DEVELOPMENT OF AN OPEN AND MODULAR PLATFORM FOR HYDRAULICS TO INCREASE PRODUCTIVITY AND FLEXIBILITY

Marco Genise[1]*, Peer Mumcu[2]

[1]*Bosch Rexroth AG, Partensteiner Str. 23, 97816 Lohr a. Main*
[2]*Bosch Rexroth AG, Lise-Meitner-Straße 4, 89081 Ulm*

* Corresponding author: Tel.: +49 9352 18-6703; E-mail address: marco.genise@boschrexroth.de

ABSTRACT

This document describes the challenges and solution strategy for decoupling software solutions that implement hydraulic control functions from their hardware and unifying them into an open and modular software platform for hydraulics. The challenges we identified primarily concern the software engineering domain, which may be surprising since our starting point and even the resulting software building blocks are all about controlling hydraulic systems. Thus, the key solution elements are based on software engineering principles: computer hardware abstraction and model-based software development.

Keywords: Hardware Abstraction, Model-Based Software Development, Licensing, Digitialization, Software Engineering

1. INTRODUCTION

Over the last decades, hydraulic systems have evolved from purely mechanical to electromechanical systems. With the evolution of digital electronics, software emerged and took over more and more functions that were previously either realized in an electromechanical way, or perhaps not even possible. These software solutions were tightly coupled to their hardware and limited to the hydraulic setups of their actual intended target industrial applications.

This tight coupling of software and hardware resulted in a variety of different software solutions with inconsistent control behavior, user interfaces, and architectures. Subsequently, this inconsistency made it difficult for customers to understand and use all these different solutions in an efficient way. As it concerns Bosch Rexroth Industrial Hydraulics, it was rather difficult to maintain and adapt to new customer requirements.

Thus, the next step in digitalization is to decouple software solutions from their hardware and unify them into an open and modular software platform for hydraulics. Using the principles of software development, hydraulics know-how can be encapsulated in hardware-independent software building blocks that can handle the wide variability of hydraulics systems in a compact manner.

The resulting software building blocks increase usability by allowing users to implement them in their familiar development environment, perhaps even by eventually implementing them on an existing machine control device rather than in a separate motion controller. In addition, these software building blocks can be used throughout the life cycle of a machine, for example in a simulation environment during planning and design, see Figure 1.

In the following, Section 2 describes the challenges to unify these disparate software solutions into a software platform for hydraulics, followed by Section 3, which presents the solution strategy using

software engineering principles. Disadvantages of software-heavy approaches and measures to mitigate them are presented in Section 4. Finally, Section 5 concludes this paper.

