# Use of Open Source Distribution for a Machine Tool Controller

William P. Shackleford and Frederick M. Proctor

National Institute of Standards and Technology
100 Bureau Drive, Stop 8230
Gaithersburg, MD 20899 USA

## ABSTRACT

In recent years a growing number of government and university labs, non-profit organizations and even a few for-profit corporations have found that making their source code public is good for both developers and users. In machine tool control, a growing number of users are demanding that the controllers they buy be "open architecture," which would allow third parties and end-users at least limited ability to modify, extend or replace the components of that controller. This paper examines the advantages and dangers of going one step further, and providing "open source" controllers by relating the experiences of users and developers of the Enhanced Machine Controller (EMC). We also examine some implications for the development of standards for open-architecture but closed-source controllers. Some of the questions we hope to answer include: How can the quality be maintained after the source code has been modified? Can the code be trusted to run on expensive machines and parts, or when the safety of the operator is an issue? Can "open-architecture" but closed-source controllers ever achieve the level of flexibility or extensibility that open-source controllers can?

**Keywords:** open architecture, open source, open architecture control, computer-numerical control, real-time control, Linux

## 1. GENESIS OF THE OPEN SOURCE MACHINE TOOL CONTROLLER

In the late 1980s, buyers of industrial automation equipment such as machine tools and robots looked at the business revolution fueled by desktop computing and began calling for a revolution in their industry. Announcing an assault on life cycle costs, they targeted the proprietary computer-numerical controller (CNC) and began pressuring CNC vendors to provide open architecture solutions that could seamlessly integrate with other vendor's equipment and could be easily customized.

The focus on open architecture CNCs resulted from U.S. President Ronald Reagan's 1987 announcement of a seven-point initiative to revitalize the flagging U.S. machine tool industry, which was losing market share to foreign competitors, mainly from Germany and Japan. The initial action was the June 1987 Machine Tool/Manufacturing Technology Conference aiming to establish a national research and development agenda. Seven independent groups ranked next-generation controllers as one of the top two technologies that should be pursued, along with sensors.

The U.S. Air Force Manufacturing Technologies office responded with the Next Generation Controller (NGC) program, a four-year contract to Martin Marietta Information and Communication Systems to develop, prototype, and test an open architecture controller, and publish specifications for an open system architecture standard (SOSAS). The National Institute of Standards and Technology (NIST) participated in NGC through a contract from the Air Force to publish a white paper on the Neutral Manufacturing Language (NML),[1] NGC's proposed method for component communication.

In 1992, following the NML work, NIST began the Enhanced Machine Controller (EMC) program[2] under funding from the U.S. Navy to investigate standards issues in open architecture controller for machine tools. Also in 1992, the Department of Energy (DOE) initiated the Technologies Enabling Agile Manufacturing (TEAM) program[3] to promote synergy between private industry and DOE facilities in the area of agile manufacturing. In a June 1994 TEAM technical program kickoff meeting, NIST and TEAM began collaborating on open architecture control standards, and the General Motors Powertrain (GMPT) Division proposed a series of production tests on their

This publication was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright. Email correspondence to WPS: will.shackleford@nist.gov, FMP: frederick.proctor@nist.gov

equipment to validate open architecture concepts. This proposal led to NIST's installation of a PC-based EMC on a Kearney & Trecker 800 four-axis horizontal machining center at the GMPT facility in Pontiac, Michigan*. Completed in February 1996, the EMC was part of a TEAM demonstration of remote access via the Internet, during which engineers at the Lawrence Livermore National Laboratory in Livermore, California operated the machine tool in Pontiac, Michigan via a Windows PC. The controller consisted of two PCs linked by Ethernet, one running a Windows-based graphical user interface (GUI) and the second running a real-time operating system and hosting a hardware motion control board. It was programmed in C and C++.

Shortly after the completion of the EMC at GM, NIST received a request for the EMC software from Open Engineering, who came across the project writeup during a web search. Open Engineering had hoped to use the PC-Based EMC replacement for a broken Bridgeport Boss-5 CNC on a Series 2 three-axis knee mill. NIST understood that Open Engineering is entitled to the software, which as a publicly-funded work of the U.S. Government is not copyrighted and in the public domain. However, NIST explained that the software would likely be of no use, since it was specifically written for the Kearney & Trecker machine tool at GM and required two PCs. Open Engineering noted that most of the machine-specific software could simply be eliminated for their Bridgeport three-axis knee mill, and the rest ported to a single computer running a multitasking operating system. Since this fit in with the EMC program's goals of investigating portability issues, NIST agreed. What followed was a series of ports that culminated in a full-software version running on Linux, using real-time Linux for motion control and eliminating the need for peripheral motion control computer boards. Language support was extended to include Java and Tcl/Tk for the GUI. Ensuing discussions about the EMC on Internet mailing lists attracted new users, and the EMC software was moved to SourceForge, a free hosting site for open-source projects.[4]

It is important to note that although the EMC has been successfully downloaded, installed, and used by dozens of people, building a free CNC was not the original motivation for the project, nor has it become one. NIST's purposes were and remain to be to have a full-software reference implementation of a controller for machine tools, robots, and coordinate measuring machines for the purpose of open architecture control standards validation. The open source aspect benefits the EMC in many ways, however, and is quite important to its progress.

