

# Embedded GNU/Linux - drawing the big-picture

Nicholas Mc Guire

*OpenTech EDV-Research GmbH*

*Mistelbach, AUSTRIA, A-2130*

der.herr@hofr.at, <http://www.opentech.at>

## Abstract

Embedded GNU/Linux [1] [2] has become a serious option for industrial applications. It's virtues are performance and stability. But, are stability and performance alone sufficient reasons for management to consider it a serious alternative to proprietary solutions?

This paper gives an outline of what makes GNU/Linux valuable for industrial applications. Needless to say the outline is not complete. It would be an easy task to fill the requested six pages with references to successfully completed industrial projects alone!

This article will try to create the big-picture of embedded GNU/Linux and embedded real-time Linux. The intent is to show that GNU/Linux is more than just a stable operating system. Rather, it is a rich and complete system of development tools, architecture specific components, debugging and testing environments, and, last but not least, source and project management software aswell as services. The aggregate sum of these is the strength of GNU/Linux, a fast evolving resource pool that is based on the solid foundation of the free software idea.

## 1 Introduction

There are many introductions to GNU/Linux and embedded GNU/Linux. Here we wish to emphasize the integrated picture, the framework of the GNU effort, and how embedded Linux builds on open-source facilities.

## 2 Building the Big-Picture

The first part of this big-picture is the available software pool. In this section we will take a look at the facilities available and show how this mesh of software, seemingly so chaotic in its developmental approach, integrates into a complete and coherent whole.

### 2.1 Linux Kernel

The Linux Kernel itself is a good, maybe the best, example of "chaotic" development providing a highly competitive environment, while, at the same time, protecting the rights of the individual developer with a clean license model, the GPL [4]. The GPL ensures both the rights of the community and the individual.

The GPL license model and the willingness of leading figures within the Linux community to share responsibility have been, and continues to be, key issues for the ongoing Linux kernel project's success. There have been other open-source projects, BSD for example, that have failed. These failures have been primarily due to dependence on a license that provided the individual with little or no incentive to do very much, as there was no "return on investment"

connected with the BSD style license.

The Linux kernel project, was, from the very beginning, forced to build on open standards that would find wide and enthusiastic acceptance. At the same time Linux has to inter-operate with almost everything to find its way into the existing computer and network infrastructure.

Quite often Linux got its chance when a commercial component needed to be replaced, and the flexibility of the Linux kernel allowed developers to do the job quickly and efficiently based on the open interfaces the Linux kernel project provided. With time this lead to support for twenty different processor families, and innumerable peripheral devices.

#### Linux drivers and integration capabilities

- documented open interfaces
- good support for many bus systems
- publications for writing drivers
- many examples

#### Linux kernel core

- Kernel resources
- debugging
- error reporting
- testsuits
- community support
- hardware abstraction

## 2.2 GNU/Linux

The Free Software Foundation [3] GNU project is probably the fastest growing software facility ever launched. At the core of the GNU project is the willingness to provide communications infrastructure, source management utilities, and a platform independent set of development tools. It is no surprise that the GCC project, a free compiler to generate free software, lies at the very beginning of the development of GNU. This is not an illusionary freedom, it is a competitive freedom. It is the idea of giving equal opportunities to a community of individuals and then seeing who produces the best code.

The GNU concept is an important expansion of the traditional approach to competitive science,

an approach that relied on openly available information - on software technology. I personally believe that software is a science, not a set of products.

Selling software concepts is the same as selling mathematical principles and ideas, the same as selling number theory or set theory, or the calculus. Mathematics could never have developed in a society in which mathematical theory and know-how were not in the public domain.

The Free Software Foundation's projects are good examples of software development that manage to accommodate, and respect, both commercial and individual interests. Free software makes it possible for every individual to have the opportunity to build software that exactly fits their needs. Free software accommodates not only 'mainsstream' developers, but also minorities, handicapped users, such as myself, artists and scientists, and allows them to use their full creative potential, to exercise their creativity unencumbered and unlimited by proprietary, restrictive, or nonexistent software.