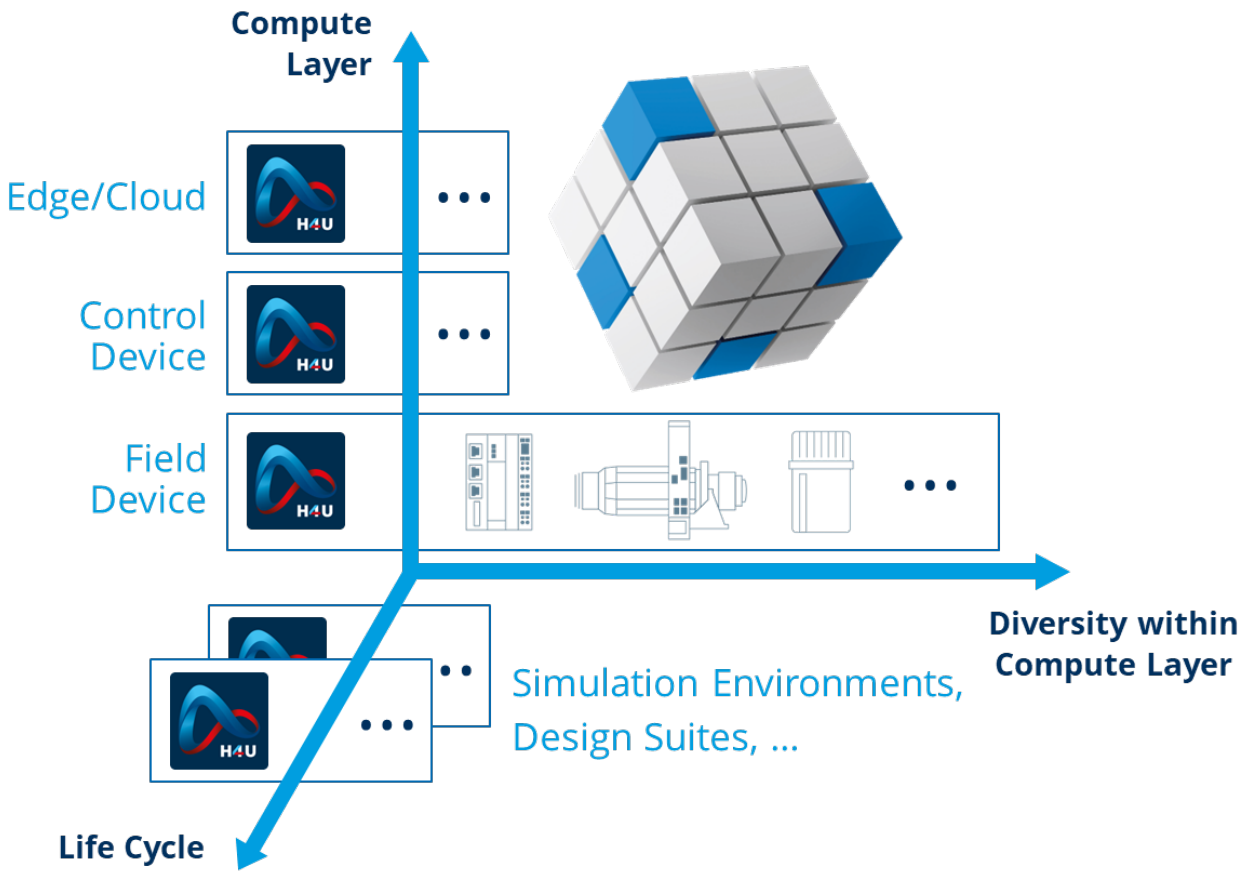

Figure 1: In this figure potential execution environments for software solutions are illustrated. The compute layer represents an aspect for different classes of computer hardware, for example field devices, control devices and edge devices. For each such class of computer hardware there are various realizations, for example the class of control devices comprises many different products from various manufacturers. This diversity within one class of computer hardware represents another aspect. The third aspect concerns the influence of the life cycle stage on the execution environment. For example, during productive use an application software is typically run on a dedicated hardware such as a control device, but during development and testing often virtual simulation environments realized with common PC hardware are used.

2. CHALLENGES

Software is commonly understood as all non-technical-physical functional components of a computer system, typically programs and the associated data. It determines the function of a software-controlled device and how it works.

As mentioned above, current software solutions already provide most of the functionality needed to control hydraulic systems, but they are still specialized for the hardware products they were designed to support. It is then, by definition, trivial to separate the functionality from the hardware. The real challenge is to unify these disparate solutions into a single software platform that can flexibly handle the variability of computer systems and hydraulic systems.

2.1. Variability of Computer Systems

The decoupling of software solutions from the computer system on which they may finally be deployed opens the opportunity of targeting broader applications than today.

A decoupled software solution is no longer limited to being implemented and used within only one compute layer. For control tasks, the possible deployment target is now a question of sufficient performance of the communication channels between the computer systems and the actuators and sensors, regardless of whether it is a traditional field device, a control device or perhaps even an edge or cloud solution. As an added dimension, the same software solution can now also be used in a different environment, such as a simulation environment or a design suite (Figure 1).

The challenge lies in the heterogeneity of (virtual) computer systems. Software can be deployed across several computing layers, in the diversity within each layer, and to other targets along a machine's lifecycle (e.g., simulation environments). Across all of these (virtual) computing systems, we encounter different computer architectures, operating systems, programming systems, and application ecosystems.

