

# OPEN CONTROL SYSTEMS

**Peter Wurmsdobler**

Control.com, Inc  
134 Flanders Road, Westborough MA 01581, USA  
peter@control.com

**Mario J. R. de Sousa**

Faculdade de Engenharia da Universidade do Porto  
Rua Dr. Roberto Frias, s/n 4200-465 Porto  
PORTUGAL  
msousa@fe.up.pt

## Abstract

This work in progress report is on an on-going effort toward open control systems, both in terms of hardware as in terms of software. The presented paper gives an overview of different types of control applications and common approaches to implement control systems. Beyond that, the presentation expands to an explanation of the open control paradigm as it differentiates from a proprietary one. An open control architecture for an open controller is proposed in an abstract way. Advantages and requirements of open control systems are highlighted.

## 1 Introduction

Control engineering and automation is very heterogeneous and even the word ‘control’ is perceived differently by different people working in different areas and at different abstraction levels. So from the view of control algorithms, control can refer to logic control, feed-back control or feed-forward control; some people may think of PID control, some other on  $H_{\text{inf}}$  control, Fuzzy control or Neural networks. From the application standpoint, the major fields of control are:

- Motion control
- Robot control
- Machine control
- Plant control
- Batch control
- Process control
- Distributed control
- Supervisory control

Various control algorithms can be employed in different control applications, e.g. PID loops for motion control. Several control applications are in general combined hierarchically to a control system, e.g. motion control is part of feed forward robot control which again is part of some supervisory control. The elements in this system are nowadays implemented in various technologies, with dedicated hardware and software:

- Micro-controller,  $\mu\text{C}$ , to control motion
- Programmable Logic Controller, PLC, to control a robot
- PC-based controller, to implement supervisory control

Even though all elements may have different design and performance criteria, they all have in common a minimum set of basic requirements, they should be

- deterministic,
- predictable,

- rugged,
- scalable,
- redundant,
- fail-safe,
- maintainable.

That said, companies in the control business build elements of control systems, but usually with proprietary technology inside and with proprietary interfaces between them. The result is a monolithic control system, because elements cannot be replaced by foreign pieces. This is the proprietary control systems paradigm. The next section is about an alternative.

## 2 Open control systems

Before the term "Open control systems" is explained in more detail, let's start with the meaning of the word 'open'. Of course, the meaning will depend on the context, but in engineering, different interpretations can be heard when applied to the word 'standard':

- open = widely used corporate technology,
- open = well documented corporate technologies,
- open = not patented technologies.

In the context of control systems, 'open' control systems are control systems composed of 'open' control components. What are 'open' control components? Pieces of control hard and software which interfere one to each other according to 'open' standards as does the entire system to the outside world. What are 'open' standards? Standards which are open for discussion, available and documented. And what are standards? Sets of rules how to implement technology or in general how to do something, which *per definitionem* is not under control of a single person or vendor, but rather by a group or consortium, so-called standardizations institutes. So the term "open standard" is a pleonasm, and the term "corporate standard" a *contradictio in adjectu*.

Certainly, standards may be technology initiated by a vendor which is often the case, like PCI or TCP/IP. Then sometimes it becomes widely used which is still not open nor a standard, it is just "widely used". Only if control is given to a multi-vendor platform or standardization entity, a higher degree of "openness" can be achieved and technology becomes a true standard. All these definitions describe the open control paradigm with its consequences:

**Components** A system consists of components which can be of the shelf and are invariant to a specific vendor as they comply to standards, like a PC104 CPU board or a cPCI analog i/o board for PC104 and cPCI systems, respectively.

**Modular** Any component can be replaced by another one offering the same functionality without influencing the entire control system. Simple to very complex control systems can be built like in the LEGO world. The prerequisite is again open interfaces between modules.

**Extenible** Due to open interfaces the global functionality can be increased by adding components. Not only open inside, also to the outside world.

**Portable** Concerning software parts of the control system can be moved on other platforms. POSIX for operating systems can be an example.

**Scalable** Open control systems are scalable. Not only performance can be improved by exchanging the critical component, CPU module, but also the system can be distributed onto several modules.

**Maintainable** A modular system is per se more maintainable. Not the entire system has to be shut down or replaced, only the module.