Critics have claimed that freedom of software, and more generally the freedom of intellectual property, would impede development as there would no longer be any incentive to create new ideas and solutions. Aside from the simple fact that the very existence of GNU/Linux disproves this idea, the concept of the freedom of technology is at the root of the success of the industrialized nations. And those nations with the greatest intellectual freedom have been the most successful. Proprietary know-how has not lead, and will not lead, to technological evolution. Nor has the public domain technology eliminated the demand for specialization, for individual solutions for specific problems.

Viewing software as a service, and not a product, eliminates the 'threat' of know-how being in the public domain. Recent discussions on logic patents within the EU have shown that there is no consensus in our sciety on this essential issue. Here is a choice yet to be made.

### GNU/Linux Open-source community

- Conferences that aim at delivering technology not selling brands
- Response time on mailing list is often lower than on commercial "hot-lines"
- Community interaction is a many-to-many communication process which results in many people correcting erroneous information "peer-review-nature" of open-source communication.
- Commercial hot-lines are many-to-one communication easily resulting in errors staying undetected
- Open communication channels allow correlation detection
- Open source communication allows early detection of future development directions, you hardly can be surprised by the content of a new release in an open-source project

### GNU/Linux services for internet presentation and project management

- mailing list management and source browsing tools
- Webservers and web editing tools
- open-source platforms, sourceforge, osdl, gnu.org
- database tools
- active community – talking to real humans

### GNU/Linux source management tools

- Platform independency of source management systems (you can even run them under Windows!)
- data-base interfaces for logging
- cvs, bk, etc. for safe source management
- interfaces to mail and web subsystems
- source browsing facilities

### GNU/Linux documentation and presentation tools

- Strong lean towards open standards simplifies doc. and allows development of core competence that will last
- Abundance of documentations and literature
- Core concepts are documented

### GNU/Linux platform abstraction tools

- Architecture independency of utilities and interfaces
- Architecture support in development tools
- Architecture abstraction – its a ./configure parameter only!
- Cross debug tools

### GNU/Linux development tools

- Complete documentation, tutorials and examples
- Follows open interface standards

### GNU/Linux GUIs

- graphics desk-top for high-end and low-resource systems
- on the jump to completeness
- office suites

### GNU/Linux shell/file-utils/text-utilities

- language support especially for minorities ignored by the big-players (Sweden)
- support for open formats allowing almost unlimited data exchange
- many different healthy competing flavors
- support for special groups (blind, death)
- completeness
- well documented

## 2.3 Sub-kernel facilities

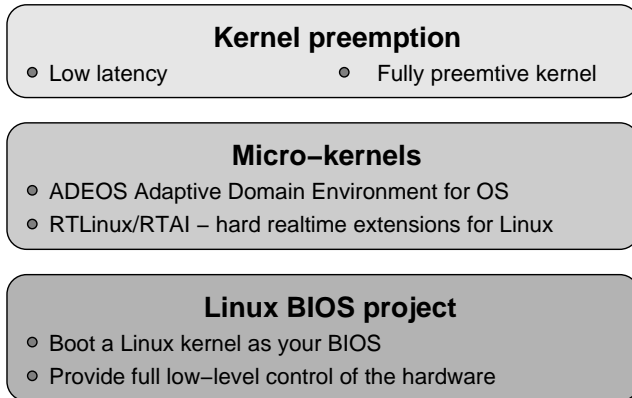
The itemization above covers a general purpose operating system suited for server and desk-top systems as well as some embedded applications. But what specific demands are there in embedded industrial applications ?

- Customizable boot-sequence/BIOS [6]
- Hard real-time demands [7] [8] [9]
- Soft real-time demands [2]

The open-source community has developed these 'sub-kernel features' over the past years to the

point where they are well suited for industrial applications.