2.2. Variability of Hydraulic Systems

A major advantage of hydraulic systems is the tremendous variability of feasible solutions for a given task. Figure 2 shows some possible hydraulic setups to build a pressure and flow-controlled subsystem (p/Q control) with a motor-pump combination. The actuating variable may be the motor speed, pump displacement or even a combination of the two. Having multiple control elements also enriches the simple p/Q control with the ability to perform higher-level functions such as operating point optimization – an added value that can be dynamically provided in a software solution.

This variability is also the greatest challenge when trying to standardize higher-level functionality, such as p/Q control, in a single solution.

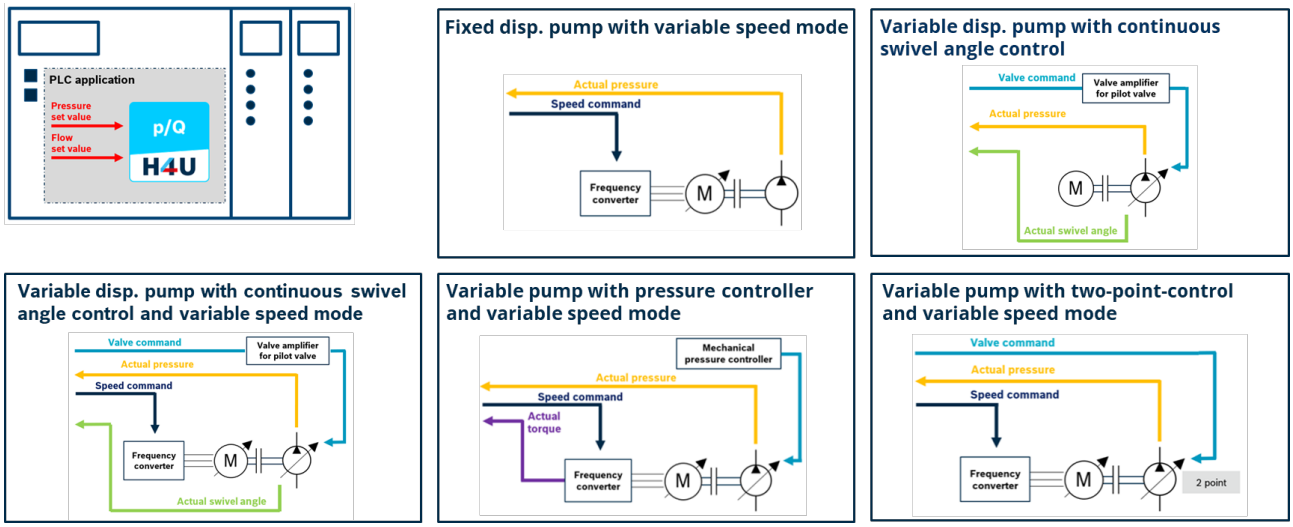

Figure 2: Examples of pressure and flow-controlled motor-pump combinations.

3. SOLUTION STRATEGY

The challenges described above should be primarily regarded as software engineering challenges. This may be surprising since our starting point and even the resulting software building blocks are all about controlling hydraulic systems. However, accepting the increased importance of software makes it less surprising that the key elements of the solution that we propose are based on software engineering principles: computer hardware abstraction and model-based software development.

3.1. Defining Computer Hardware Abstraction Layers

"Separation of concerns" [2] is a fundamental principle of software development that divides an application into distinct sections. Separation of concerns is achieved by encapsulating knowledge within a section of code that has a well-defined interface. This results in a high degree to which parts within one section belong together (strong cohesion) and a low degree of interdependence with parts of other sections (low coupling).

When applying this principle to computer systems, we can design computer hardware abstraction layers by identifying logical layers in data/signal processing, separating these layers from each other (decrease coupling), and making each layer its own compact section of parts that belong together (increase cohesion).

Figure 3 shows the chosen layer architecture, which is common in other industrial standards such as AUTOSAR [3]. The top layer, the device function layer, implements the actual hydraulic control functions in a hardware independent manner and has a well-defined interface based on physical dimensions used in the application domain. The middle layer, the control device abstraction layer, serves as an abstract binding between the hardware-independent device function layer and the actual (potentially virtual) computer system. It provides machine-level abstraction by adapting system calls and converting data types, and transforms this data into the semantic information required by the device function layer.