## 2. OPEN ARCHITECTURE

An architecture is a set of components and their relationships. Components may be hardware, such as stereo receivers, speakers, compact disc players, or videocassette recorders, whose relationships are the electrical specifications and data formats that enable them to plug and play together. Components may be software, such as web browser plug-ins, whose relationships include what mechanism is used for communication and what data is transferred. Architectures may be "closed" or proprietary, in which only the owners are privy to the component specifications, or "open," in which the components' behavior and relationships are publicly known. If the behavior of components is poorly specified, components may be able to be integrated and without obvious errors, but may exhibit unexpected performance due to vendors' different interpretations of their intended behavior. If the relationships are poorly specified, components may not be able to be integrated at all.

Open architecture software does not necessarily mean that the source code is freely available. For example, the Portable Operating System Interface (POSIX) specification for operating systems defines components (e.g., operating system services, interactive command shells, and command-line utilities) and their interfaces (e.g., programming language bindings and program syntax), without requiring that vendors of POSIX-compliant operating system divulge any source code at all.

In the domain of industrial automation, open architecture controllers are those for which control functions have been partitioned into components, such as motion control, tooling control, and operator interface, and documented to the point where vendors and third parties can integrate components from various sources to build working machines. Safety and reliability become an issue with open architecture control. Desktop computing "accidents" cause loss of data; industrial control accidents can cause loss of life. Production line downtime can cost tens of thousands of dollars per minute, costs which are increasingly the contractual responsibility of the equipment vendors. Industry user groups, such as The Open Modular Architecture Controller (OMAC) Users Group, the Robotic Industries

*No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, instruments, or materials are identified in this report in order to facilitate understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

**Figure 1.** Open Engineering's Bridgeport three-axis knee mill retrofitted with the EMC.[5]

Association (RIA), and the Metrology Automation Association (MAA) are balancing the need for openness with the requirements for safety and reliability.

## 3. OPEN SOURCE

There are many variations on software distribution, ranging from typical proprietary software through full public-domain source code availability.

**Proprietary** software is the traditional commercial model. Software executables are sold or leased with no user right to distribute the executable to others. Commercial licensing terms vary, but limited copying rights may apply for backups or for other fair uses that do not deprive the software vendor from rightful income under its license. Source code is not provided under this model.

**Licensed source** is a modification to the proprietary software model, in which a licensee pays the software developer for access to the source code. Typically this is done to effect a port to a new platform, or to make changes required by the licensee for their products. These licenses may require the payment of royalties to the source code vendor. The licensee has no rights to further distribute the source code.

**Shareware** executables have no restrictions on distribution, but users are required to pay for the software under the honor system if they decide to use it. Unrestricted distribution is a marketing technique to spread the software to potential users and attract sales. Source code is not provided.

**Freeware** executables are like shareware in that there are no restrictions on distribution. Further, the author expects no payment, but may retain copyrights or patents, if any. Source code is usually not provided or available.

**Free software** gives users access to source code. Users have the rights to use the software, redistribute it, modify it, and distribute their modifications. Users can receive payment for these distributions. However, since any user can redistribute as many copies of the software as they would like, the price is normally very close to the

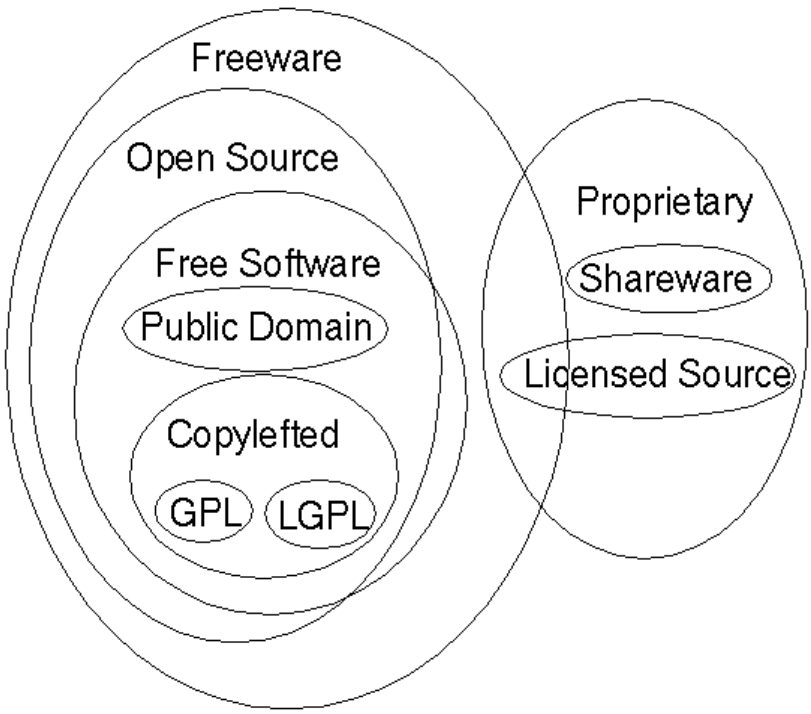


**Figure 2.** Set diagram of the various types of software.[8]

cost of media. Other than the copylefted conditions below there are virtually no restrictions on the use of the software or how it can be modified. This includes primarily public domain software and copylefted software.