**Independence** Any component can even be replaced by a component from another vendor offering the same functionality.

What are open control systems good for and what is the advantage for the user: all assets boil down to **choice** at all levels, comprising hardware and software, and at the end **freedom**. The user does not depend on corporations any more. Like if you buy DIN screws, you can buy them from any supplier, and it will fit in any DIN thread.

How do such open control systems look like? All over the world there are many ongoing projects on open control for different types of applications, ranging from motion to process control, and at different levels of abstraction. Just to mention some of them:

**LinuxPLC** aka MAT (Machine Automation Tools), Programmable logic controller software with PLC tasks running as Linux processes, [1].

**PuffinSCADA** A UNIX based software collection to implement SCADA systems with an extremely modular design, with clients and servers, [2].

**MCA** The modular control architecture, mainly targeted to robot control and based upon reusable software modules. Any controller can be implemented from C++ objects providing four edges: sensor and controller input and output, [3].

**OMAC** The open modular architecture controller. An effort towards intelligent machine control on an abstract system wide level, [4].

**ORCOS** (Open source RObot COntrol Software) is a new open source project for robot control, the

design and implementation of a “framework” for real-time motion control[5].

Sourcing from all these efforts, an open control system could be composed of open controllers with an open architecture as smallest cells (Figure 1). In terms of hardware they are based on ISA, PCI, cPCI and PC104 compliant hardware as here a certain modularity is given per se. Building upon cPCI or PC104 hardware, a POSIX compliant operating system hosts all different control application. These could be implemented as real time processes or threads, in user or kernel space, share and exchange data over shared memory or other inter process communications methods, locally or over the network, by an open control protocol. One example for a generic control application is MatPLC, a PLC running on top of Linux as explained in the next section.

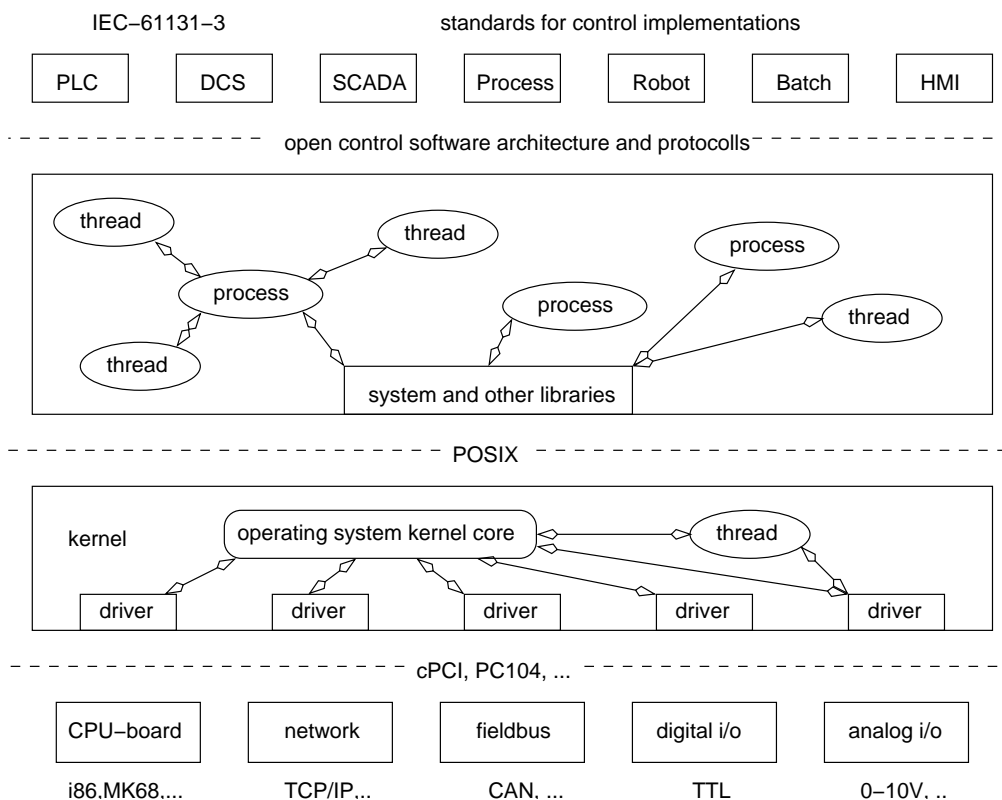


FIGURE 1: *The open control architecture*

### 3 MatPLC – running a PLC on Linux

Although the MatPLC is intended to emulate a standard PLC, it has been designed in such a way that it can scale from a small embedded microcontroller, up to a fully fledged supervisory and control application with graphical interfaces. It’s highly modular archi-

