Elektrobit

emlix
embedded linux systems

How to make Linux safe in a standards-compliant manner

Open-source software for safety-critical systems

elektrobit.com

The development and certification of safety-critical systems must follow clearly defined processes. Linux, which was developed by a community without a manufacturer that may be held liable, does not meet these requirements. However, the standards specify ways that allow Linux to be used in safety-critical environments.

The tasks of software-controlled systems have continuously increased in the last years.

This includes medical technology and automotive engineering. Innovation in functional systems development increasingly takes place in software. Today, a premium vehicle contains over 100 electronic control units that are programmed with more than 100 million lines of program code [1]. For a long time, functions were linked to the ECUs themselves so that more functions meant more ECUs, which was also due to the supply chains in the industry. The growing number of functional properties, for example, in infotainment, connectivity, driver assistance, or autonomous driving increases the number of ECUs. This results in an enormous rise in complexity as functions simultaneously interact with each other.

Decoupling hardware and software

One way to reduce this complexity is to decouple the function from the ECU, thus separating hardware and software. This centralizes computing power in the in-vehicle E/E architecture, i.e., throughout the entire vehicle electrical system so that functions are purely implemented as software and different functions from multiple suppliers must coexist on one ECU (Figure 1). Similar system architecture developments can, for example, be seen in the Integrated Modular Avionics (IMA) architecture used in aircraft construction [2].

The approach also shows parallels to data center operations where, first through virtualization and now through containerization [7], different functions are run on the same hardware. It can be assumed that the thus enabled servitization business models from IaaS to FaaS can also be applied to the automotive industry, resulting in new opportunities for collaboration. Technically, consolidation must ensure the freedom from interference between applications. In data security discussions, this is often referred to as multitenancy or tenant security.

With this in mind, both manufacturers and suppliers ask themselves at what levels differentiation is possible: only at the function level or also at the middleware, hypervisor, and operating system level. This question has not yet been answered. However, initial trends have emerged: By applying parts of the technology from data centers, they will also apply the business models and integrate themselves into their structures. Once the middleware has been standardized, the manufacturers' differentiation will take place essentially at the function level. In the automotive sector, for classic ECUs, this was done through AUTOSAR (AUTomotive Open System ARchitecture) and is currently extended to central processing units with Adaptive AUTOSAR [5].

Centralizing functions

In medical technology, consolidation of ECUs is not of the same paramount importance. However, this field, too, experiences an increase in the number of functions to be integrated. What can often be seen is modularization on a uniform software/hardware platform. At the same time, certain functions are centralized on one control computer in the device: This applies especially to data processing/visualization/logging, operation, and communication between components and with a local network or the internet. Particularly processing and visualization are usually safety-relevant functions. Their integrity must be ensured.

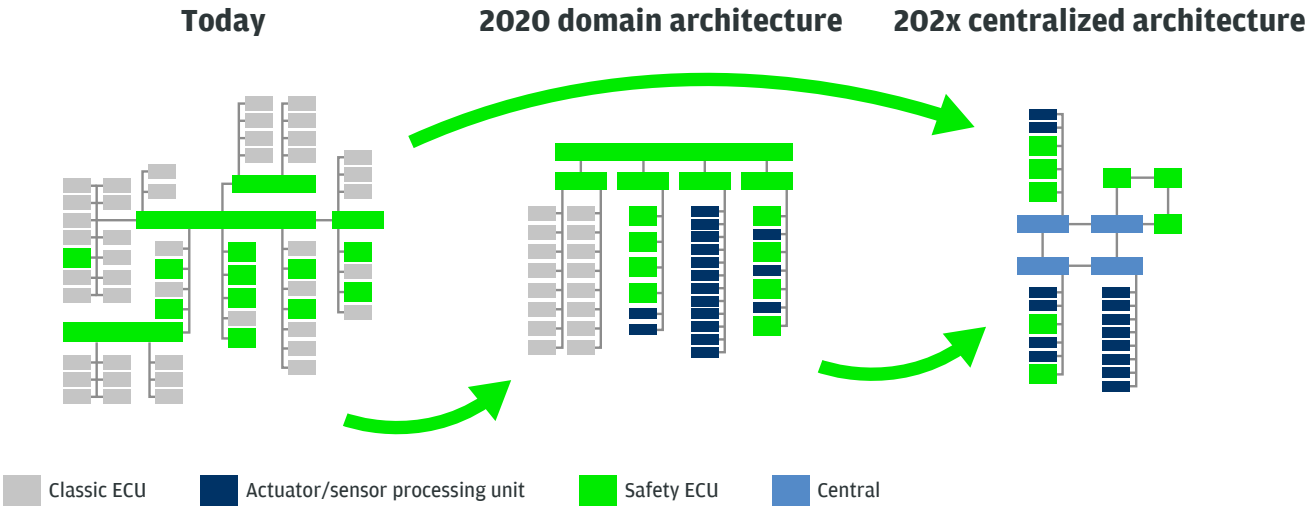


Figure 1: Evolution of vehicle architectures. (Source: Emlix)