The most important advantage of this approach is that a simulation environment or design suite can now be treated as a virtual computer system. This allows the device function layer to be used there by providing an appropriate control device abstraction layer, e.g., to enable application testing in a virtual environment.

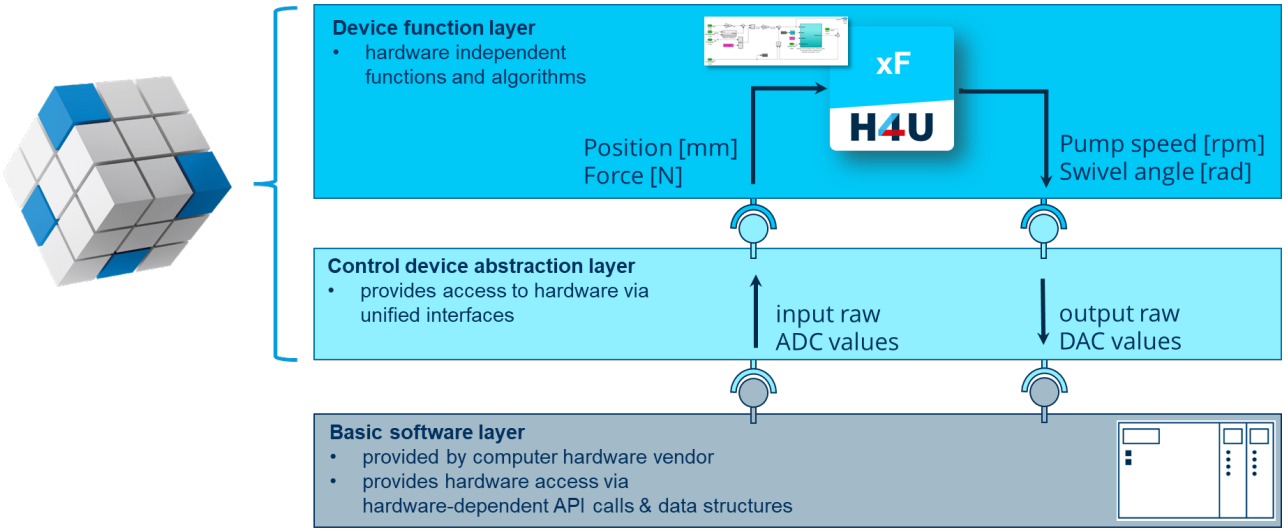

Figure 3: Layer architecture of Bosch Rexroth's Industrial Hydraulics software platform H4U.

The example shown in Figure 3 comprises reading sensors with electrical interfaces, processing the signal through the control device abstraction layer into physical units as input to the device functions, and then reversing this chain to finally output a voltage or current signal to control the actuator. The application in the device function layer is relieved from concerns about how a particular piece of electronics reads voltage signals, what kind of corrections or calibrations need to be made, or how it generates its output signals.

This principle is not limited to the abstraction of analog electrical interfaces, but also applies to sensor and actuator signals transported over digital interfaces or fieldbuses.

3.2. Model-Based Software Development

In software engineering it is common practice to express domain aspects in a language that fits the domain. This is a result of the understanding that general purpose programming languages are often difficult to use as a means of communication between the functional domain and the software engineering domain. In addition to these domain-specific languages, there are entire development methodologies that reflect the domain focus, such as Domain-Driven Design [4].

The idea behind model-based software development is to describe the actual system functions in a language familiar in the system domain to express its structures and processes on a higher level of abstraction. This language abstraction is another fundamental principle of software development. Here, MATLAB Simulink is used to express the system functions in a language that is much closer to the domain of hydraulic control than general-purpose programming languages such as C, see Figure 4.

