Reprinted from *Industrial Ethernet Book* 52

Open Source industrial software: more hype or a new, better way?

The term 'Free and Open Source Software' (FOSS) denotes a type of software license that allows the licensee, among others, to continuously use, adapt and distribute a certain piece of software without fearing its discontinuation. The fact that Open Source software cannot be discontinued is probably the most important reason why FOSS is quickly gaining uptake in the machine and automation industry. When you ask the manufacturers of industrial products why they prefer FOSS, you frequently hear responses such as the *machine builder needs complete control of his machine... independence from discontinuations...*, or, *avoiding vendor lock*. The Open Source Automation Development Lab's manager, Carsten Emde, explains

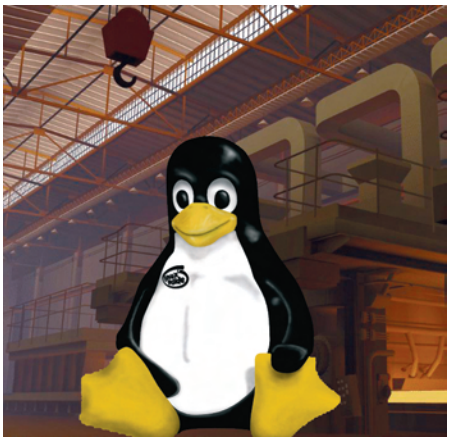
CONTRARY TO COMMON belief, the underlying economics of FOSS are not totally new. The 'Open Innovation' concept has long been used to let different companies – even competitors – jointly develop industrial components. Cost reduction is achieved by avoiding unnecessary parallel development. As such, FOSS can be seen as a variant of 'Open Innovation'. The most important prerequisite for a company to enter such a project, however, is to ensure that the components for joint development are not aimed at providing uniqueness to one of the participating companies.

For machine software, this is illustrated in the 'uniqueness pyramid' (Fig. 1) where the bottom level of the pyramid includes basic technologies without an exclusive benefit, and the top area includes unique knowledge that is differentiating the company from its competitors. Somewhere between the base and the tip of the pyramid is the 'uniqueness limit' to be individually determined for a specific company. If this limit is directly at the base of the pyramid, there is little or no expertise that can be exchanged with competitors, and FOSS cannot be used (at least not with a copyleft license – see the FOSS text panel for an explanation). If the uniqueness limit is higher (yellow and green lines), there is no particular reason why FOSS with a copyleft license cannot be used to develop components in the area below the uniqueness limit.

Linux: The new RTOS?

In the good old days when most of the computer systems on a machine were built specifically for the purpose, the traditional RTOS was also a custom job. In contrast to large and demanding operating systems that were already available at this time, the machine operating systems (e.g. OS-9, VxWorks, etc.) were small and fast and could be loaded without any problem in 512-KB eproms and run at 8MHz in a single MByte of ram. In addition, they provided the required real-time capabilities other operating systems could not.

But about ten years ago, with the general availability of the Internet and the ubiquitous



presence of computers, people were no longer prepared to consider a machine as something special. It was only natural that a CD/DVD file system, USB-based interfaces and unrestricted network capabilities were requested in machines in the same way as they were available, for example, in laptops. In addition, the advanced features of state-of-the-art processors – 64-bit, multi-core, virtualisation etc. – had to be supported in machine computers as well.

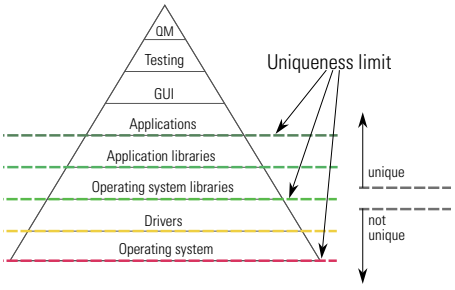


Fig. 1. Example of a 'uniqueness limit' that shows the different levels of knowledge of a company. Above the uniqueness limit lies the knowledge that differentiates a company from its competitors; below this limit are the basic technologies that do not make any difference in the market. If this threshold is at the bottom of the pyramid for a company (red line), that is, practically all the knowledge is market-relevant and confidential, FOSS should not be used. The higher the limit (yellow and green lines), the more (economically) attractive it is to use free and open software (FOSS).