If a medical system receives, for example, measurements of a probe, the overall system makes the path of this data safety-critical in the sense of patients' rights and integrity. Depending on the application, mechanisms are required that detect or prevent loss and corruption of measurements across communication paths and during save operations within the system. Aside from the standard numerical error analysis, a question that arises during data preparation is how reliable the mathematical libraries and algorithms used are. In the end, the representation and correct transfer of the results must be ensured. At the different levels – from architecture to verification –, this must be planned and documented in an understandable manner.

However, legal obstacles and compliance requirements of, for example, hospital operators often make it impossible to connect medical devices via wide area network connections. This results in paradoxical requirements for maintenance over the life cycle: up-to-date software vs. necessity for recertification.

Open-source software – mainline-compliant

Over the past years, open-source software has conquered even industrial applications that were initially largely excluded due to functional safety and certification considerations. For a good ten years, this has been the case for medical fields of application. In the automotive sector, Linux, outside of infotainment, currently debuts on central computers in the vehicle.

In this context, key questions arise that concern the development process and software maintenance over the entire life cycle. The established standards, certifications, and audits – e.g., IEC 62304 for medical technology or Automotive SPICE and ISO 26262 for the automotive sector – provide useful process instructions regarding planning, developing, testing, and documenting software. At least IEC 62304 includes life cycle management requirements.

However, both standards do not fully consider two essential aspects:

- The specific characteristics of open-source software, in particular embedded Linux, during the development process and later life cycle management.
- The fact that, for security reasons, devices that are connected to the internet and therefore accessible worldwide require active maintenance and regular updates.

Embedded Linux – although significantly reduced compared to desktop/server distributions – is not only considerably complex, but it is also a very comprehensive piece of software that initially is "simply just there"; it is not developed and maintained by a manufacturer that may be held liable but by an anonymous community that, at first glance, appears to be non-transparent. This software was created outside the control of the standardized development process. Now it must be integrated and maintained in the following life cycle stages in a standards-compliant manner.

This process no longer independently goes through the entire V-model: Development objectives, quality

objectives and, derived from that, requirements in a narrower sense have been set and documented for the Linux kernel and the other open-source components; however, they often do not meet industry standards. A planned, standards-compliant procedure starts with the selection of the components or a software version and their composition and optimization.

A large part of the maintenance activities during the life cycle regarding information security and quality is performed by the community [3]. This community identifies vulnerabilities or malfunctions and provides new versions (Figure 2). Aside from its open source code, this is, by the way, the key benefit of open-source software. A global community, basically consisting of developers from companies that use OSS, collectively continuously improve the code base.

All this is based on one of the largest international development projects [3], contractually protected by copyleft. Simply put, this practice grants all use rights to every user, subject to the condition that improvements be made available to the community. Today, Linux can be found in many critical areas: from air traffic control [8] and stock exchanges [9] to medical technology [6].

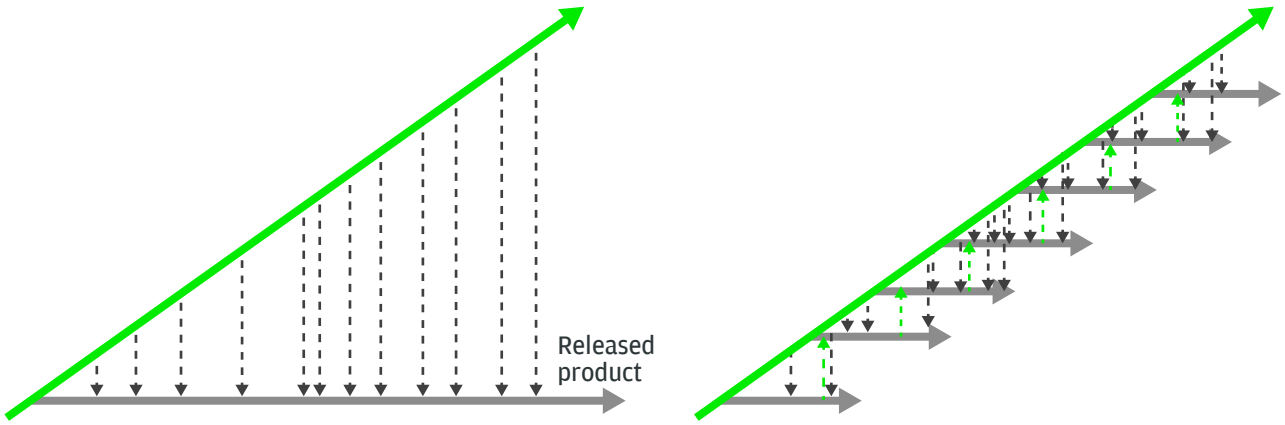
Standards-compliant "clearance certificate"