**Copylefted Software** is free software that has been copyrighted so that the users rights to modify and redistribute the source code are always transferred to subsequent users. This is termed *copyleft*, in contrast with copyright, meaning that anyone who redistributes copylefted software, with or without changes, must pass along the freedom to further copy and change it. The Free Software Foundation (FSF), which was responsible for much of the development of the GNU suite of compilers and development tools, describes the free software philosophy[6] and has a specific copyleft license, the General Public License[7] (GPL) that details your rights with their software. The Lesser General Public License (LGPL) is primarily used for libraries. It allows the library to be used within a larger program without requiring the larger program to modify its possibly proprietary license.

**Public domain software** is source code that has no restrictions on use or distribution. It can be used in any way, copied, modified, distributed, and sold with any added licensing terms desired. The EMC is public domain software.

**Open source** as defined by the Open Source Initiative(OSI) is very similar to free software. Just as with free software the user is given access to the source code and guaranteed the right to redistribute it, modify it, and distribute their modifications. Some open source projects have strings attached to give the original developers more control that would not allow them to qualify as free software. Since the term "open source" is no longer trademarked by OSI, some proprietary vendors have tried to use the open source label, but this is generally not accepted by the open source community. One of the requirements of the SourceForge service is that projects meet OSI's definition of open source.

The Open Source Initiative (OSI) has a list of nine criteria required for a software to be certified Open Source. Most of the criteria are usually met by licensing the software under one of OSI's approved licenses or by releasing

the source code as public domain.[9] Neither definition would necessarily include software just because the source code was available without charge. To meet either definition, everyone receiving the software must be free to modify and redistribute the source code.

Users not only get free software, but they get a much stronger feeling that they are able influence the direction of the project. Developers are eventually able to hand off tasks which do not match their particular talents and interests. Software is more likely to be ported to new platforms. The scrutiny of many more eyes on the source code eliminates bugs. Documentation is developed more quickly.[10]

Many well respected software programs have claimed to benefit from the open source development. These include the Apache HTTP server[11,12]; the Free Software Foundations set of GNU compilers, editors and debuggers[13]; Netscape's Mozilla web browser[14]; the Sendmail mail transport agent[15,16]; and the Linux Operating System.[17,18]

The open source philosophy adopted by Linux has greatly benefited the real-time control community. By itself, the Linux operating system is not suitable for hard real-time applications. However, several open-source patches/add-ons allow hard real-time processes to share the CPU with normal non-realtime linux. Many of these were only possible because the Linux kernel was open source and could be closely examined and modified. RT-Linux, developed by Michael Barabanov and Victor Yodaiken of New Mexico Tech, was one of the earliest extensions to become popular.[19] It works by modifying the Linux kernel source code, intercepting all calls to enable and disable interrupts and dispatching them to a real-time scheduler.[20] The Real-Time Linux Application Interface (RTAI), developed by Paolo Mantegazza of the Dipartimento di Ingegneria Aerospaziale Politecnico di Milano (DIAPM), provides similar capabilities. It is based on DIAPM's Real-Time Hardware Abstraction Layer (RTHAL).[21] Kansas University Real-Time Linux (KURT) developed at the University of Kansas provides normal Linux user process with better timing resolution and control over scheduling than the standard Linux kernel. Although KURT is not strictly deterministic, it may be a choice for "firm real-time" applications.[22]

In the area of factory automation, open source applications include QCad[23] and FREEdraft,[24] 2D computer-aided design (CAD) tools; Open Cascade,[25] a set of reusable C++ object libraries for graphic modeling applications; and PuffinPLC,[26] a programmable logic controller (PLC) that enables PCs to be used as embedded controllers; and the EMC.

OMAC, in addition, is sponsoring the development of an open architecture specification, the OMAC Application Programming Interface (OMAC API),[27] that it hopes will standardize software interfaces to open architecture controllers. OMAC neither requires nor prevents open-source implementations of the OMAC API, but expects that commercial systems will retain proprietary control over component implementation.

Unlike the preceding projects, the EMC did not originally adopt the open source philosophy to further its goals. Rather, it was a consequence of requests for government work in the public domain. However, our experience with EMC shows that more is required to capitalize on open source distribution than simply releasing the source code under an appropriate license. A community of user–developers needs to be fostered. Tools need to be built or found to improve communication between developers, the testing and diagnosis of problems. As always, comprehensive documentation is necessary.