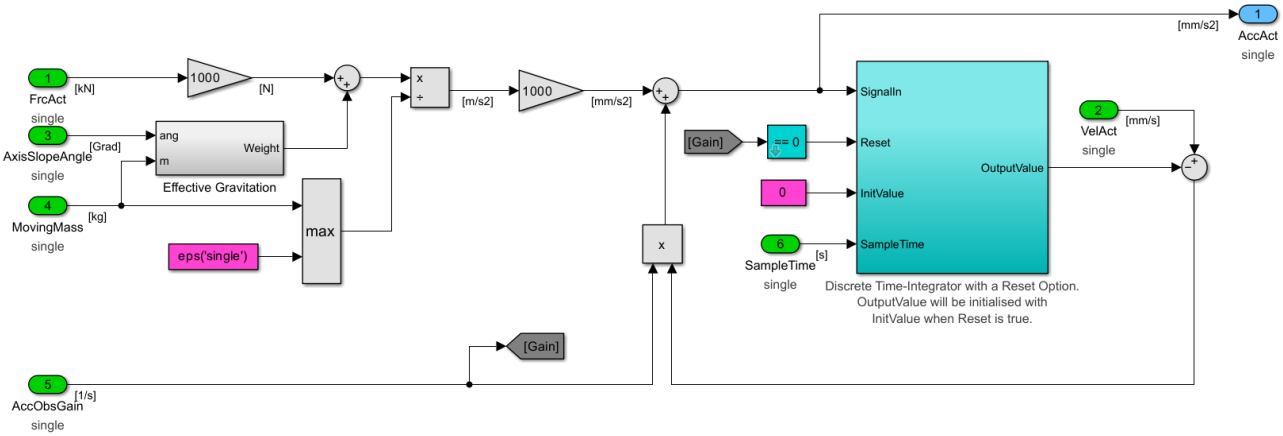

Figure 4: Example: Luenberger observer implementation in MATLAB Simulink.

Since the modeling language is a formal language, the models can be translated into a semantically equivalent program in another formal language, such as IEC 61131-3 Structured Text or C. This principle is used to compile the device function layer into program code for different development environments like TIA Portal, TwinCAT or ctrlX PLC, as shown in Figure 5. This process is commonly known as code generation.

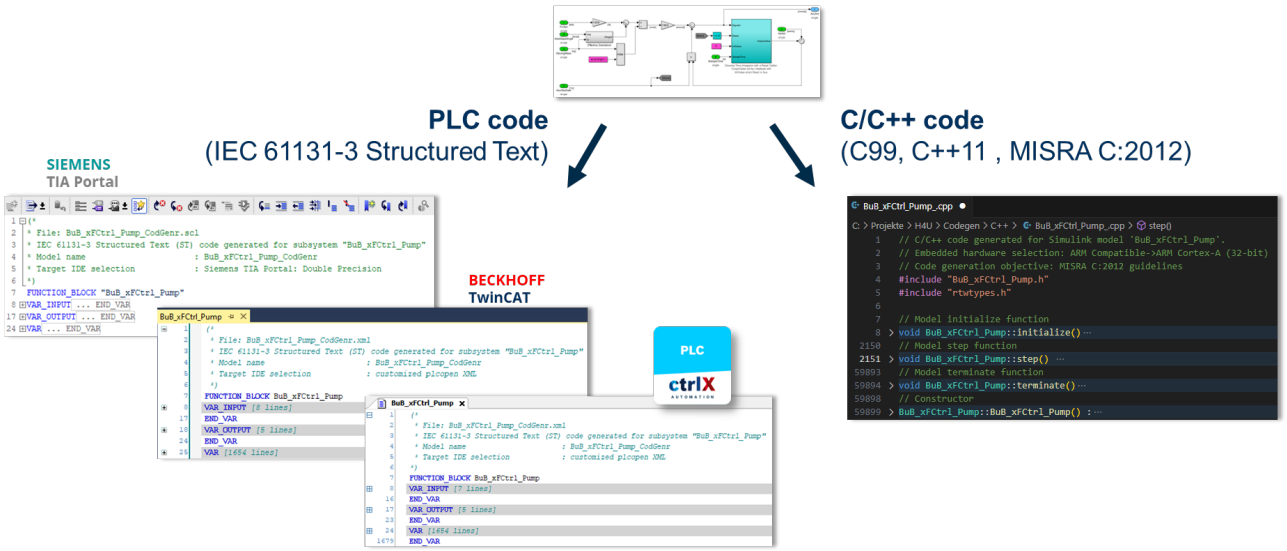

Figure 5: Compiling model to target specific programming languages.