Formally, some standards allow developers to take Linux as "given" and to actively develop and configure it in a standards-compliant manner only after it has been applied to their own software system. In IEC 62304, this is done using SOUP (Software Of Unknown Provenance). According to IEC 62304, this is "a software item that is already developed and generally available and that has not been developed for the purpose of being incorporated into the medical device or software previously developed for which adequate records of the development process are not available".

In the automotive sector, ISO 26262 allows a similar procedure with the declared objective that "the re-use of qualified software components avoids re-development of existing software components [...] or enables the use of commercially available software (commercial off-the-shelf, COTS)".

This procedure is particularly interesting for an operating system as it implements essential mechanisms for the freedom from interference between safety-critical and non-safety-critical software components. Generally only this part of the operating system inherits the integrity level of the highest classification of all functions. This does not mean that the entire operating system is classified as safety-critical.

Safety requirements in functional safety always refer to the overall system and potential effects. A logical error that is unfortunately made often is to confuse the function with the components: Safety requirements are always assigned to requirements and not to components. As long as, for example, a failure of a component is



Stable version, selective change over a long period

Active switch to new mainline versions

Figure 2: Maintenance strategies when using a Linux kernel. (Source: Emlix)

reliably detected and cannot result in the violation of a safety requirement at the system level, it is not necessary to classify the component as safety-critical.

Mainline kernel: intensive testing ensures quality

When using Linux, an approach established in medical technology is to start directly from the original source. Such versions are called “mainline” or “vanilla”. Changes made there are subject to an established and strict review process by the community and so-called maintainers. They must release a piece of code before it is integrated into the kernel.

In addition, the quality of testing such vanilla kernels has significantly improved in recent years and the tests are performed by various community members. Worth mentioning in this context are, e.g., the Linux Test Project (LTP) [10], the continuous integration environment of the 0-day testing service [11], or syzkaller [12].

Once a vanilla kernel has been integrated into the project, changes directly to the kernel are no longer made. The project is “only” tasked with configuring and protecting in the project context. If the project requires extensions or direct changes – such as security patches –, they, in a first step, must not reduce the verification quality that has already been achieved. Even without normative requirements, it is strongly recommended that the components of which the overall system is composed be limited to those parts that are essentially needed and that further dependencies be deliberately excluded. This is the prerequisite for later system hardening [13].

Moreover, the recommendation significantly reduces maintenance and, if applicable, recertification as this requires that all changes to the mainline code be planned, documented, and tested as defined in the relevant standards to ensure full traceability of the changes and their verification.

Using test tools from the mainline community

Test suites developed and maintained by the community may well be used in quality assurance. Although they can generally not readily be run on embedded systems such as control computers, using them pays off. If they are correctly configured, they can at least be used to verify the expected behavior of a specific system’s components in the selected configurations.

Besides this, however, the reliability of the overall system is a key requirement. Against this backdrop, functional tests of the components used or of how they interact in a functionality are of great importance. All tests should be available in a reproducible manner.

This, in turn, should be part of a maintenance strategy. Systems in medical technology and automotive engineering are often used for several years. A maintenance and update strategy is crucial for using software in safety-critical areas, especially for systems with functional safety or information security requirements. This is as relevant to commercial software as it is to open-source software.

Considering IEC 62304, such a defined maintenance process must also exist for the Linux system. The Linux kernel is constantly improved by the community. So-called “long-term maintenance” is available for earlier versions. For example, version 4.9 was released on December 11, 2016 and is expected to be maintained until January 2023, i.e., for approximately six years [14]. For the use of Linux in a project, this means keeping an eye out for new versions and assessing their criticality and effect on the product.