With this growing number of requirements, it became virtually impossible to implement all of them in the various traditional RTOSes. The Linux operating system, on the other hand, provided such technologies, but possessed no real-time capabilities. Many people thought it simply impossible to turn Linux into a real-

What is Open Source software?

The terms 'Open Source software' and 'Free Software' represent different ideological perspectives, but legally and practically, both terms are equivalent descriptions of a specific type of software licensing. The term 'Free and Open Source Software' (FOSS) is, therefore, best used for this type of license.

The GNU General Public License (GNU GPL)

Most active FOSS projects use the GNU General Public License. This license grants the rights:

1. To run the program, for any purpose;
2. To study how the program works, and adapt it to a user's need;
3. To redistribute copies so one can help his or her neighbour;
4. To improve the program, and release the improvements to the public, so that the whole community benefits.

The second and fourth points require free access to the source code. This access is, thus, a necessary but not exclusive condition for qualifying a software as Open Source. When the software is passed on to others, the recipient must also be able to acquire these rights depending on the type of FOSS license. If the user only uses the software himself and does not pass it on, all the requirements from the Open Source license no longer have meaning. Contrary to a frequently-stated belief, money may be charged for FOSS, but only for associated services such as the effort required to pass on the FOSS, and not for the license itself. Apart from this consideration, a service provider may, of course, also charge customers for the activities required to develop and maintain FOSS.

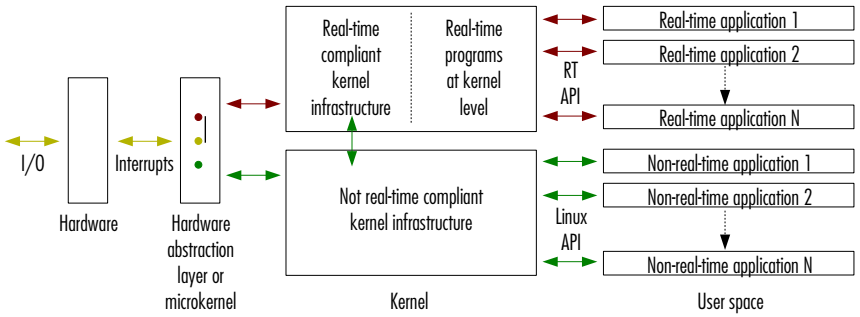


Fig. 2. The 'Dual-kernel approach' to achieve real-time capabilities with Linux

time operating system, yet equipping traditional RTOSes with all the new technologies was equally impossible...

Anyway, making Linux an RTOS was seen as having more possibilities. In a first step, a separate microkernel was used to provide the real-time capabilities, and a modified Linux kernel was running under the control of the microkernel to provide the required interfaces and protocols. This principle is known as 'Linux Real-Time Extension' or 'Dual-kernel approach' (Fig. 2). RTAI, RT-Linux and Xenomai are examples of such implementations. The disadvantage with this approach however is that the Linux kernel must be patched in order to be used in this environment, and it was clear from the very beginning that these patches would never be accepted in the mainline Linux kernel.

In consequence, other developers, namely Ingo Molnár and Thomas Gleixner, tried to patch the native Linux kernel in such a way that it becomes an RTOS; this project took the name PREEMPT_RT, as 'Linux mainline real-time' or as 'Single-kernel approach' (Fig. 3). At the occasion of the Ottawa Linux kernel summit in summer 2006, it was agreed that this is the way to go, and the PREEMPT_RT patches were gradually merged into the Linux kernel since then. Today, about 80% of the patches are available in mainline, and it is expected that the rest of it will follow by the end of the year.

A number of commercially available Linux distributions (e.g. Red Hat's MRG) already contain the PREEMPT_RT patches in production quality. The worst-case rule-of-thumb latency is in the range of 10^5 multiplied by the CPU cycle duration with these systems without special optimisation. For example, a 1GHz processor has a cycle duration of one nanosecond which results in a worst-case latency of about 100µs. This is considered sufficient for the vast majority of real-time projects. Using latency optimisation, however, the worst-case latency can be reduced by up to three times.

