Design the system to be used with **keyboard-actions** and allow for mouse-control. Not vice versa! With the widespread use of windows oriented systems this feature is not fancy any more, nevertheless it is still important.*

As can be seen with the example of inserting shortage between two rungs, an incremental system does most of the graphical rearrangements by itself!

The graphical editors all employ the operator input concept: *Mark* (object or area) and then *perform the action*. This speeds up inputs and alterations considerably and also enables inputs to be made exclusively on the keyboard as well as by using the mouse.

When marking program parts it is possible in both graphical and textual editors to select single objects, whole areas or the entire program. In the graphical editors, insert operations with objects selected in this way are always subjected to a syntax check.

For single marked graphical objects the method of pre-defined editor actions is used, whereby certain default actions - e.g. double click or space bar - trigger standard operations on the object concerned. These graphical elements with attributes make it easy, for example, to negate or insert a contact or change it into a labelling field.

Full support is provided for the editing functions of the standard Edit Menu (Cut, Copy, Paste, Find, Replace) and the use of the clipboard. For the graphical languages the correctness of the syntax of the program is guaranteed in operations using the clipboard.

All users of interactive systems know that a little bit of uncertainty or selecting the wrong option like „cut“ instead of „copy“ can make the work of some hours all in vain. Modern systems therefore provide sophisticated aids for undoing and redoing actions to recover from the mess we may create under pressure (like writing this article late in the evening).

Principle 9:

An untrained user fears that an action he takes will lead to disastrous effects.

Make sure that each action that has a chance of loss of work already done has to be checked or at least can be made undone.

5 ON-SITE DEBUGGING MADE EASY

For testing and start-up, programs and data need to be exchanged between the controller and individual tools. The function of the communication manager is to present a universal view of the controller, i.e. to conceal the actual PLC from the tools. This enables the tools to be manufacturer-independent and ensures an open and efficiently adaptable interface to the controller.

These logical connections are used, for example, for transferring current I/O values via the on-line interface from the controller to the editors or for loading the PLC program into the controller for remote control. Depending on the functionality of the connected remote device there exist several features for remote control of a PLC:

- Remote control (start, stop, warm restart)
- PLC status display: RUN, STOP
- Program name and versions
- Program comparison PC <> PLC
- Status information
- Monitoring of inputs/outputs
- Force/Set via direct I/O

Using simulation it is possible to run the program in different cycles and to debug the program by a stepwise execution without any hardware attached. This gives a powerful possibility for „black-box-testing“ the function blocks and the program.

A more typical example is the execution of an application while the process and the controller is up and running. Monitoring of parameters of a Function block in a graphical environment can be done in real-time without disturbing the performance of the PLC. Some purists argue that this is not true engineering style but such is life!

Principle 10:

A typical control-application is not designed, it is engineered. That means it is programmed on the plane to the facility where it will be installed. Because real application development is done by trial and error, a programming system must support experimental programming with continuous improvement.

To support the user in this engineering task we have to provide tools which allow a better recognition of binary transient signals and have the possibility to have a powerflow display of binary variables. This is already known from debugging ladder diagrams and a proven technology which greatly simplifies the recognition of what's going on with the process.

It is important to note that such features could not be substituted by pure monitoring of global variables, because the parameters depend both on the calling environment (the program which calls the FB) and the actual instance which is called (the duplicate of a FB which has a unique internal data storage):

Adress	Daten	Online	Anzeigen	Hilfe
0001				
0002				
0003	123	101101	CAL	p1[
0004				IN_P1 := 123
0005	32	101100		]
0006				P2.IN1_P2 := OUT1_P1.
0007	27	001100		P2.IN2_P2 := OUT2_P1
0008				]
0009				
0010				
0011	32	101100	CAL	p2[
0012				IN1_P2 := P1.OUT1_P1.
0013	27	001100		IN2_P2 := P1.OUT2_P1
0014				]
0015	27032	101101		t_int := OUT_P2
0016				]
0017				

hexadezimal Offline ☒ Zustandsanzeige Modifizieren

Figure 6: Online monitoring instances of function blocks in a running process

6 DOCUMENTATION OF APPLICATIONS

A PLC is part of the machinery in which it is integrated, it is not a computer. Because a machine lives from 5 to 10 years, the chance that it has to be modified by someone else, who has no floppy disk with the original program - or at least no computer which can read in those old-fashioned floppies.

The features of the documentation system play a major role in selecting an appropriate system from the engineer's point of view. That is because listings are the only documentation the end user gets.

Therefore sophisticated documentation features are a must and include:

- Complete program listings in graphics for programs, function blocks and functions
- Sub-projects and the project tree, cross reference list
- Declarations, instructions, assignment list
- Compiler results, program comparison results
- Controller configurations, on-line statuses etc.

Principle 11:

*Provide for **re-documentation of the application** solely from the PLC.*

When it comes to documentation of the application the printed documents are intended to be used by personnel who don't know much about the algorithms the implementers used and in most cases are unfamiliar with the programming environment. So if the documentation contains terms or expressions which are unknown to them the documentation may be of no use at all.

Principle 12:

*Use **terms and verbs** which are convenient for the user. Best is to use the vocabulary he or she already knows.*

Because real programmers don't read manuals this applies also to the help files, menus, dialogues and other stuff!

7 CONCLUSIONS

This paper shows principles which should be obeyed by every designer of CASE-Tools for control automation. In applying these principles to modern programming environments infoteam Software created a software package which is widely accepted because of its reliability and its user-friendly interfaces and methodologies.

8 REFERENCES

- Brendel, W. (1994) Objektorientierte Werkzeuge zur SPS-Programmierung. *Automation Precision* 10/94, 10-12
- Brendel, W.; John, K.-H.; Grötsch, E. (1994) BASE LEVEL Certification of IEC 1131-3 compliant products. *PLCopening April 1994*, 6-7

9 BIOGRAPHY

Dr. Wolfgang Brendel studied computer science at the „Friedrich Alexander Universität Erlangen - Nürnberg“ in the early seventies. Special emphasis was on graph-grammars and incremental compilers. He then joined Siemens and was responsible for the development of the STEP 5 programming software tools until 1981. After rejoining the university he worked on hardware description languages and silicon compilers finishing with the degree of Ph.D.

In 1983 Dr. Brendel, with three colleagues, founded „infoteam Software“, a company in the software services industry located near Nuremberg, Germany. Nowadays infoteam is one of the leading manufacturers of PLC programming systems with regard to the new standard IEC 1131-3 and is working in a close cooperation with almost all renowned PLC manufacturers. Developing quality software is a main focus.

Infoteam is a company owned by the employees, which ensures a high degree of commitment between employee and employer.