These sub-kernel features are part of the kernel, but can be viewed conceptually as services below the kernel, as these services modify/control the kernel proper's behavior directly.



Especially the availability of soft and hard real-time extension to the Linux kernel allow embedded GNU/Linux to cover the entire range of industrial applications from cardiac arrhythmia controllers to 60MW puls generators from SoftPLC to Humenoid Robots [5].

## 2.4 GNU/Linux for Industry

The picture presented above is not simply a tool chain for industrial applications. It is a heterogeneous matrix of software that allows the construction of sub-sets that fit individual demands, demand spanning embedded low-end systems all the way to clusters with thousands of CPUs processing in parallel.

GNU/Linux is a complete software structure. This completeness is due to the ability of the individual to add to the matrix at points where she/he finds pieces missing, thereby expanding this open technology almost daily.

## 3 Communication structure of open source:

In this second section we cover one specific property of free software development, one where we

see industry failing to clearly understand the impact on industry's ability to utilize open-source. We introduce the specific communication models of open-source projects as we see them. Understanding these and adopting management concepts and software life cycle models to fit them are vital to opening this vast resource.

### 3.1 The service picture of software

Traditional development in industrial environments is based on a one-to-many (1xM) communication model. In recent developments of software life cycle concepts, such as the V-model, this has been reduced even further to a hierarchical one-to-one (1x1) communication system, bringing a specific developer to a specific customer, communicating on the same level in their respective hierarchical systems.

This concept assumes that the exchange of specifications, for the utilization of available technology, is the core issue in a development cycle, and that this information is interconnected between the individual levels only in a procedural manner. To overstate the case slightly, the V-model assumes that the layers are independent and that the procedural demands are communicated without the different layers needing to know what is going on at the technical level. The programmer knows nothing about management, and the manager does not know what POSIX is.

The open-source model is an all-to-all, or many-to-many (MxM) model, that targets know-how transference rather than the exchange of specifications. The underlying assumption is that through the transference of know-how, know-how grows on both sides, and multiple views of know-how are then applied to a specific problem.

### 3.2 The technology picture of software

Traditional software life cycle concepts assume, an assumption that I find to be absurd, that technology is simply available. Free software assumes that technology is generated during its application and thus the continual development

of technology can only take place if there is a know-how transfer in both directions. Unidirectional know-how transference leads to technological starvation.

Software projects are generally designed in the context of a company portfolio. This inherently limits the integration of such a product into a larger context. The integrative view, that is, not claiming that every editor and every utility-app is part of some master plan, is one of the great strengths of GNU/Linux. It is the product of an evolutionary model based on open standards for software technology.

Industrial software is built upon the analysis of demands. However, for obvious reasons, not all demands can be satisfied by an individual product. This leads to a definition of the maximum subset that can be provided with a given effort, thus excluding some demands, and some groups, by design. As an example I would like to note that not only are handicapped people often excluded from the 'mainstream' software, but also linguistic minorities. For example, that part of the population in Sweden that speaks old-Swedish simply can't get Microsoft products that support their language because there are "only" 800,000 people affected, and a special release for so 'small' a group would not be profitable. This is an inherent result of closed frameworks. Corporate frameworks are generally closed, whereas GNU/Linux has an open framework, and can thus allow specialized components to be integrated without problems into the mainstream.

### 3.3 Software feedback cycles

Traditional software follows the black box model. The customer buys a service in a box, uses the service, but is never allowed to either open, or look into, the box. The problem with this paradigm is that it only works for simple services. Software is not a simple service, and the black box model doesn't work. Imagine buying a car and not being able to open, or look under, the hood. Should something go wrong with your car then you send it back to the factory - or buy a new car.