From theory to practice

In principle, there are three different ways to install Linux on an embedded system:

- Use a standard distribution (e.g. Fedora, Novell, Red Hat, Debian etc.) and adapt it to the requirements of an embedded system. This is, obviously, restricted to processor and chip sets for server and desktop computers supported by a particular standard distribution. While there is normally no problem with x86-based architectures, others such as ARM and MIPS are rarely supported.
- Use a dedicated distribution for embedded systems (Montavista, Timesys, Windriver or ELDK/Denx, Elinos/Sysgo, PTXdist/Pengutronix). These distributions support a wide variety of architectures, chip sets and controllers.
- Use the standard (vanilla) Linux kernel and assemble the required tools and libraries directly from the related repositories. This method allows the developer to make maximum use of the Linux kernel but requires considerable knowledge. ▶

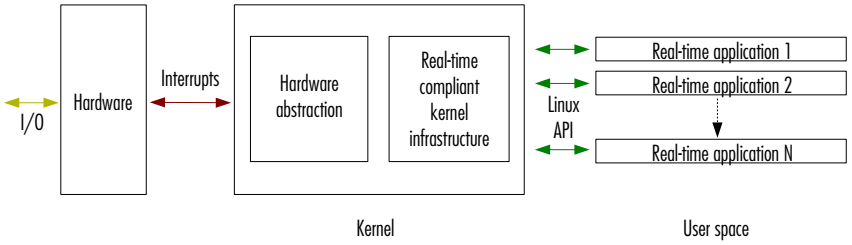


Fig. 3. The 'Single-kernel approach' to turn Linux into a real-time operating system

When a standard distribution is used, it may be difficult to obtain support and training from the software distributor, since these packages are not intended to be used in embedded systems and the distributor may not have the required expertise. Manufacturers of dedicated distributions, however, provide support and training which may be purchased on a per-time basis. In addition, the development of special drivers and extensions may be commissioned.

The same holds for the third way when a standard vanilla kernel is used. A number of Linux software service providers such as Linutronix, Germany, offer support to manage the native Linux kernel and to build tool chains and other software packages individually. Overall, these ways to install the operating system on an embedded system are not very different from the ways of traditional RTOSes, except that Linux makes it possible to change either the distribution or the software service provider more easily, since the Open Source nature of the license forbids the addition of proprietary components to the kernel.

The development process

How to develop system software and user applications? Developing for an embedded system in general is done either on a cross-development system or in a self-hosted environment.

Traditional RTOSes are either self-hosted, e.g. OS-9/68k, or cross-development systems, e.g. VxWorks, and cannot be changed easily. As an advantage of cross-development, the fastest

What is a copyleft?

The word 'copyleft' is an artificial word to denote the 'opposite of copyright'. This wordplay employs the term 'left' as the opposite of 'right' and also in the sense of the past tense of 'to leave' (somebody, something). Whereas the normal copyright forbids the passing on of a copyright work, the copyleft allows it and forces the identical license to be used as for the original work [*in the manner of a legal covenant – Ed*]. This requirement applies not only to the original version of the work, but also to all changes and additions. This constitutes a unique regulation since normal copyright regulations leave it up to the license holder to choose the type of license that will govern the changes and additions. However, when a license holder acquires a license containing a copyleft, he assumes this particular requirement which takes effect when the work is passed on.

The presently-known different Open Source licenses differ according to the strength of this copyleft effect. The BSD (Berkeley Software Distribution) license does not dictate to the license holder which license will govern the subject software along with the changes and additions; it is therefore an Open Source license without copyleft. An example of an Open Source license with full copyleft is the above-cited GNU GPL; it forces the license holder to retain the original license in its identical form for the subject software and any changes and additions.

For examples of the requirements of the GPL when passing on the Linux kernel, see also the FAQ box.

