applied sciences

MDPI

Article

COTS-Based Architectural Framework for Reliable Real-Time Control Applications in Manufacturing

George K. Adam [1,*], Nikos Petrellis [2], Georgia Garani [3] and Tilemachos Stylianos [1]

1 Department of Digital Systems, University of Thessaly, 41500 Larisa, Greece; tilemaxos@uth.gr
2 Department of Electrical and Computer Engineering, University of Peloponnese, 26334 Patra, Greece; npetrellis@uop.gr
3 General Department, University of Thessaly, 41500 Larisa, Greece; garani@uth.gr
* Correspondence: gadam@uth.gr; Tel.: +30-2410-684596

Received: 26 March 2020; Accepted: 2 May 2020; Published: 6 May 2020

Featured Application: The proposed architectural framework may benefit the design and development methodologies of systems based on commercial off-the-shelf (COTS) components, with its enhanced approach in building reliable real-time control systems. Moreover, this architectural framework broadens engineering ideas and improves the solutions of such systems by increasing the trust and reliability of COTS components in manufacturing and reducing engineering costs. The real-time applications that rely on COTS components can be directly benefited from this framework.

Abstract: The challenge of keeping the development and implementation of real-time control systems reliable and efficient and at the same time, low-cost and low-energy, is getting harder. This is because system designers and developers are faced with the dependability, inflexibility and often high-cost of specialized or custom-built hardware and software components. This research attempts to tackle issues such as the reliability and efficiency of real-time control systems and advance further the current state-of-the-art. For this purpose, a strong emphasis is placed on finding novel efficient solutions based on standardized and commercially available off-the-shelf hardware/software components. In this direction, this research applies credible and feasible methodologies (e.g., model-based design, component-based design, formal verification, real-time scheduling, prototyping, and validation) in an innovative enhanced way. As an important outcome, a versatile integrative design approach and architectural framework (VIDAF) is proposed, which supports the development and implementation of reliable real-time control systems and applications using commercial off-the-shelf (COTS) components. The feasibility and applicability of the proposed system's architecture are evaluated and validated through a system application in embedded real-time control in manufacturing. The research outcomes are expected to have a positive impact on emerging areas such as the Industrial Internet of Things (IIoT).

Keywords: control systems; real-time; general-purpose; commercial off-the-shelf; hardware/software components; reliability; availability; efficiency; low-cost

1. Introduction

The advances of computer technologies and the continuous growth of their applications bring new challenges (e.g., tasks concurrency in multicore processors) in building reliable real-time control systems. The complexity of such systems continues to increase, by integrating various computing and networking components and devices, particularly wireless communication equipment and other structures and services that rely on the physical world as well as on the virtual world e.g., cloud computing and the Internet-of-Things (IoT) [1,2]. The drastic increase in the versatility of computing

devices and their deployment in key sectors, such as health-care, energy, and the military raises several issues regarding the reliability, availability, and adaptability of their operation. Many sectors rely on the reliable, on-schedule, and cost-efficient operation of such computer-based systems in order to provide appropriate and time effective services. Such essential aspects are a necessity for critical systems deployed in key sectors. In these environments, the architectural framework of such systems is based primarily on specialized or custom-built hardware/software (HW/SW) components, which impose extra undesirable costs, slow down the development and maintenance process, and limit the reusability and versatility of the final product or service [3].

The challenge of keeping the development and implementation of real-time control systems reliable and efficient, and at the same time, low-cost is getting harder. This is because system designers and developers are faced with the dependability, inflexibility, and often high-cost of the specialized hardware/software components [4,5]. In addition, building systems based on specialized hardware/software imposes extra costs which are not desirable. Developers would prefer to utilize primarily general-purpose, reconfigurable/reprogrammable hardware/software components, in order to lower the cost, speed up the development process, and make the final product more versatile. Such components are commercial off-the-shelf (COTS) items sold in substantial quantities in the commercial marketplace. For example, commercial off-the-shelf components have become the preferred way of designers in military and aerospace applications for the development of unmanned aerial vehicles [6,7].