- effective feedback requires informed users
- problem localization requires the ability to detect correlations
- the drive for improvements should come from the users, not only from the designers
- conceptual problems only are detectable if there is conceptual feedback

1x1 or Mx1 communication simply does not satisfy these demands for an effective feedback cycle. If it is supported by developers, MxM communication can, and does, fulfill these requirements. Open-source software, based on the GPL, is one framework that provides MxM communication, but it is not the only possible one.

### 3.4 Hiding errors

Looking at the development of scientific publications since the 1970's one can notice a strange phenomena - science itself seems to change, there are fewer and fewer errors. By the 1990s only success is deemed worthy of publication. Failures are hidden. And to what effect?

Industry has reassessed the triangular wheel over and over again in the past decades. If one research group had published their findings on triangular wheels, simply stating that they don't roll properly and are not usable, then industry would have been spared a great deal of developmental work. Reinventing does not push technology, reinventing simply wastes time, resources and the creative potential of engineers.

The free software concept is to publish errors, publish mistakes, publish disasters, to make failures as well known as successes. One very good reason for this is that a concept that might lead to disaster in one area might very well lead to success in a slightly different one. Putting out a fire with water is a disaster when trying to extinguish an oil fire, but it works quite well with burning wood. Publishing this failure can be the basis for a new technology. The success

of GNU/Linux is that failures are just as transparent as successes. Just look at kernel.org and think of what

<ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.11-dontuse.tar.bz>

might be. It was a 22MB (or 130 MB decompressed) disaster leading to a restructuring of a number of subsystems in Linux, resulting in a lasting improvement of Linux as a whole!

Industry's traditional approach is to hide failures. Industry has lost much due to this, but things still worked as long as the scientific community went on publishing errors. At present, with the scientific community only publishing successes - and on occasion 'producing' successes quite independent of the facts - the creativity of technological development is decreasing dramatically.

The power of the free software concept is the availability of the underlying technology, a technology that makes mistakes. A technology that is sometimes a chaotic branching set of half done ideas, put together in constellations that have the potential for creating the other halves needed to make the compound structure work. Development requires knowledge, the know-how of failures and successes.

## 4 Conclusion

Healthy open-source competition, in the free software community, is driving technology towards stability and improved performance, providing exactly what embedded application developers need. Open source CAN work, but it is not a systems that works by itself. It requires that the participants play by the rules of the free software community. These rules are sometimes hard to get across to management, as they strongly contradict management's traditional idea of what a 'product' is.

Embedded and desktop Linux systems, including the hard and soft real-time enhanced vari-

ants, are being successfully utilized for factory automation, medical, robotics and telecommunication applications. Embedded Linux is ready for industry - but large parts of industry are not yet ready for the open-source concepts needed to make embedded Linux successful for them. This will change.

Open-source projects often fail in industry because proprietary software life cycle models are applied. To be a success, open-source software needs to build on the integrative concepts of free software, and on open standards. The software life cycle must adopt the MxM communication model, the community concept, and the error/feedback models, outlined above for GNU/Linux, in order to provide the foundation for successful open-source projects.

## References

- [1] [GNU not UNIX] <http://www.gnu.org>
- [2] [Kernel] Linux Kernel developers site, <http://www.kernel.org>
- [3] [FSF] Free Software Foundation, <http://www.fsf.org>
- [4] [GPL] GNU General Public License, <http://www.gnu.org/copyleft/gpl.html>
- [5] [RTLF] Real Time Linux Foundation Inc. <http://www.realtimelinuxfoundation.org/event/event>
- [6] [LinuxBIOS] The Linux BIOS Project Home-Page, <http://www.linuxbios.org>
- [7] [RTLlinux/GPL] RTLlinux main developers sites, <http://www.rtlinux-gpl.org>  
<http://www.fsmlabs.com>
- [8] [RTAI] Real Time Application Interface, <http://www.rtai.org>
- [9] [ADEOS] Adaptive Domain Environment for Operating Systems <http://www.nongnu.org/adeos/>