## 4. THE USER'S EXPERIENCE

With typical open source applications, users do not compile, read, modify, or otherwise access source code. These activities are left for developers. In many cases, however, users become developers through curiosity and a desire to contribute. The promotion from user to developer also happens when a user discovers a bug, tracks down its source, and fixes the problem. This effort is aided immensely by comment-documented source code. For the most part, however, users limit their source code activities to just what is necessary to get the application running properly, and then use it as they would any application, proprietary or open source.

### 4.0.1. Classes of Users

The clear user/developer demarcation does not apply so cleanly to the EMC. There are two categories of users: those who buy a machine on which an EMC is already installed and merely want to cut metal, and those who install an EMC on their machine as a means to a metal-cutting end, but otherwise want no part in porting to new platforms, fixing bugs, or developing new features. The first class of user does not benefit directly from open source, but benefits indirectly through platform ports, bug fixes, and the accumulation of added features. The latter class of

user, or "configurer," in fact requires some degree of open source, since the EMC in general won't work with arbitrary hardware configurations, as discussed in Section 4.1. Although many EMC configuration parameters are localized in runtime initialization files, some information (such as the kinematic equations for nonstandard machines) requires expression in algorithm form and hence some source code modification.

#### 4.0.2. Symbiosis Through Documentation

The needs of configurers points out one symbiotic benefit of open source projects and their users: the writing of documentation. Configurers clearly need documentation, not only instructions for where to put wires, but instructions on how to change the software to reflect machine characteristics that can't be bent to fit the software. For NIST's developers, the small amount of source code documentation supplemented with email between the few project staff is adequate. This small amount of documentation is not adequate for outside users and configurers. Much more detailed documentation is required, including step-by-step instructions that aren't required for the NIST project team.

Email mailing lists and web sites have proven to be effective at generating and archiving documentation. Initially, an existing commercial email list dedicated to computer-aided design (CAD), computer-aided manufacturing (CAM), electric discharge machining (EDM), and digital readout equipment (DRO), `CAD_CAM_EDM_DRO@egroups.com` was used. After several months of traffic on EMC, a dedicated list was set up: `emc@nist.gov`. List traffic followed the usual question-and-answer format, and some subscribers began archiving answers to build a documentation set. The documentation was moved to a web site, `www.LinuxCNC.org`, which now hosts EMC documentation.

### 4.1. Hardware Issues

Unlike many open source projects, EMC depends heavily on hardware, comprised of the "silicon" computing hardware (e.g., input/output boards), "copper" wiring harnesses, and the machine tool "iron" itself. Much of this hardware-specific configuration has been localized in runtime initialization text files that can be easily edited, but some configuration details are either infeasible to express in text files or are cases that have not been anticipated. Hardware customization is thus one problem where open source is a good solution.

#### 4.1.1. "Silicon" Hardware

Silicon hardware refers to computer boards for which there is no standard operating system interface, as there is for typical computer components such as mass storage, serial ports, and network interface controllers. Atypical components include digital- or analog input/output boards and position counter boards using quadrature encoders. The EMC provides a standard calling interface, the External Interface, that consists of a C-language header file that prototypes the functions that will be called by the core EMC code. Implementation of these functions is the responsibility of the user, in general, although implementations for some supported boards are provided in the baseline EMC distribution. External interface functions include:

```
extern int extEncoderRead(int encoder, double * value);

extern int extDacWrite(int dac, double value);

extern int extMaxLimitSwitchRead(int switch, int * value);
```

These functions are usually written to call board-specific code, and can be customized to stub out functions that aren't provided by a particular configuration. Note that while the function prototypes are declared in C header files, the functions may be implemented in any language that can be compiled into C-linkable object code, or in assembly language.

In cases where hardware is supported by the baseline EMC distribution, the user can select between supported boards without resorting to editing external interface functions. This can be done either by running one of the precompiled executables matching the user's hardware exactly, or by editing the "make files" and combining object files à la carte to build a custom executable. Here, the source code modifications required are localized to the make files, and no C or C++ source code need be changed.

### 4.1.2. "Copper" Hardware

Copper hardware refers to the wiring between actual machine tool components like limit switches, amplifier reference voltage leads, and encoder feedback cables. Although there are intuitive schemes for wiring these components into the interface boards, it is not always possible. For example, the intuitive way to wire the first (or **X**) axis is to connect its motor and encoder wires to the first available connections on the input/output board. In practice, it may be necessary to shift this assignment around broken board inputs, in this case wiring the **X** axis to the second connection set, the **Y** axis to the third, etc. Here, source code access to the external interface functions can be quite handy.