Model-Based Hydraulic Functions Are Software

As mentioned above, model-based software development still results in programs. Therefore, these programs are subject to the principles of software development. To manage the complexity of the hydraulic control function, separation of concerns is used to separate layers, as shown in figure 6. In our implementation to control position and force for a differential cylinder in a closed hydrostatic transmission, the control elements are decoupled using separate layers: a control-layer, the inverse (flow → linear motion) transformation layer and the inverse (rotary motion → flow) transformation layer, representing control, cylinder, and motor-pump units. Any modification in the axis arrangement will concern the arrangement of motor-pump units and will therefore restrict software variations to the last functional layer [1].

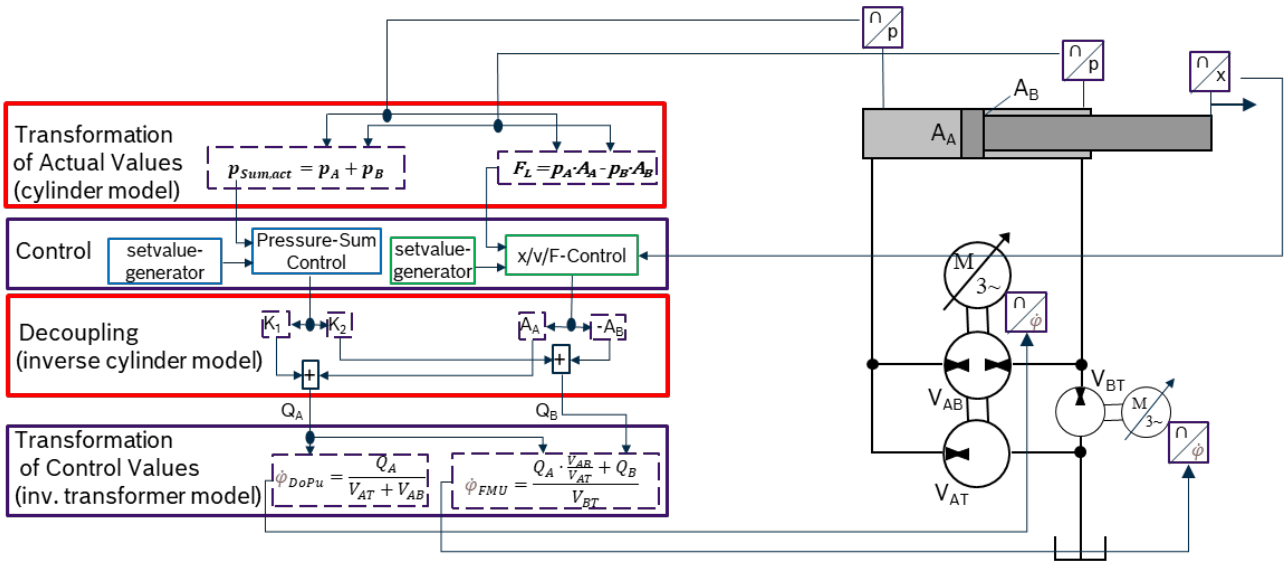

Figure 6: Example: software structure of a position- and force-controlled application [1].

4. DISADVANTAGES OF SOFTWARE-HEAVY APPROACHES

By introducing software-heavy approaches into hydraulics development, we not only gain a lot of efficiency advantages, but we also have to deal with new challenges, namely intellectual property theft and general IT security issues.

By decoupling software functions from their hardware, another fundamental property of software emerges: the marginal cost of replicating software is essentially zero. To avoid economic damage, additional measures must be taken to mitigate the uncontrolled distribution of such software solutions. Because software is usually easily modified, it is also necessary to detect modifications and protect integrity. Otherwise, the proper functioning of the software solution cannot be guaranteed.

Typically, these measures are implemented using cryptographic primitives. However it should be clear that the success and residual risk of any of these measures depends on the level of trust available in the target computer system.

4.1. Cryptography Primer

This section provides a basic introduction to cryptography, which is helpful in understanding the measures taken to mitigate the disadvantages of software-heavy approaches.

A cryptographic primitive is a well-established, low-level algorithm that is used to construct more complex cryptographic systems. These algorithms include one-way hash functions and encryption functions.