The components of the Linux system are part of a list of all SOUP components that must describe not only the sources’ origins, but also the purpose of the software in the overall system. For all SOUP components, improvements, bug fixes, and new versions must be monitored and subjected to a predefined relevance test. In complex, software-based systems, systematic errors cannot be avoided despite the use of common industry standards. If, based on the defined criteria, a bug fix proves to be relevant to the system’s function or data security, it has to be applied.

However, these formal definitions do not render state-of-the-art development and testing of the software used and the overall system unnecessary. The system should have monitoring mechanisms to detect and appropriately respond to errors that can cause safety-relevant failures. This, too, is independent of whether or not the system used open-source software.

Achieving what is necessary for functional safety or information security is not set in stone – it is subject to a consensus among manufacturers, suppliers, and certification bodies that evolves over time. Therefore, collaborations such as the Sil2LinuxMP project at OSADL [15] or currently ELISA (Enabling Linux In Safety Applications) at the Linux Foundation [16] are helpful in developing this consensus.

References

[1] https://www.visualcapitalist.com/millions-lines-of-code/, retrieved 2019-03-03.

[2] https://en.wikipedia.org/wiki/ARINC_653, retrieved 2019-03-10.

[3] https://www.linuxfoundation.org/publications/2017/10/2017-state-of-linux-kernel-development/, retrieved 2019-03-10.

[5] https://www.auosar.org/, retrieved 2019-03-03.

[6] https://conf.qtcon.org/system/attachments/121/original/Gerloff.StrategicUseOfFreeSoftwareatSiemens.pdf%3F1473084229, retrieved 2019-03-06.

[7] https://en.wikipedia.org/wiki/Virtualization#Containerization, retrieved 2019-03-03

[8] https://archive.fosdem.org/2017/schedule/event/air_traffic_control/, retrieved 2019-03-03.

[9] https://www.pressebox.de/inaktiv/suse-linux-ag/Deutsche-Boerse-Systems-AG-konsolidiert-mit-SUSE-LINUX- Enterprise-Server/boxid/11948, retrieved 2019-03-03.

[10] https://linux-test-project.github.io/, retrieved 2019-03-03.

[11] https://01.org/lkp/documentation/0-day-test-service, retrieved 2019-03-03.

[12] https://github.com/google/syzkaller, retrieved 2019-03-03.

[13] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Leitfaden/GS-Leitfaden_pdf.pdf?__blob=publicationFile&v=3, retrieved 2019-03-06.

[14] https://www.kernel.org/category/releases.html, retrieved 2019-03-03.

[15] https://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html, retrieved 2019-03-05.

[16] https://elisa.tech/, retrieved 2019-03-05.

Author:
Alexander Much
Alexander Much studied mathematics and computer science at the Universities of Erlangen and Cambridge. After working as a Linux developer at SuSE Linux in Nürnberg, he joined Elektrobit Automotive in 2003. At EB, he is responsible for systems engineering, focusing on functional safety, information security, and open source.
Alexander.Much@elektrobit.com

Author:
Heike Jordan
Heike Jordan studied molecular genetics. After working freelance in nuclear medicine, she joined Emlix in 2006. In 2012 she became Managing Director, responsible for multi-project management, especially for development projects requiring certification such as security and life cycle maintenance.
hjordan@emlix.com

Elektrobit

emlix
embedded linux systems

About emlix

emlix' core competence is the development of embedded Linux-based system solutions for devices, plants and machinery in different industries. In our projects, we can rely on many years of experience in standards-compliant development in accordance with, for example, IEC 62304, WELMEC and Common Criteria. System solutions developed and maintained by emlix have been certified by FDA, TÜV, PTB and BSI successfully.

Product-specific software platforms from emlix are successfully in use in millions of devices – safe and secure.

emlix GmbH
Gothaer Platz 3
37083 Goettingen, Germany
www.emlix.com

emlix.com

About Elektrobit (EB)

Elektrobit (EB) is an award-winning and visionary global supplier of embedded and connected software products and services for the automotive industry. A leader in automotive software with over 30 years serving the industry, EB's software powers over one billion devices in more than 100 million vehicles and offers flexible, innovative solutions for connected car infrastructure, human machine interface (HMI) technologies, driver assistance, electronic control units (ECUs), and software engineering services. EB is a wholly owned subsidiary of Continental.

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
www.elektrobit.com

elektrobit.com