Some wiring choices can be specified in the EMC initialization file. This is a text file conformant with the Microsoft Windows INI file format, in which sections contain keyword–value pairs, e.g.,

```
[EMCIO]
SPINDLE_ON_INDEX = 1
SPINDLE_ON_POLARITY = 0
```

which specifies that the digital output bit that turns the machine tool spindle on is at digital input index 1, and the polarity is such that when the EMC writes 0 to this bit, the spindle turns off, otherwise it turns on.

### 4.1.3. "Iron" Hardware

Lastly, the configuration of the machine's links and joints must be taken into account. This is handled by the kinematic functions, which relate the user's Cartesian coordinate frame to the motor positions. For typical three-axis machine tools, the default configuration maps motor positions 0, 1, ... to Cartesian positions **X**, **Y**, ... directly and is sufficient for most machines. For machines with non-trivial kinematics, such as robots or hexapod machine tools, the default kinematics need to be replaced with software specific to the machine.

Even with the default kinematics, the scale factors between axis position feedback from the encoders and user units, e.g., inches or millimeters, must be specified. These vary widely based on the resolution of the encoders, the number of teeth on pulleys, and the pitch of drive screws. This can be specified in the INI file, as:

```
[AXIS_0]
INPUT_SCALE = 4000 0
```

which specifies that 4000 units of feedback (typically encoder counts) equals one user unit (typically inches).

The EMC comes with trivial kinematics as the default, and includes example kinematics for the PUMA 560 robot and Stewart Platform hexapod machines.[28] This is an important responsibility of open source projects: well-commented sample source code for a variety of options or alternate configurations is a necessary component of documentation.

## 5. THE DEVELOPER'S EXPERIENCE

Developers are those who need to do more than get the application to run properly, and who need access to the source code. Examples of the need for source code include:

- fast bug fixes. With open source, either one can track down and fix bugs oneself, or post the problem to the development community. Bug fixes can be posted much more quickly than if source code access were limited to only a few.
- Ports to new platforms. This may be able to be done with no changes to the EMC source, but source access is required for compilation. In general, source code needs to be changed to reflect platform differences that cause errors at compile or run time.
- Addition of new features. In some cases, this may be able to be done with no changes to the EMC source, for example when adding standalone utility programs that interact with the EMC using the public communication buffers. This is no different than extending an open architecture controller without open source. Extending core EMC functionality, for example adding new motion planning or servo control algorithms, does require editing the source.

## 5.1. Building a Community

Developers must have some degree of software programming experience. In open source projects related to software development, for example the GNU compiler, users are by definition developers. In machine tool control, users are machinists, production engineers, or shops owners who may not have programming experience at all. For EMC, this has meant reaching out to audiences that are likely to include programmers who have machine tool experience, or vice-versa. The most fertile audiences have been university engineering departments, and machine shop hobbyists.

In order to encourage this community of user–developers to grow, it was necessary to create and administer a web page, an email list, and an FTP archive. Additional documentation needed to be created. Now, however, EMC is benefiting from web sites and documentation built by some of the early adopters, as well as free services from sites like the Open Source Development Network (OSDN)[29] and SourceForge.[4] This has reduced NIST's costs in these areas and is a clear benefit of adopting the open source model.

## 5.2. Open Source Development Mechanics

Users who wish to start modifying EMC have several options:

- Keep their modifications private. The advantage to this is they need do nothing else. The disadvantages are that no one else in the group benefits, and any new releases may overwrite the local modifications.
- Make their add-ons and modifications available on their own site, or on the `www.linuxcnc.org` drop box. This requires action by EMC developers to bring in and update the core EMC code.
- Email a patch file or a set of modified files back to the mailing list to be included in a later EMC distribution. Like the previous method, this requires action by EMC developers.
- Join the EMC community on SourceForge and check their changes in directly. This is the most effective way to drive EMC development, since no actions need be taken by other EMC developers. This method is initially the most cumbersome.

The `linuxcnc.org` web site and its drop box feature were created by an EMC user. The drop box is popular because it requires little setup or technical expertise to use. In conjunction with the `emc@nist.gov` mailing list, it makes the files available to other EMC users almost instantaneously.

The EMC repository on SourceForge can be accessed from any computer on the Internet. SourceForge uses the Concurrent Versioning System (CVS)[30,31] to track versions and merge changes. To prevent source code sabotage, CVS uses the Secure Shell (ssh)[32] protocol to encrypt transactions. Developers need to register with both SourceForge and the EMC administrators to get password authentication. Once set up, the development scenario is simply to do a `cvs update` to get the latest checked-in version of the code; edit the code locally using whatever editors and compilers are desired; and finish with a `cvs commit` to check it back in. In the occassional event that two developers are editing the same file, CVS will indicate conflicts during the merge. The developers then need to resolve conflicts manually.

## 5.3. Contributions to EMC

EMC contributions benefit everyone in the community. NIST enjoys continual software improvement and addition of new features without having to expend scarce project resources. EMC users benefit through rapid deployment of software improvements and new features, which can take place during off hours or when the NIST staff are occupied with standards validation work. Several contributions worth noting are described in the following sections.