The market of reliable real-time systems is rapidly expanding due to the increasing number of applications in several domains and sectors (e.g., health services, industrial robotics, unmanned aviation, smart cities, etc.). In such critical real-time systems, typically, the more severe the consequence of potential system failure, the more specialized the control system needs to be. Usually, such specialized solutions impose significant costs. The new market applications, particularly of reliable embedded systems in critical applications in IoT and cloud computing, are seeking to use low-cost COTS-based components and devices in building real-time control systems due to their immediate availability, rapid utilization, ease of reconfiguration, and versatility. Low-cost COTS components are expected also to affect the reliability of many systems, as reduced costs could improve the threshold for introducing reliable solutions in real-time systems too, for which reliability is not a strict prerequisite (e.g., in soft real-time systems). In this direction, several companies report the development of such systems based on COTS components. Just recently, companies such as Abaco Systems, Real-Time Innovations (RTI), and Wind River Co. announced the joint development of the first hardware, operating system, and communications framework with commercial-off-the-shelf (COTS) components for Airborne Systems [8]. Systems built on COTS-based components provide increased reliability and quality over custom-built components as these are utilized and validated by various independent developers and organizations, due to their large scale utilization in various applications and implementations. However, despite the continuous increase in the utilization of COTS-based components, their reliable performance in real-time systems still remains under further research, which is an objective of this work too.

An important motivation for this research into reliable real-time systems based on COTS components, particularly for control applications in manufacturing, is the continuous increase of the demand for rapid, low-cost, low-consumption, and short-time development. Additionally, there is also a need for the immediate availability of versatile and general-purpose hardware/software components (e.g., cloud computing is the on-demand availability of computer system resources). For this reason, a strong emphasis of this research is to find efficient solutions to relevant open questions and issues, regarding the reliability and efficiency of real-time control systems based on general-purpose hardware/software components, such as:

- How to ensure availability and reliability through a warehouse of general-purpose hardware/software components?
- How to integrate and synthesize complex general-purpose hardware/software components into real-time systems applications?

- How to assess and measure the reliability of real-time systems applications based on general-purpose hardware/software components?
- How to evaluate the overall performance of real-time systems based on a versatile range of general-purpose components and devices?

This research work proposes a versatile integrative design approach and architectural framework (VIDAF) that addresses these challenging issues and open questions, with credible methodologies and approaches that lead to novel efficient solutions, validated in real-world cases. VIDAF defines an open architectural framework in building reliable, real-time, and cost-effective systems and applications, based on general-purpose hardware/software components, and minimizing the use of specialized ones. In resolving these challenges, the research targets the following objectives:

- Define a novel system architectural framework that uses general-purpose hardware and software components for building reliable real-time and cost-efficient control systems.
- Propose a novel system application infrastructure that enables efficient and reliable functionality and implementation of such systems in real-time applications.
- Demonstrate the efficient and reliable development and exploitation of such systems based on general-purpose hardware/software, in real-time embedded applications in manufacturing.

This research paper presents the approaches and methodology to achieve these objectives and goals in an innovative way, proposes an architectural framework that enables the development and implementation of reliable real-time control systems, and demonstrates its feasibility and applicability through a system application in manufacturing. A COTS-based development process is proposed, upon which a system architecture is implemented into the control of a lime mash/slurry (milk-of-lime) storage and delivery machine. In this case study, a custom-built (80C188EB-based and assembly driven) controller was substituted successfully with COTS-based hardware components (RaspberryPi-based microcontroller board, referred from now on as RPi3) and open-source control software modules (written in standard C and Python). The experimental results of the architectural working prototype provide evidence that the system performs reasonably well and correctly.

Overall, this research work presents a framework with its own structure specifications and constraints, upon which COTS-based systems could be developed, and further on, based on the architectural framework proposed, a case study demonstrates the development of such a COTS-based system in real-time control in manufacturing, verifying its validity and applicability. The proposed architectural framework is just a prototype architecture that could not be generally applied as a roadmap or as a standard in developing real-time COTS-based systems in the manufacturing domain. However, it constitutes a successful architectural attempt in developing COTS-based systems for specific manufacturing cases where real-time reliable control is also essential among low-cost, rapid development, and other desired requirements.

The paper is structured as follows: Section 2 describes previous related work; Section 3 describes the proposed architectural framework, design methodology, and structure; Section 4 presents the development process of such a COTS-based system application; Section 5 presents the implementation of such a system architecture in embedded real-time control of a lime mash/slurry delivery machine, and presents the experimental results; Section 6 provides a brief discussion of the research outcomes and system's overall performance, and; Section 7 draws conclusions.

