emlix
embedded linux systems

White Paper

emlix Yocto Framework

Design, composition and maintenance of Yocto-based Linux systems

by Martin Homuth

emlix Yocto Framework

Design, composition and maintenance of Yocto-based Linux systems

by Martin Homuth

Introduction

The Yocto Project is a frequently used starting point for the development of industrially used embedded Linux systems. Product management, development management and Linux developers are challenged by the question of how project risks for Yocto Linux-based industrial products can be systematically reduced. Against the backdrop of the EU Cyber Resilience Act and the Directive on Security of Network and Information Systems (NIS-2), the topic of security by design is also becoming increasingly relevant. A controlled development process is a success factor for minimizing the costs of maintenance and CVE security monitoring over the life cycle.

This article shows the positive economic and technological effects of using a framework consisting of a tried-and-tested process model and adaptable and reusable building blocks as a starting point for the composition of product-specific Yocto systems. The focus is on the typical requirements of an industrially used Yocto system: reusability, reproducibility, maintainability and security as well as reliable version control.

The outlined framework can be used both in the context of new developments and for the maintenance and (security) update of existing Yocto systems. It offers all those involved in product development a strategically based approach and cost-efficient technology planning.

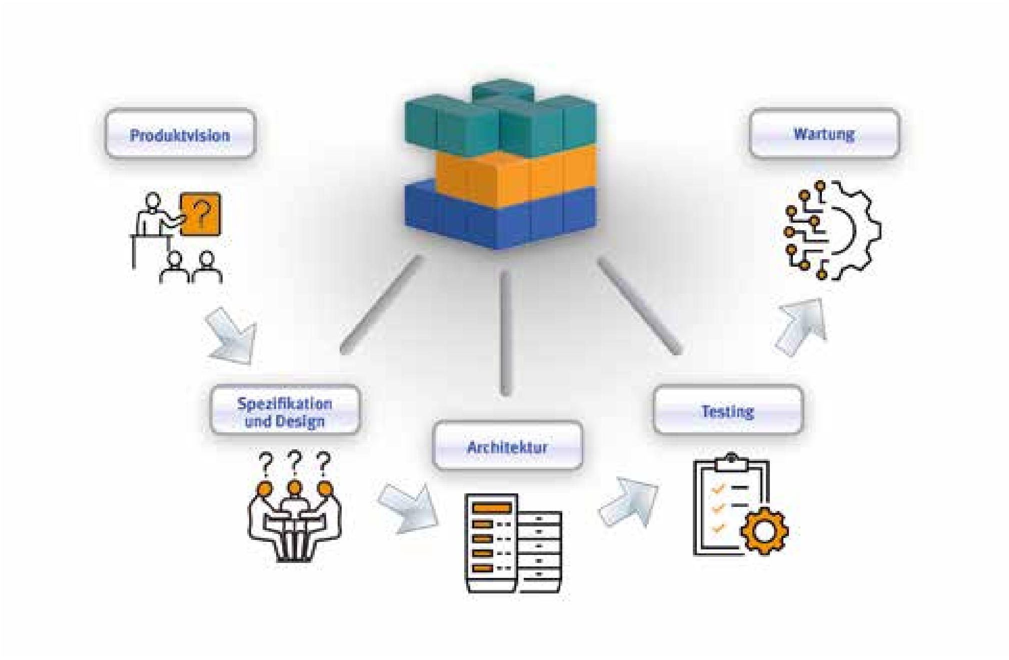
Yocto as de facto standard for embedded Linux systems

OpenEmbedded has developed as a building system of the Yocto project into a de facto standard for industrial embedded Linux systems. Due to the size, structure and complexity of the Yocto ecosystem, it is a technical and organizational challenge for developers to create product-specific, maintainable, secure and at the same time reusable systems and to maintain them over the typical life cycle of products.

Strategically based project and technology planning is necessary in order to develop industrially used embedded Linux systems against defined requirements and, where applicable, normative requirements. In order to proceed cost-efficiently, proven solutions and implementation concepts should be used. This also serves another central requirement: the maintainability of the embedded Linux system over the product life cycle.