A one-way hash function is a function that takes arbitrary data of any length and outputs a short, fixed-length hash value. A key property of a one-way hash function is that it is unfeasible to find input data that matches a known hash value (also called pre-image resistance).

Encryption functions create an unintelligible blob of data (also called ciphertext) from the actual data in order to ensure its confidentiality. The operation of an encryption function usually depends on a piece of auxiliary information called a key. Without knowing the key, it is extremely difficult to decrypt the resulting ciphertext back to the original input data.

In public-key cryptography – also called asymmetric cryptography – two keys are used, one for encryption and one for decryption. Depending on the application, one of the two keys can be made public, that is, not kept secret. For example, if a sender, Alice, wants to send a message to Bob, Alice uses Bob's public encryption key, which is made available to anyone who wants to encrypt a message and send it to Bob. Alice then encrypts the message using Bob's public key before sending the ciphertext to Bob. Only Bob can decrypt the ciphertext using his private decryption key, which remains secret.

4.2. Integrity Protection

Digital signature mechanisms are usually used to detect modifications and protect integrity of a software building block. A digital signature is essentially a hash value that is encrypted with the private key of an asymmetric cryptographic algorithm. The hash value is generated by a one-way hash function given the software building block as input data. This digital signature is distributed with the software component.

When the software starts, it computes the hash value of itself, decrypts the digital signature with the public key, and compares the hashes - and refuses its operation if the hashes aren't equal, meaning the software building block has been modified.

4.3. Mitigation of Uncontrolled Distribution

A common solution to mitigate uncontrolled distribution is to link the software application to a physical reference point using licensing mechanisms. For example, a physical reference point, such as a unique hardware identifier, is placed in a file. This file is then digitally signed as described above. This file and its digital signature make up the license file that is provided to the software application.

At startup, the software verifies the validity of the license file by proving that it has not been modified. It then compares the physical reference point stored in the license file with the one provided by the computer system. If either of these steps fails, the software will refuse its operation.

5. CONCLUSION AND OUTLOOK

In conclusion, this paper provides an exploration of the challenges in developing a capable software platform for hydraulic control functions. Recognizing the pivotal role of software engineering principles in overcoming these challenges, our proposed solution strategy hinges on two fundamental pillars: computer hardware abstraction and model-based software development.

However, the proposed transition to a software-heavy approach also introduces challenges such as intellectual property theft and security concerns. The outlined mitigation measures serve as initial safeguards. Continuous research and development are imperative to fortify these measures, ensuring the integrity and controlled distribution of such software solutions.

Looking ahead, the envisaged software platform holds the potential for deployment on cutting-edge computing architectures, including edge and cloud solutions, extending its reach beyond traditional

hardware constraints. This expansion not only ensures scalability but also positions the hydraulic control functions to operate seamlessly in distributed and connected environments, opening the possibility for regular updates, which may provide optimizations and support for new hydraulic components. Furthermore, the proposed approach establishes a fertile ground for applying data analytics. By consolidating data from diverse sources, the software platform can enable its users to leverage analytics for various purposes, for example performance optimization and resource utilization.

The outlook for the proposed software platform extends beyond the immediate challenges addressed in this paper, envisioning a future where hydraulic control functions not only meet the demands of today but also adapt and thrive in an ever-evolving technological landscape. The integration of edge and cloud computing, coupled with internet-enabled updates and data analytics, positions the software platform as a dynamic and future-ready solution, driving efficiency, sustainability, and innovation in hydraulic systems.

REFERENCES

[1] Bonefeld R, Gellner T, Brandt L, Bauer K (2024) Precise hydrostatic Cylinder Drive with increased Pressure Level for Industrial Applications. 14th International Fluid Power Conference. Dresden, March 19-21, 2024, Dresden, Germany

[2] Dijkstra, Edsger W (1982). On the role of scientific thought. In: Selected writings on Computing: A Personal Perspective. Texts and Monographs in Computer Science. Springer, New York, NY

[3] AUTOSAR (AUTomotive Open System ARchitecture). https://www.autosar.org

[4] Evans, Eric (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, Boston