### 5.3.1. Bug Fixes

The EMC is constantly being developed and installed. New code is being written, which introduces new bugs, and the wide variety of field experiences bring existing bugs to light. The motor control algorithm is one example. Motor control is accomplished through proportional-integral-derivative (PID) control, based on the error between the commanded position and the measured position. Angelo Brousalis, in Athens, Greece, noticed problems with steady-state performance and uncovered a numeric sign error in the application of the integral term. He also added compensation for motor backlash and responded via email with the fixes.

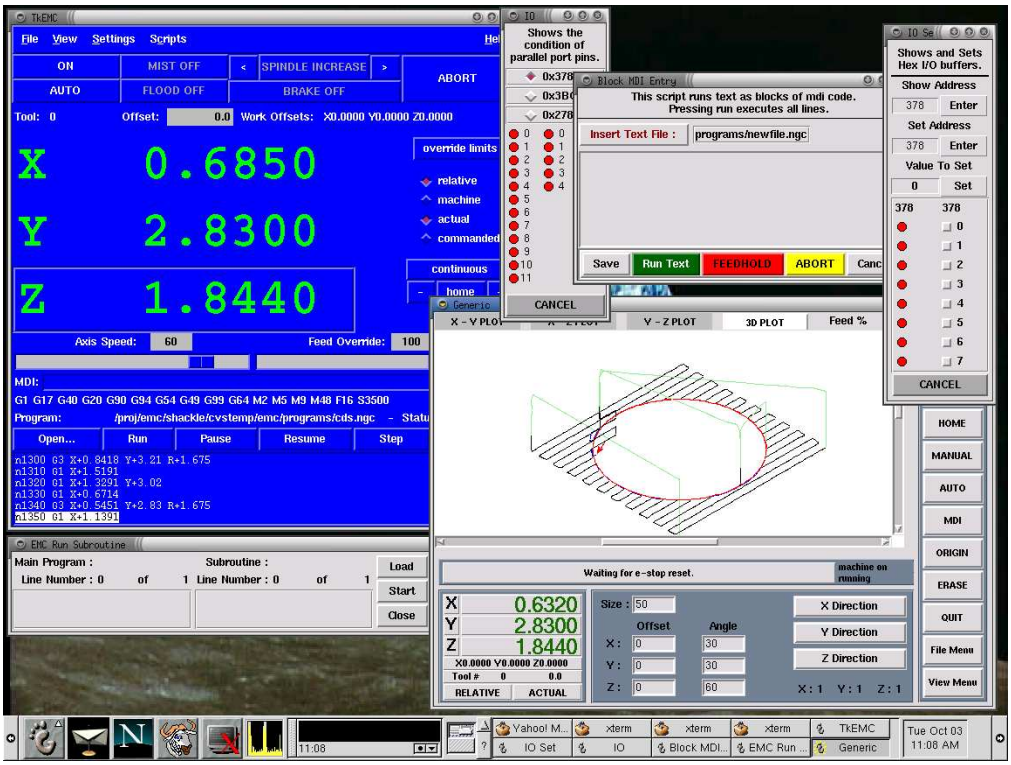


**Figure 3.** A screen shot showing only some of the Graphical User Interfaces (GUI), contributed by Ray Henry.

### 5.3.2. New Graphical User Interfaces

One of the most visible benefits of the open source model was the development of several new GUIs. The baseline GUI is written in the scripting language Tcl/Tk,[33–35] whose scripts run on many popular platforms including Microsoft Windows, Linux, and Sun Microsystems' Solaris. The GUI is easily extensible in its native source code form, and provides a Scripts menu that can run any Tcl/Tk script.

Ray Henry, in Crystal Falls, Michigan, wrote several Tcl/Tk scripts. A "backplotter" plots the 3-D position of EMC as well as offering several substantial improvements to the existing front end. A input/output point diagnostics tool shows the status of the digital I/O connected to relays that control spindle, lubricant, and coolant. A manual data input (MDI) window allows the convenient interactive entry of "G code" statements used to program tool path motion.

Current work includes the development of a "conversational programming" interface, in which tool paths for common features such as drill patterns, rectangular pockets, circular pockets, and bolt hole circles are generated from simple form fill-outs.

### 5.3.3. New Distributions

The EMC can be quite difficult to install from scratch, a process that includes a full Linux operating system install, real-time kernel configuration and compilation, the setting of boot parameters, network configuration, X Windows configuration, and EMC installation. While the EMC community has done an excellent job documenting the steps involved, it is not a "click the install button" procedure.

Dave Schecter (Arizona) and William Scalione (Texas) have each built a "Zip-EMC" CD-ROM installation of Linux and the EMC that runs from an MS-DOS partition. These distributions use the ubiquitous ZIP compression format and the LOADLIN utility to unpack the Linux file system, real-time kernel, and EMC into a DOS directory and jump to the Linux loader, which boots Linux and runs the EMC. Users are still required to configure the INI file for their machine, but not operating system expertise is required.