In addition to the meta layers located and maintained in the Yocto project itself, there are a large number of other layers that are provided from specific project contexts or by hardware manufacturers, but are not necessarily incorporated into the Yocto or OpenEmbedded project and thus with the same quality requirements.

Despite an extremely active community and established best practices, there is a very high degree of freedom in the creation of Yocto systems, which in practice leads to problems with long-term maintenance, compliance and implementation of security requirements or consolidation efforts of several systems. The challenge is to identify and create the right project combination from all the available recipes and configurations. However, the flexibility and wide range of functions can be easily controlled using established design approaches, processes and tooling tailored to the development process.



Complexity reduction through defined processes

Project risks, in particular the non-economic viability of additions or maintenance costs, can be countered with an adequate process model when compiling and developing Yocto systems. The starting point is the analysis of the underlying requirements and an evaluation of which successful approaches and solutions can be used in the field of Yocto systems in use.

The functional and non-functional requirements often result from competing influencing factors. These requirements can be easily divided into primary and secondary requirements, whereby the primary requirements are always the focus and secondary requirements are only partially focused on or only at a later stage of the project. Primary requirements are the specific product requirements that define the hardware configuration and package composition of the system, as well as legal requirements. Secondary requirements are the maintainability and reproducibility of the system, open source software license compliance, general system security or requirements for consolidating multiple software platforms into a common infrastructure for build and maintenance tasks.

Secondary requirements are sometimes not incorporated into the system design, or only after a delay: Due to a lack of time or resources, they are only addressed at a late stage of the project and thus generate considerable effort and risks. This ranges from downstream configuration of all software components to the need to completely refactor the system.

A well-structured process model with corresponding processes supports the early and efficient implementation of the various requirement classes.

System design with the help of ADRs

A large number of very different development projects for applications in various industries have resulted in a model for the planning, implementation and maintenance of Yocto projects that has grown out of practical experience. A Yocto systems engineering process with building blocks typically starts with an analysis of functional and non-functional requirements and an initial system architecture. The aim is to define the basic core components and interfaces of the system. Definition gaps, incorrect assumptions or degrees of freedom become apparent, which must be clarified between the Product Owner and Development in the course of the project.

Based on the requirements analysis, a system architecture is developed with the help of building blocks. A building block is an independent, specific function or subtask of the system, such as Trusted Boot, Mandatory Access Control or VLAN, and serves as a topic checklist in this situation. The entirety of all building blocks makes it possible to ensure that all relevant topics have been addressed and considered in the system design. For example, if you look at the storage of configuration data, the "Data migration" building block automatically raises the question of whether and how this should be considered.

The iterative system design, the possible technical implementations and also the detailed processing of the specific requirements lead to controlled decision-making processes that must be carried out together with the product owner. This is documented in a comprehensible and continuous manner with so-called ADRs (Architectural Decision Records), in which all alternatives to a problem, such as which boot loader is used, are prepared with their advantages and disadvantages and a decision is made together with the customer. ADRs are an adequate method of documenting and referencing all decisions in the product lifecycle.

Yocto Building Blocks

A Yocto Building Block is a delimited functionality in the form of recipes or configurations together with its documentation, which can be reused with appropriate adaptations in different products or systems with similar requirements. It allows as few implementation details as possible to be defined individually, most of which are identical across projects.

By working out a functionality in a Yocto Building Block, it can be improved in the long term so that it can be dynamically applied to different application contexts or split up for further specialization. Different Yocto Building Blocks can be combined with each other in the uprooting process in order to realize more complex product features.

System development with building blocks

Building Blocks provide the basis for structured, efficient and scalable development and maintenance of Yocto systems. At the same time, Yocto Building Blocks minimize risks resulting from the inherent complexity and varying quality of the Yocto project.

