Object Oriented Programming for Motion Control

By

David E. Halpert, Technical Director
Creonics Operation, Allen-Bradley Company
Lebanon, New Hampshire USA

Abstract: The application of motion control to solve general automation problems has been hampered in the past by the steep learning curve of traditional programming languages. From the application development standpoint, traditional motion control languages demand not only a complete understanding of the details of the application, but also a thorough knowledge of programming concepts before a solution can be implemented. A radically new motion control concept utilizing a graphical user interface and icons to represent motion functions is presented. This graphical motion control language allows free-form motion programming by simply connecting the icons to conceptualize the application in block diagram form. Motion and process details can be filled in later. Once the diagram is complete, it is converted into a text file in the native language of the particular motion controller to be used and downloaded for execution.

The Need for Motion Control Software

Today's manufacturing needs are generating requirements for greater speed, greater efficiency and affordable cost in many types of machinery and for many types of applications. An industry-wide desire to reduce capital equipment costs and at the same time increase the variety of products available has stimulated a requirement for so-called "flexible manufacturing" capabilities. This need has caused a rethinking of machine design and process control in newly engineered systems as well as providing the impetus to retrofit existing equipment with new functionality in motion control. Many machines with existing fixed-motion automation—such as mechanical timing devices (cams, jackshafts, gears, etc.)—are now being redesigned with servo mechanisms and intelligent electronic controls. Also, many small machines that were never previously automated are being fitted with computerized control systems.

Today, powerful technology is available for sophisticated and precise motion control which allows innovative solutions for "flexible manufacturing" requirements. The cost of these controls has dropped significantly in recent years permitting broad use of motion control products for both OEMs (Original Equipment Manufacturers) and end users.

0-7803-0453-5/91$1.00© 1991IEEE

The electronic and performance capabilities of these controls are reaching a high level of maturity and standardization. As this has happened, the job of factory automation has become more focused on software and application development tools, rather than on hardware features and performance. Certainly, one of the largest costs involved with implementation of electronic motion control, is the cost of developing the application solution rather than the actual cost of the control hardware itself. Maintenance of the electronics and software has a significant impact as well. It involves employing trained personnel with the right expertise, available in-house.

Tools Currently Available

Most of the functionality required for solving motion control applications now resides within the resident software of the of motion controller. This has led process and design engineers to focus on the software to get the efficiencies they need for plant floor automation. With the primary thrust of developing motion control solutions now focused on software, more efficient software development tools are becoming increasingly important. The software tools now have the most direct impact on both the time it takes to implement an application solution as well as on the cost in associated engineering resources.

Presently, many motion controllers require the use of some form of proprietary programming language to access their functionality. These languages range from variants of traditional programming languages—such as BASIC, Pascal or Forth—to completely customized command sets. Even though they may have different syntax, the software available in all motion controllers share a common heritage in their use of traditional programming methodology (i.e. commands in text files).

Customized languages are diverse, highly specialized, and incompatible with each other as well as with traditional programming languages. There are six or eight standard programming languages in common use and an even a broader spectrum of *specialized* languages. As a result of this diversity and the fact that no control manufacturer dominates the field, no standard has arisen. Therefore, OEMs and end users today are faced with the fact that unless a particular facility has only one supplier of motion control equipment, they are likely to have a difficult support issue. Training people to understand and be efficient in developing application solutions and supporting them with multiple programming environments is very costly and time consuming.

Some motion control manufacturers have chosen the alternative route of providing their functionality as an extension to existing commercial software development packages rather than implementing a custom language themselves. Such motion controllers usually include libraries that are linked into third party products such as Microsoft's C or QuickBASIC; or Borland's popular TurboPascal. Some even supply assembly language drivers and/or interfaces. This still requires expertise in traditional programming techniques to implement motion automation solutions. The third party approach also requires the developer become familiar with two or more *different* products from different suppliers. This can lead to a problem of responsibility and support since the commercial software vendor usually has no knowledge of motion control and the specific extensions provided by the motion control manufacturer. The motion control manufacturer, on the other hand, supplies only a small part of the software and is hard-pressed to support the entire language and maintain compatibility with the software vendor's enhancements and upgrades.