## 6. QUALITY CONTROL

One complaint often leveled at open architecture controllers, and particularly open source controllers, is that they will be less reliable than proprietary systems. This is somewhat inconsistent with open source advocates' assertions that the peer-reviewed nature of open source code improves its reliability. The unreliability complaint stems from scenarios in which a working system is modified in the field, after is has been tested and certified by the vendor, and fails. No amount of peer review on the baseline system will ensure that a modified system will have equal reliability.

There are many ways modifications can compromise system integrity. In controllers based on common off-the-shelf (COTS) components, users could simply substitute less reliable memory, disks, video- or network cards. This affects both open architecture- and open source systems. Architectural components, i.e., ones that conform to the architecture specification, are interchangeable by definition, and there is no guarantee that components from other vendors will work as advertised. With open source controllers, modifications can be made relatively easily, even inadvertently, through mistaken edits on system files.

Users can enforce their own policies concerning field changes, using mechanisms like passwords and permissions to prevent changes from happening. This points out a consequence of open architecture control: end users need to take ownership much more than with traditional turnkey proprietary systems.

## 7. FLEXIBILITY

For control applications that require a high flexibility, open source controllers may be the only solution. Consider the so-called *PC* Augmented architecture. Here, a PC-based open system serves as a front end to a proprietary controller, connected via a regulated communication link. This link serves as a bulkhead, enabling the proprietary system to validate traffic and limit commands to a small approved set. Architecturally, the components are the PC front end and the proprietary component, and the interface specification defines what information flows through the bulkhead. This coarse granularity makes it easier for the vendor to certify their product, but it limits flexibility by making changes possible only at the front end.

Fine-granularity architectures spread control functions across a larger number of components, each which can be replaced independently from the others and thereby increasing the flexibility users have to customize their systems. The price for this flexibility is the difficulty certifying the overall system. In fine-granularity architectures, end users take ownership to the degree that they become system integrators.

Open source systems maximize the granularity down to individual lines of code, thus maximizing flexibility. At this limit, end users take total ownership when they modify their systems, and become developers themselves.

## 8. SUMMARY AND CONCLUSIONS

An open architecture defines components, their behavior, and their relationships in a publicly-available specification. Components may be hardware, software, or a mix of the two. The implementation of components is left up to vendors, who are free to determine target platforms and price/performance to suit their market. The POSIX operating system specification is an example of an open architecture.

Open source software implementations go further than conforming to a specification. They open up the implementation of components through the public release of the source code itself. In most cases this is a result of the nature of the work, for example publicly-funded work or the work of students. In some cases, however, companies release their source code implementations to enlist programming support, achieve ports to new platforms, or grow their product through the addition of new features. The Linux operating system developed by the student Linus Torvalds is an example of an open source implementation, in this case of the POSIX specification.

Open source business models often include technical support, printed documentation, or ease of installation as ways to generate income from the distribution of free source code. These businesses may have originated the product themselves, or they may have been built as service providers to existing open source software to which they did not originally contribute. The many Linux operating system distribution vendors, such as Red Hat, Caldera, and SuSe, are examples of open source businesses.

Open architecture control is the application of open architecture concepts to the domain of industrial control. Users are encouraging open architecture control as a way to reduce the life cycle costs of machine tools, robots, and inspection equipment through the elimination of proprietary barriers to integration and feature addition. Industry

user groups like OMAC, RIA, and MAA have formed to ensure a balance between user need for openness and vendor needs for marketable, safe, and reliable products. Some are even pursuing open architecture standardization efforts, such as the OMAC API.

The Enhanced Machine Controller (EMC) is an open source implementation of a machine tool controller that was written by the National Institute of Standards and Technology (NIST) to support standards validation, including the OMAC API. The public-domain nature of the EMC, due to its development by the U.S. Government, has resulting in its use outside of NIST by individuals including students, hobbyists, machine shops, and machine tool retrofitters. Beginning with an initial contact through a web search, and continuing through test installations and reports on Internet mailing lists, the EMC is now a full-fledged open source project, hosted at the SourceForge web site and supported by a dedicated mailing list and web sites.

Users and developers need varying levels of source code access. At a minimum, machine configuration is limited to editing a runtime initialization file. This technique is often applied to proprietary source applications, for example, with Microsoft Windows INI files. In some cases full source code may be required, for example to port to new platforms or make substantial customizations.

EMC has benefited from the open source model in several ways. Users have found and fixed software bugs that would have otherwise slowed NIST's standardization work. New features have been added, particularly in the graphical user interface area, that have proved useful to NIST and others who do not have the time to develop them. Perhaps most importantly, the availability of source code has accelerated the writing of documentation. This has helped the NIST project, which like any organization experiences staff turnover and needs to train new personnel, and the EMC user community at large.

## ACKNOWLEDGMENTS

This work was aided by the many people on the `emc@nist.gov` and `CAD_CAM_EDM_DRO@egroups.com` mailing lists, and support from `LinuxCNC.org`.