According to the model outlined above, a Yocto project ideally starts with a defined baseline, which typically consists of three things:

1. A minimal Yocto configuration as a starting point
2. Initial hardware bringup (e.g. ramdisk, kernel, bootloader)
3. A suitable methodology for layer and configuration management

The aim of the minimal Yocto configuration is to initially have as few unknowns as possible in a system. These include unwanted dependencies and components that enter the system via package groups or inheritance. The system composition is reduced to a minimal Linux system with a generic kernel (also with minimal configuration), an init system and a shell, such as the busybox. From there, every addition to the system can be made consciously and documented.

In the next step, the focus is on hardware commissioning. The board manufacturer's Yocto layers are generally used as the basis for this. They offer the advantage of functioning and tested configurations of Linux and kernel for the respective board as well as additional components such as the ARM Trusted Firmware or the compilation of a boot image.

Whether the management of layers and configurations is carried out on the basis of scripts, git submodules or a dedicated tool such as kas[1] is not just a question of individual taste, but must be chosen with particular regard to the maintainability of the implementation and the development methodology. A building block that maps this fundamental decision and allows it to be reused is ideal here.

The advanced composition process of Yocto systems is primarily based on selecting and qualifying suitable layers with regard to the functional requirements, integrating the corresponding components and configuring these or the system itself. The layers are qualified on the one hand on the basis of individual experience, and on the other hand it must be analyzed and evaluated whether a specific requirement can be fulfilled with the respective layer. Yocto Building Blocks make it possible to underpin this individual decision with concrete, tried-and-tested implementations and broader empirical knowledge.

For Yocto projects that have grown over a long period of time and have been exposed to different requirements, it can be worthwhile to restart the system composition using the defined baseline. This allows legacy issues to be removed in a structured manner, alignments to be reviewed and the entire system to be restructured in an orderly manner in order to ensure maintainability and updatability.

Examples of Yocto Building Blocks

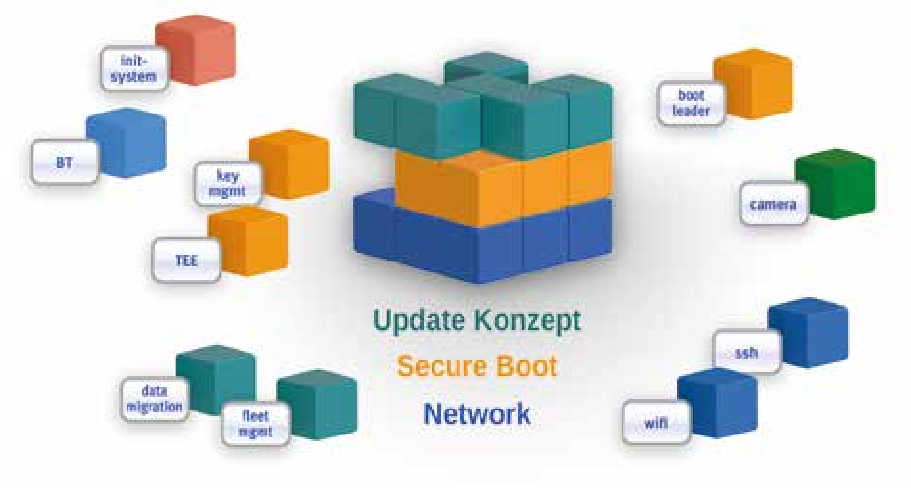
From many years of Yocto-based projects, regularly recurring tasks and functional requirements can be identified, which can be summarized thematically within the Building Blocks and a number of reference implementations can be derived. A Yocto Building Block exists not only at the level of pure system composition, but also at the overlying infrastructure level, for example in the context of CI. They comprise three areas:

1. product-unspecific and minimalist reference implementations including documentation of recurring problems
2. a collection of product-specific implementation examples from which sub-aspects are adopted
3. knowledge and experience of what has proven successful in the use and implementation of the building blocks and what should be avoided

A Yocto Building Block therefore develops from a specific context or task. The aim is not to define a one-size-fits-all solution, but to provide a cross-project pool of implementations and an initial implementation approach. Neither the reference implementation nor the implementation examples can be integrated and activated in a project without adaptation and it is essential to document the respective core information for adaptation and classification in detail. However, this offers the potential to have already defined a building block for all issues and to be able to use this as a starting point for development.