ture supports concurrent code development, and takes maximum advantage of the underlying operating systems’ services. Each module is dedicated to a specific function: some handle physical I/O, reading inputs from I/O boards and copying their states to the MatPLC coils (and/or vice-versa), other modules are dedicated to executing the program logic, and yet others may be dedicated to communicating

with remote PLCs.

Unlike standard PLCs, the MatPLC does not necessarily run in an infinite loop: read the inputs, execute the logic, and update the outputs. MatPLC modules are autonomous and, by default, will execute in separate processes. Each of these modules is free to decide whether or not to execute in a standard PLC loop. Communication and synchronisation among the modules is made through a common MatPLC library. This library accesses common memory areas that together harness the PLC's state, and synchronises using semaphores. All modules run asynchronously by default, but may be synchronized using the MatPLC library

Despite this, however, the standard PLC model permeates the structure: the MatPLC library offers PLC like semantics for the modules that wish to use them. For example, inputs that can only change at the beginning of the logic, and outputs that are only written at the end of the logic. This means that a user's PLC skills can easily be transferred to the MatPLC.

A specific module can also have multiple instances running in a working environment. For example, a user wanting to run two digital filters at different sampling frequencies can create two processes of the same digital signal processing (dsp) module with different configurations. Each module will run under a different name.

The architecture does not differentiate between modules executing logic and modules doing physical I/O. For this reason all the coils of the MatPLC are internal coils, called plc points, and no output and input coils exist. The mapping between plc points and physical I/O is left to the configuration of the I/O modules. It is perfectly possible to have the same plc point mapped to more than one output. The opposite, however is not supported as it does not make sense. For every plc point there is at most one module instance that can change that point (set/reset or set to a value).

The common MatPLC library is itself modular, and can be divided into several sections as depicted in figure 2:

PuffinPLC figure

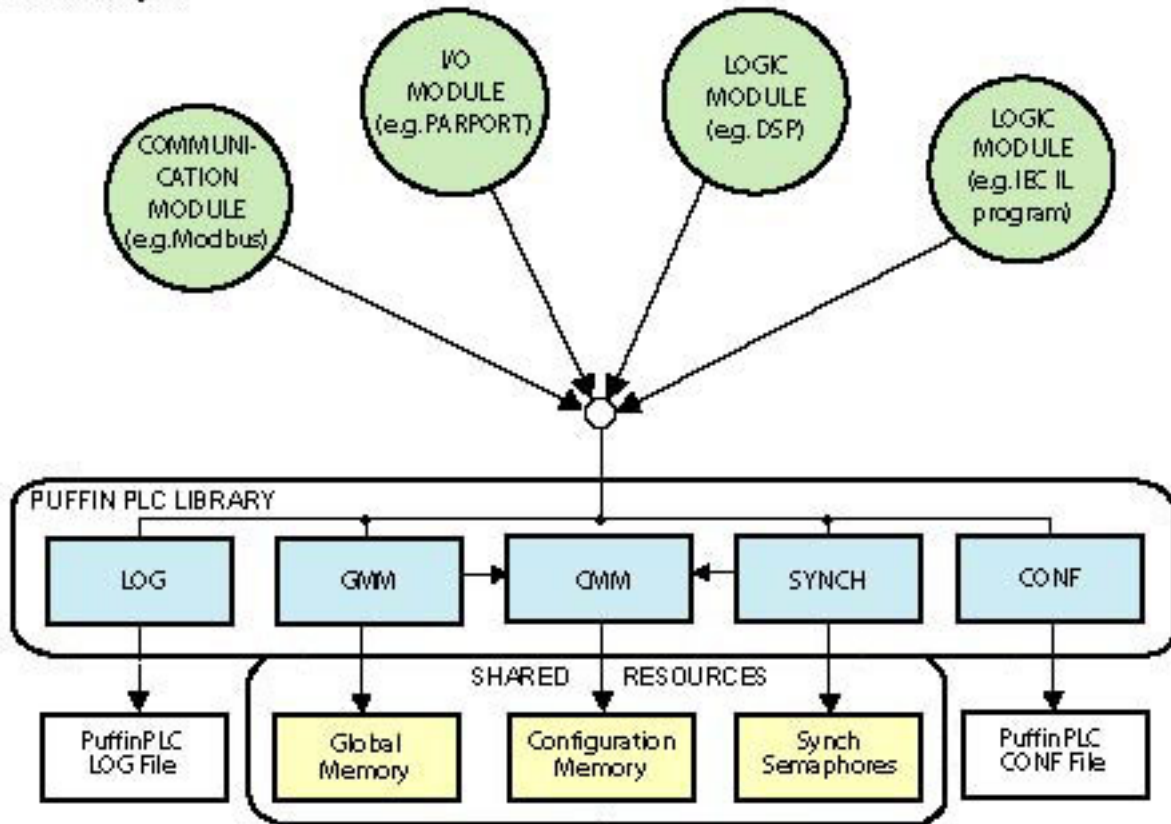


FIGURE 2: *The matPLC architecture*

- CMM** The Configuration Memory Manager manages the shared memory that stores configuration data;
- GMM** The Global Memory Manager manages the global memory used to store the state of the plc points;
- Synch** The Synchronisation library handles the synchronization between modules;
- Period** The Period library handles module periodic execution, i.e. scan timings;
- State** The State library handles module running state;
- Config** The Configuration library is used for parsing of the configuration files;
- Log** The Log library is a group of functions that lets every module produce logs in a consistent manner.

Storing the MatPLC configuration in a common memory area guarantees that every module uses the same configuration data. Access to the configuration memory area is made through the CMM. At present, the CMM does not have a public API, so the modules themselves never access the CMM directly, but always through other sections of the MatPLC library. Each module has its own private memory map and a private map mask, both created upon module initialization, and both independent from the global memory map. When a module accesses a plc point, it is actually accessing its private memory map. Private and global memory maps are synchronized by calling the GMM's `plc_update()` function. Synchronization is controlled by a semaphore, which provides atomic updates. During synchronization, only the plc points to which the module has write access are copied into the global memory map. All other plc points are copied from the global memory map into the private memory map. The map mask is used to determine whether a module has write access to a plc point. Because of the use of a common shared resource controlled by a semaphore, using the MatPLC in a hard real-time environment will require the use of a scheduling mechanism that bounds priority inversion.

The use of private memory maps allows modules to run asynchronously without interfering with each other. Although it is not mandatory, it is expected that modules will run in an infinite loop, synchronizing their private memory maps at the beginning

and end of each loop iteration. Nevertheless, the MatPLC architecture is sufficiently flexible to support modules that do not execute in an infinite loop; these may synchronize their private maps (in whole or in part) whenever they see fit.

Modules are not aware of any synchronisation among themselves. It is the end user that configures module synchronisation through a Petri Net model. The Synchronisation library maps this model into semaphore accesses, without any help from the module programmer.

Scan rates for each module are handled by the period library. Currently this is implemented using POSIX timers, but in the future we hope to use scheduling functions of a Real-Time Operating System.

The state library handles the state of a module. This library has not yet been completed, and at the moment merely keeps a tab on the pid of running modules. In the future it will control the state of the module (running, stopped, switching to a new instance, ...). It will also be used to support online configuration changes by synchronising scan execution of the new and old instance of the same module.

## 4 Conclusion

In any case, an entire control system could be built using such open controllers as they can be scaled and programmed to be anything, a micro-controller, a PLC or an HMI. However, some design work has to be done to define a framework where different types of control and control related applications fit in. Here the common effort has to start.

## References

- [1] The LinuxPLC homepage,  
<http://www.linuxplc.org/>
- [2] PuffinSCADA homepage, soon on  
<http://www.control.com/>
- [3] Forschungszentrum Informatik, Interactive Diagnose und Servicesysteme,  
<http://www.fzi.de/ids/>
- [4] The OMAC Users Group,  
<http://www.arcweb.com/omac/>
- [5] Katholieke Universitaet Leuven, Department of mechanical Engineering,  
<http://www.mech.kuleuven.ac.be/>