## REFERENCES

1. W. P. Shackleford, F. M. Proctor, and J. L. Michaloski, "The neutral message language: A model and method for message passing in heterogeneous environments," in *Advances in Robotics, Manufacturing, Automation, Control, Soft Computing, Multimedia, Image Processing, Medical Imaging and Financial Engineering*, M. Jamshidi, P. Borne, A. Maciejewski, S. Nahavandi, R. Lumia, M. Fathi, and T. Furuhashi, eds., *Proc. World Automation Congress* **4**, 2000.
2. F. M. Proctor, "The Enhanced Machine Controller." World Wide Web, 2000. www.isd.mel.nist.gov/projects/emc/emc.html.
3. Department of Energy, "TEAM Final Report, UCRL-MI-133790." CD-ROM, May 1999.
4. VA Linux Systems, "SourceForge." World Wide Web, 2000. www.sourceforge.net.
5. M. Shaver, "Bridgeport Series 1 Mill with EMC Control." World Wide Web, 2000. www.erols.com/mshaver/bps1.html.
6. The Free Software Foundation, "What is Free Software?." World Wide Web, 2000. www.fsf.org/philosophy/free-sw.html.
7. The Free Software Foundation, "GNU General Public License." World Wide Web, 2000. www.gnu.org/copyleft/gpl.html.
8. The Free Software Foundation, "Categories of Free and Non-Free Software." World Wide Web, 2000. http://www.fsf.org/philosophy/categories.html.
9. B. Perens, "The Open Source Definition." World Wide Web, 1999. www.opensource.org/osd.html.
10. E. Raymond, "The Cathedral and the Bazaar." World Wide Web, 1997. www.tuxedo.org/ esr/writings/cathedral-bazaar/.
11. B. Laurie, P. Laurie, and R. Denn, *Apache : The Definitive Guide*, O'Reilly & Associates, Sebastopol, CA, 1999.
12. The Apache Software Foundation, "The Apache Software Foundation." World Wide Web, 2000. www.apache.org.
13. The Free Software Foundation, "GNU's Not Unix!." World Wide Web, 2000. www.gnu.org.

14. The Mozilla Organization, "mozilla.org." World Wide Web, 2000. www.mozilla.org.
15. B. Costales and E. Allman, *Sendmail*, O'Reilly & Associates, Sebastopol, CA, 1997.
16. The Sendmail Consortium, "Sendmail Home Page." World Wide Web, 2000. www.sendmail.org.
17. M. Kofler, *LINUX: Installation, Configuration, and Use*, Addison-Wesley, Menlo Park, CA, 1999.
18. Linux Online, "Linux." World Wide Web, 2000. www.linux.org.
19. realtimelinux.org, "The Real Time Linux Portal." World Wide Web, 2000. www.realtimelinux.org.
20. V. Yodaiken, "The RT-Linux approach to hard real-time." World Wide Web, 1997. www.rtlinux.org/rtlinux.new/documents/papers/whitepaper.html.
21. P. Mantegazza, E. Bianchi, and L. Dozio, "DIAPM RTAI." World Wide Web, 2000. www.aero.polimi.it/projects/rtai/.
22. R. Hill, B. Srinivasan, S. Pather, and D. Niehaus, "Temporal Resolution and Real-Time Extensions to Linux." World Wide Web, 1998. www.ittc.ukans.edu/kurt/.
23. A. Mustun, "About QCad." World Wide Web, 2000. www.qcad.org.
24. C. Johnson, "FREEdraft2D libre cad." World Wide Web, 2000. www.freeengineer.org.
25. opencascade.org, "What is Open Cascade?." World Wide Web, 2000. www.opencascade.org/about/whatis.html.
26. K. Crater, "Welcome to PuffinPLC.org." World Wide Web, 2000. www.puffinplc.org.
27. J. L. Michaloski, "OMAC API Working Group." World Wide Web, 2000. http://www.isd.mel.nist.gov/projects/teamapi.
28. R. Bostelman, A. Jacoff, F. Proctor, T. Kramer, and A. Wavering, "Cable-based Reconfigurable Machines for Large Scale Manufacturing." Proceedings of the 2000 Japan-USA Symposium on Flexible Automation, 2000.
29. VA Linux Systems, "Open Source Development Network." World Wide Web, 2000. www.osdn.com.
30. K. F. Fogel, *Open Source Development With CVS*, The Coriolis Group, Scottsdale, AZ, 2000.
31. Red Bean, "The CVS Book." World Wide Web, 2000. cvsbook.red-bean.com.
32. SSH Communications Security, "The Secure Shell Community Site." World Wide Web, 2000. www.ssh.org.
33. Ajuba Solutions, "Tcl/Tk Developer XChange." World Wide Web, 2000. dev.scriptics.com/software/tcltk/.
34. J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Menlo Park, CA, 1994.
35. B. B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall, Engelwood Cliffs, NJ, 2000.