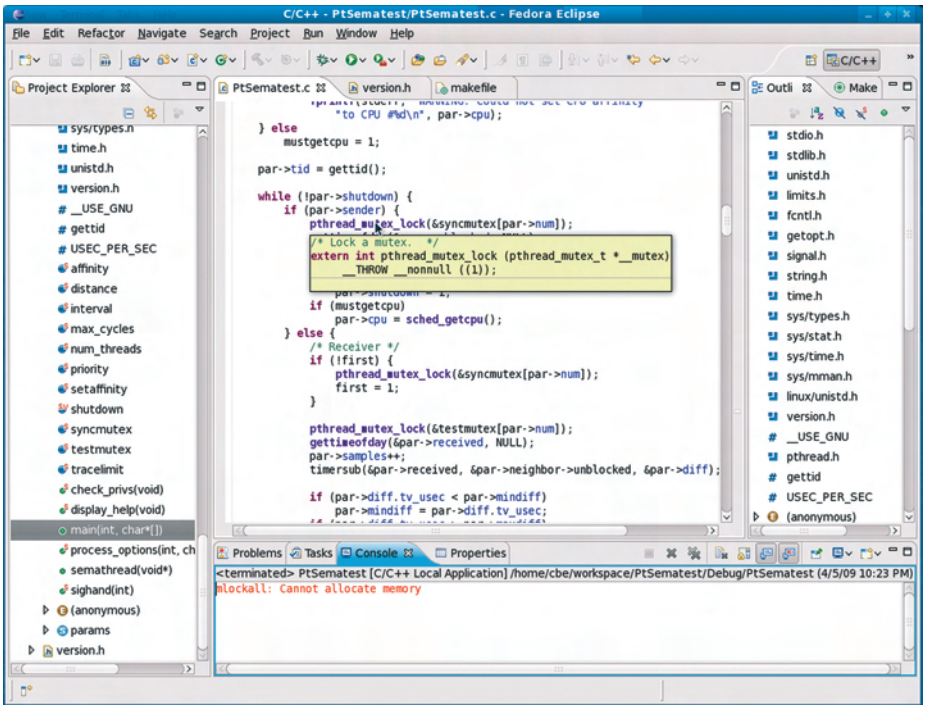


Fig. 4. Example of a real-time project in the Eclipse Open Source integrated development environment

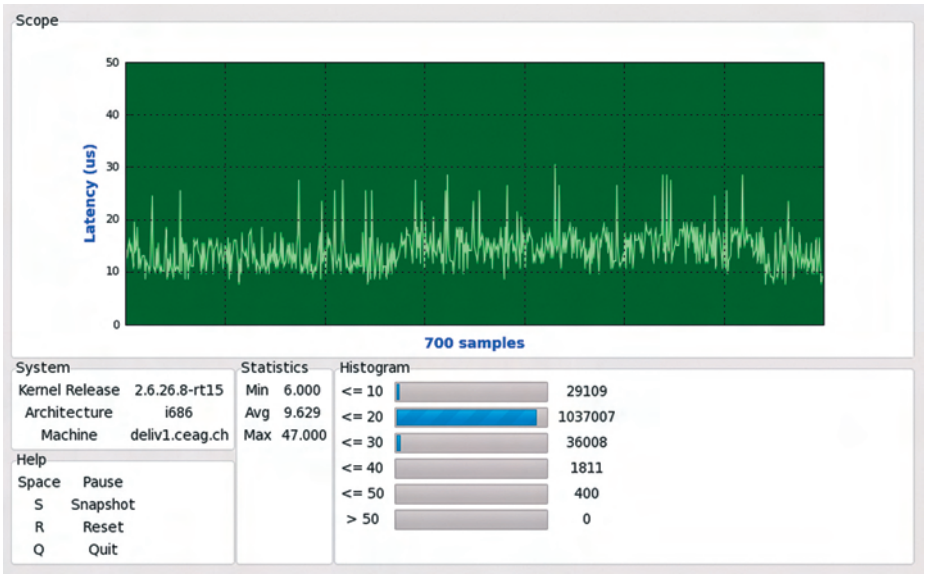


Fig. 5. The Open Source CyclictestoSCOPE developed by Arnaldo Carvalho de Melo is used to register and detect latencies on a real-time system

available system can be used for the development in order to reduce the compile time; the advantage of a self-hosted system is that debugging and tracing is more efficient and straight-forward. With Linux it is often possible to combine cross and self-hosted development which makes it possible to use the best of both worlds.

Many traditional RTOSes come with a proprietary work bench that integrates a number of tools under a common GUI to develop, debug and optimise a given system. Under Linux, there is a wide variety of such tools, but not all of them are integrated into a common user interface. Neither are they always easy to find and to use. Overall, however, similarly powerful tools are available. The most frequently used Open Source integrated development environment, for example, is Eclipse (**Fig. 4**). It provides everything needed during code development.

Another example that may be very helpful during optimisation of a real-time system is the CyclictestoSCOPE developed by Arnaldo Carvalho de Melo (**Fig. 5**). Again, Open Source software service providers may help to find tools which are best suitable for a given situation and provide training to use them.