A selection of Yocto Building Blocks also makes it easier to fulfill the functional requirements in that a large part of the subsequent implications are already known and can be evaluated during the selection, i.e. in the design phase. There are sometimes several alternative options that have proven themselves in the field and whose boundary conditions have been formulated.

The following examples of building blocks illustrate their added value.



Minimal Yocto Configuration

When a Yocto project is launched, there is always the question of a starting point. The core problem has already been described above: reducing system complexity to a minimum. A Yocto Building Block, which answers the following recurring questions, is ideal for this:

1. How is the project structured in terms of layers and configurations
2. Which minimum configurations are required
3. How continuous integration is implemented for the project

The structural questions are easily defined and transferred to a corresponding layout, and standardized usage helps with orientation in different projects. Depending on the CI platform used, a configuration can be provided for Gitlab or Jenkins, for example, which provides the project directly in a buildable state in the infrastructure and uses central caches where possible to speed up builds and protect the infrastructure.

In terms of component configurations, each project starts with a generic kernel with a minimal configuration without any drivers and the most basic functionality: including device nodes, logging, scheduling and a console. As a Yocto distribution, poky is used to provide the necessary toolchain configuration, but the additional packages are explicitly deactivated. This results in a minimally bootable system with a busybox as the starting point, without any superfluous content. Based on this, all further additions can be integrated point by point.

Networking

Various requirements relating to the network configuration of an embedded Linux system can already be fulfilled using various switches in Yocto. However, this tends to take place at a very basic design level or at packet level. The actual network configuration, on the other hand, is expected to be so project-specific that it does not make sense to define templates, such as a VLAN configuration.

However, a corresponding setup can be prepared for specific configuration types which, depending on the Init system used, can be easily integrated and used as a solid starting point for further detailed configuration. An example of this is the configuration of 2 network interfaces, one of which is to be used for internal and one for external communication.

Update concept

An update concept for an embedded Linux system has a large number of variables that need to be considered. What exactly is updated, which tools are used, what features does an update package have, how are updates managed or how is data migration carried out are typical questions when implementing such a concept. Security and availability requirements are almost always added to this.

Corresponding Yocto Building Blocks make it possible to collect implementations and use them as a reference. This means that both conceptual and implementation building blocks can be used, even if almost every update solution has specific sub-aspects that need to be individually designed and implemented.

Specific examples of Yocto Building Blocks are various permutations of boot loaders and update tools such as u-boot, barebox, SWUpdate and RAUC. Any successful implementation in the project context can be transferred to a building block or used to improve an existing block.

Variant management

An important block is the definition of system variants. A distinction is made here primarily between test, production and development systems. In this context, there are various requirements for which a building block of the "knowledge" type simplifies the implementation:

For example, how is it ensured that a development system offers as many degrees of freedom as possible and still corresponds as closely as possible to the production system. How can a system be designed in which a production system can also be debugged as far as possible without generating security problems at the same time? Which approach has proven to be problematic and possibly hindered development?

Specific recommendations and best practices on how different variants of the customer-specific Yocto Embedded Linux platform for different product versions should be managed in the best possible way in an overall system are also a knowledge-based building block.

Container engine

Depending on the requirements, it may be necessary to provide a container engine in the system. Regardless of whether this involves application isolation, an (externally) defined execution environment or simply a business decision to completely separate application and system development, the key question is whether an engine should be used at all and, if so, which one. In many cases, a simple solution based on namespaces, chroot or simple rootfs containers is sufficient. However, it may also be necessary to use OCI containers, for example because there are already defined processes or further tooling is to be used, such as a container registry.

A minimal Yocto Building Block can be defined for each of these outlined use cases, which implements the respective scenario, whereby gradations are also possible here. Only the engine itself, accompanying tools or the entire ecosystem can be provided.