The Problem with Traditional Programming Languages

Unfortunately, there are several major drawbacks to using any kind of traditional programming language or a customized version of a traditional language. First and foremost, skilled software specialists are needed to implement motion control solutions. Second, there is a great inefficiency in having the process engineer communicate his desires to the "programmer". Thirdly, the resulting code is often very difficult to support by others after it is completed.

The way most process engineers think about automation is not naturally compatible with the way conventional software tools force them to think. For example, the syntax of conventional programming languages requires complete entry of all associated details at the time it is written. In other words, you must know everything all at once! They also force design of the solution in a linear way, since conventional programs are inherently linear by nature. Automation system designers need an equivalent to a sketchpad, it's much easier and more natural to conceptualize major process functions and their associated timing sequences first and then go back later to fill in the details. Process engineers—because of the way they think—are more comfortable with a hierarchical flow chart approach rather than linear program code. Conventional programming languages force too much attention to be spent on the use of the tools and not enough on actually designing!

Often machine designers are forced to turn to software experts because programming languages require specialized and non-intuitive terminology, structure, and syntax. Unfortunately, developing software requires learning and understanding a whole set of jargon that is unrelated to the actual automation problem. To write programs in conventional programming languages, the developer must understand many software-specific concepts. These include things like variable "types", subroutines, labels, loop and control structures such as IF...THEN...ELSE, DO...LOOP, CASE structures, etc. No process engineer really wants to know how to properly terminate an IF statement, or whether to use a FOR...NEXT structure rather than a WHILE...LOOP. Nor should one be expected to remember the specific name of the command (is it GOTO or GO TO). How many non-programmers (or even sometimes software experts) have been frustrated by not remembering the rules for statement terminator characters or where commas are required? Automation engineers don't spend enough time writing software to become proficient programmers and they shouldn't need to!

The traditional software development environment contributes to the problem by adding new terminology to worry about. One must know what the difference between a compiler and an interpreter is, what a linker does, etc.. The end result is that instead of concentrating on the automation *solution*, the engineer is forced to spend his time concentrating on the details of the "language".

Some motion control products have attempted to alleviate the problem by providing a "syntax-directed editor" with the development package. This is a tool which puts an English language interface between the programmer and the resulting code. Prompting for individual functions in English only provides a limited degree of assistance. In reality, a syntax-directed editor only helps you create a mess more easily, it doesn't address the main issue which is overall readability.

The multi-leveled structure required by conventional programming does not facilitate readability. Hierarchical structures such as nested loops or subroutine calls are not easily indicated in a text oriented program listing. Anyone familiar with programming knows that artificial aids such as indentation must be used to make the program readable. Such formatting aids are still insufficient to clarify the overall structure of a conventional program. It is often difficult to locate the source code for a subroutine that you are calling (in order to remember what parameters need to be specified), since the routine in question might be located in a completely different place in the program's listing. Programming languages were developed to focus on implementation, not on readability.

Using traditional programming languages for motion control applications incurs significant support costs after they are installed in the factory. Plant floor engineers, technicians and other support people are usually not trained software experts. This can make it a frustrating experience to attempt to make minor adjustments in the process or diagnose problems not anticipated by the original designer. The verbose nature and specialized syntax of conventional application programming languages make it hard to gain a quick understanding of the underlying operating concept of the machine. Natural visual markers to aid in following the sequence of operations are lacking. Attempting to understand someone else's software, one quickly encounters "word overflow" staring at pages and pages of program code listings!

The end result is that the application or process control engineer is overwhelmed with a plethora of possible software environments, and the few software tools available are difficult to use. Further, conventional programming languages are not a natural or intuitive way to developing automation applications. The problem thus persists that solving motion control problems involves an investment in software experts both for developing the application solution and for supporting it afterward. Today, there just aren't enough competent software people available.

Using Graphics

The first part of the solution to making motion control applications development easier is to take advantage of more of our natural capabilities than just language. Psychologists have known for a long time that combining the visual capabilities of our brains with the verbal facilities can make learning much easier. The old saying that "a picture is worth a thousand words" really has validity for programming as well. Dr. Jesse Quatse stated it eloquently in an article in a recent issue [1] of Industrial Computing magazine, *"It is difficult to say a dozen things at the same time but it is very easy to see a picture of a dozen things... C is for saying not seeing!"*

A good way to take advantage of this combined visual/verbal method is to allow the engineer to conceptualize ideas in the form of simple pictures which represent basic process functions with lines connecting them to represent the timing sequence. This is really a graphical flow chart which is a very natural extension of the way the engineer thinks about machine operation. The function blocks can be represented by icons on the computer's screen which also have a text field for additional clarity. The eye is much better at quickly picking up a *symbol* on a page than a word or phrase. By designing the icons so that they either look like what they do, or illicit a mental image of their function, it becomes easy to see what's going on.