Linux is now an RTOS and can be used in similar ways to traditional RTOSes. There are various options to use either standard distributions, dedicated distributions or to build local expertise in a particular company. The advantage of Open Source software and, in particular, Linux relates to the flexibility and unrestricted availability which it affords – no other operating system supports as many architectures and controllers as Linux does.

The disadvantage is that a particular tool may be difficult to find and less elegant in use compared to its commercial counterpart. On the other hand, Open Source software service providers may be able to help in cases where something does not work. Normally, the savings in the license costs far exceed those of hiring a Linux service provider for a limited time.

OSADL

The Open Source Automation Development Lab (OSADL) was founded in December 2005 and registered as a cooperative in summer 2006 with eleven founding members. Today, OSADL comprises some 30 members including semiconductor companies, software manufacturers and distributors, manufacturers of industrial computer boards and other hardware, Linux and Open Source software service providers and, most importantly, machine builders and industrial automation companies.

The goal of the Open Source Automation Development Lab (OSADL) is to promote and support the usage of Open Source software in the industry. The membership fees are used to delegate the development of Open Source software requested by the majority of members, or at least agreeing to do so. Other benefits of the OSADL membership include participation at exhibitions where OSADL is represented with a booth, free access to OSADL-organised conferences and seminars, free legal advice on questions of general interest and much more.

Details about OSADL's various projects and activities are available at www.osadl.org

Open Source software Frequently Asked Questions

Q: *I have used the Open Source GNU C compiler (gcc) to generate a machine control application. Do I have to make the gcc source code available to a customer when I am selling the machine?*

A: No.

Q: *I have used the Open Source GNU C compiler (gcc) to generate a machine control application. Do I have to make the source code of the application available to a customer when I am selling the machine?*

A: No.

Q: *The control computer of a machine that I have developed runs under the Linux Open Source operating system. Do I have to make the schematics of my machine available to a customer when I am selling the machine?*

A: No.

Q: *I have developed a machine control application for the Linux Open Source operating system. Do I have to make the source code of the application available to a customer when I am selling the machine along with the operating system and the application to a customer?*

A: No.

Q: *I have discovered a bug on the gcc and fixed it. I am using the gcc solely within my company. Do I have to publish the bug fix anywhere?*

A: No (but you will make a point of doing so if you wish the bug to be fixed in the next version).

Q: *I am using the Linux Open Source operating system solely within my own company. Are there important obligations of the Open Source license that I have to follow.*

A: No.

Q: *I am using the Linux Open Source operating system solely within my own company and do not distribute it to anybody. I have discovered and fixed a bug in the network driver. Do I have to publish the bug fix anywhere?*

A: No (but you will make a point of doing so if you wish the bug to be fixed in the next version).

Q: *The control computer of a machine that I have developed is running under the Linux Open Source operating system. Do I have to follow any obligations of the Open Source licence when I am selling the machine along with the operating system to a customer?*

A: Yes. A number of information and disclosure obligations must be followed: Information that the product uses Open Source software and the license text of the Open Source license, i.e. the GNU General Public License version 2, must accompany the product. This is best done in a dedicated section of the User's Manual. The source code of the Linux operating system must be made available in one of the various ways described in the GNU General Public License.

Q: *The control computer of a machine that I have developed is running under the Open Source operating system Linux. I have discovered and fixed a bug in the network driver. Do I have to publish the bug fix anywhere, when I am selling the machine along with the operating system to a customer?*

A: Yes. The bug fix must be licensed under the same license as the entire operating system and, thus, made available to the customer. This mechanism is called copyleft and defined in the GNU General Public License.

Q: *I am very enthusiastic about using Open Source software in the machine and automation industry and in embedded systems. I have a lot more questions with respect to legal and other issues. Who has the answers?*

A: To use Open Source software effectively in a commercial environment, a number of common interest groups have been founded. One of them is the Linux Foundation where Linux creator Linus Torvalds and other kernel developers are employed. A similar organisation, the Open Source Automation Development Lab (OSADL), is taking care of the special requirements of the machine and automation industry and of using Open Source software in embedded systems (see the section headed 'OSADL' to the left of this panel).

Carsten Emde *has spent more than 20 years as a software developer, system integrator and software consultant for industrial computer systems. Among others, he is specialised on real time, video and image processing. Since founding the Open Source Automation Development Lab (OSADL) in 2005, he is serving the organisation as general manager.*