Testing and CI infrastructure

Testing is a very wide-ranging topic and covers several levels:

1. unit tests within the components
2. package tests of Yocto packages
3. general system tests
4. regression and smoke tests

All of these areas should ideally be testable by a CI infrastructure. Even if components can only be updated as soon as the unit tests are green, it is helpful to verify them at system level. This is realized in Yocto with ptests[2], where packages provide their tests with two components: an executor and the tests themselves. The test system is executed on the target hardware and contains a corresponding runner that locates all installed package tests and executes them. Ultimately, hardware in the loop can be used to validate individual components.

Building blocks allow the practical implementation to be reused in a company and thus to expand a project with a test infrastructure and integration in a highly efficient manner without having to redefine all aspects and create the corresponding infrastructure. Individual project properties can be easily integrated thanks to the open model.

Security and standards compliance

Building blocks have a claim to completeness in their entirety. This means that there are no interfaces and system aspects in a typical embedded Linux system that are not addressed in the system architecture. This means that key system security issues are also considered in advance and embedded in the system design. If, for example, system users are able to interact with the system at a low level, a rights and roles concept must be provided and maintained from the outset. Suitable Yocto Building Blocks for the use of SELinux, for example, can thus be identified and referenced.

Building blocks can also be used with regard to specific standards in order to collect and coordinate all relevant influencing factors. By focusing on a specific standard, everything relevant to the system can be summarized in this context and broken down to the content to be implemented. Not only the characteristics of the embedded Linux system are taken into account, but also the requirements for the infrastructure. For example, IEC62304 requires so-called SOUP (Source Of Unknown Provenance) documentation for medical devices, which should be embedded in the infrastructure in a similar way to SBOM generation, from the construction process through to document generation.

Resources in the company - Yocto Workbench

Yocto Building Blocks and Building Blocks for application-specific use cases can be easily combined to form a workbench for Yocto systems. A Yocto Workbench is therefore the central repository for qualified, tested and product-specific adaptable Yocto components including a reproducible infrastructure for build and CI pipelines.

An internal Yocto Workbench can be used as a starting point for the development of all Yocto-based products. At the same time, it allows the consolidation of development and maintenance processes. For the implementation of requirements from standard, security and supply chain obligations, it forms the basis for corresponding verification documents.

Summary

The outlined development model shows the positive economic and technological effects of using customizable building blocks, proven processes and tools as a starting point for the composition of product-specific Yocto systems. The definition and use of building blocks makes it possible to design embedded Linux systems more efficiently and to systematically ensure that the resulting project meets a wide range of requirements. A key aspect of this is the more efficient (or even possible) maintenance of the Linux platform over the lifecycle of an industrial product.

The Yocto Systems Engineering process with Building Blocks allows a systematic and well-founded planning and reusability of components, configurations and concepts for the implementation of defined functional properties of a Yocto system instead of the primarily layer- and recipe-driven compilation of a system “from scratch”.

Bibliography

[1] https://kas.readthedocs.io/en/latest/
[2] https://wiki.yoctoproject.org/wiki/Ptest

About the author

Martin Homuth is a system engineer at emlix GmbH. He studied computer engineering at the TU Berlin, specializing in operating systems and machine learning. In almost 10 years at emlix, he has gained extensive experience in the design, development and maintenance of embedded Linux systems. Today he is a project manager, Yocto expert and consultant in a diverse project landscape.

About emlix

emlix offers industrial-grade embedded Linux and Android systems for the digitalization and secure networking of devices, machines and plants.For more than 25 years, emlix has been transferring system knowledge, innovations from the open source universe and market knowledge into the products of more than 350 customers in automotive, energy industry, automation technology, medical technology, safety technology and others.

As a provider of professional open source software, emlix ensures process reliability and transparency. The tools and development standards emlix uses are designed for industrial requirements and certifications. They offer long-term maintenance services and CVE montoring for their software platforms and thus assume responsibility for the product life cycle and the investments of their customers.

Contact

Should you have any further questions, please do not hesitate to contact us:

emlix GmbH
Berliner Straße 12
37073 Goettingen
Germany

www.emlix.com
solutions@emlix.com
+49 (0)551/30664-0