Some scientific evidence now supports real benefits to users of a graphical environment. The consulting firm of Temple, Barker & Sloane conducted a study in 1989 in which they measured the benefits of a graphical interface versus a conventional text entry system [2]. When two sets of subjects where given a series of tasks to accomplish, their test results showed that 35% more tasks were completed in the same amount of time by those using the graphical system. The study also indicated that 58% more correct work was done in the same time. Novice users appeared to have attempted 23% more different tasks than their counterparts using a text entry system. Rated fatigue and frustration levels were documented as 50% less when using a graphical environment.

Object Oriented Design

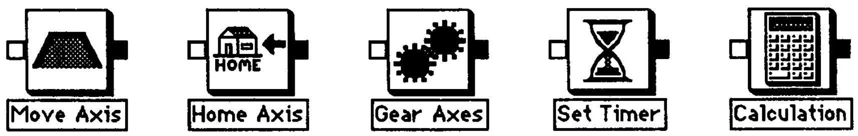
The second component of the solution to making motion controllers easier to apply is to combine the graphical user interface with an object-oriented, modular structure. Object-oriented programming is a new concept just coming into wide use, particularly for graphically oriented tools.

A major attribute of an object-oriented programming system is that it is designed to allow individual modules within the system to be developed independently. It provides the capability to design using sophisticated functional modules that don't necessarily require detailed knowledge of their inner workings. Another key attribute of object-oriented environments is the facility for allowing modules to be easily re-used with little or no modification. This allows creation of application-specific "black boxes", greatly reducing the time to develop additional related applications or modify the original one.

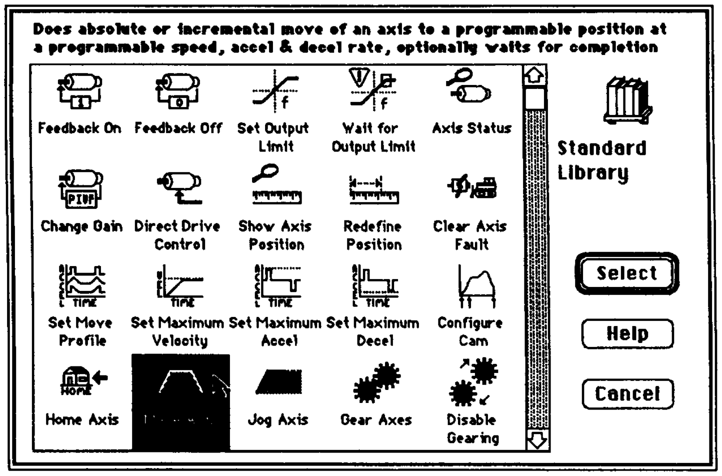
Brad Cox in his recent article on Object-Oriented Programming in Byte Magazine [3] describes the concept by stating that custom control systems can be built from off-the-shelf cards without having to understand soldering irons and silicon chips, just as vendors can engineer cards from off-the-shelf chips without detailed knowledge of transistors. This process has been applied for some time to electrical and mechanical systems design, it now needs to be applied to the software used for motion control applications. The potential productivity benefits are very substantial.

Putting It All Together

The Creonics Operation of Allen-Bradley has applied some of these principles in a unique application development environment called Graphical Motion Control Language (GML) which can be used to implement motion control solutions using Allen-Bradley Stand-Alone and PLC plug-in motion controllers. Following are 5 examples of the more than 100 functions blocks and their associated icons used to implement motion control solutions with GML:



Providing a graphical function block library—where you can scroll through all the available function blocks and just point and click to select a particular one to include in the flow chart,—makes it very easy to put together complex operational sequences. Color coding related function types, using green for all blocks that cause motion or red for all blocks that stop motion for example, can greatly aid in quickly selecting the needed function. An example of such a library is shown below:


Does absolute or incremental move of an axis to a programmable position at a programmable speed, accel & decel rate, optionally waits for completion

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.