2. Related Work

Real-time control systems integrate at large specialized and custom-build components and less open-source and general-purpose ones [9–11]. Due to hardware specialization and undisclosed software systems, many critical sectors, such as health care and industry, have to rely heavily on custom-built hardware/software, even from untrusted third parties, with undesirable delays and extra costs [12].

The use of standardized or general-purpose and commercially available off-the-shelf components and devices speed up the development, operation, and maintenance of such systems [13–15]. The cases of system applications, particularly in mobile devices, IoT, and cloud computing applications show a considerable increase, due to their immediate availability, low-energy consumption, and low-cost [16]. As the amount of commercially available off-the-shelf equipment in a system increases, it becomes easier and cheaper to modify the system, supporting greater adaptability to change and to adopt new features and capabilities. However, despite their continuous increase in utilization, their reliable performance in safety-critical systems still remains under further research. Issues remain with the reliability and security of distributed systems based on commercially available off-the-shelf components. e.g., the Internet of Things is such a massive network of primarily low-cost commercially off-the-shelf nodes components. The increase in their deployment creates further security risks and increases system vulnerability [17–19].

For critical systems, e.g., in military and industry, the architectures of the development frameworks are based primarily on specialized and custom-built components [20]. However, there is a tendency towards more versatile, flexible, and low-cost general-purpose hardware/software components [21,22]. The latest research reconfirms this, showing that real-time emerging applications need to be built upon more adaptive, immediately available, and reconfigurable/reprogrammable hardware/software components [23–26].

VIDAF focuses on the advancement of this research topic, through the demonstration of reliable methods and approaches that define the architectural framework for building reliable and cost-efficient real-time control systems and applications based on general-purpose hardware/software components, minimizing the use of custom-built solutions and products. Solutions have been proposed to support the development of similar architectural frameworks, particularly for embedded systems, but apart from a few [27–29], the majority is based on specialized mixed hardware and software components and approaches [30,31]. In the work of Ma et al. [27] a low-cost, reusable, reconfigurable platform is proposed that enables integrated design and implementation of embedded control systems, but in simulation only (Scilab). The work of Sarkar [28] proposes a modular approach for the design, implementation, and application of microcomputer-based embedded systems using open-source hardware and software platforms, but in small-scale applications. In the work of Bundalo et al. [29], a theoretical only concept is presented of optimizing commercial off-the-shelf (COTS) selection process using a prototype framework approach.

On the contrary, VIDAF aims at reliable development and operation of real-time control systems, and to deliver a novel open architectural framework for real-time control systems deployed in Industrial IoT (IIoT) and embedded systems. The outcomes of this research are expected to be worthy and have a positive impact on emerging areas such as IoT and embedded control systems, particularly in industry. Some of the novel outcomes include:

- Open system architectural framework that uses general-purpose hardware/software components in building reliable real-time and cost-efficient control systems.
- System infrastructure, working scenarios, and solutions for reliable control systems in real-time embedded applications in manufacturing.

The VIDAF is one of the few research works addressing altogether the reliability, reusability, and availability challenges for the integration of standard off-the-shelf hardware and software components into the development of an architectural framework for the exploitation of control systems for real-time applications. The delivered architectural framework is a development environment that combines formal verification techniques in simulation at an early prototyping stage and real-world validations. This promotes innovation by facilitating the development, exploration, and implementation testing of such systems at reduced time and cost. Further on with scientific advancements, VIDAF could advance the roadmap and the procedures towards the standardization in building cost-efficient real-time systems based on general-purpose hardware/software components.

3. VIDAF Architectural Framework

An important novel outcome of this research is the proposed open architectural framework and infrastructure (VIDAF) that facilitates the development and exploration of reliable control systems for real-time applications, based primarily on standardized/general-purpose hardware/software architectural components rather than specialized/custom-build components. The infrastructure of this architectural framework enables the integration and communication of hardware/software components in real-time, making applications to work together as one integrated system.

3.1. Essential Requirements

An initial requirements analysis determines the sets of requirements (e.g., power consumption, timing requirements, and latency constraints) that should be satisfied by a COTS-based system architecture in real-time control. Although the specific application impacts the requirements set for the COTS-based system, these primarily involve real-time constraints; that is, the system hardware and software components should be capable of meeting the requirements of the application task and deliver results with respect to a predefined time (deadline). Reliability and worst-case timing behaviors are the most important features for a real-time system, in particular for high criticality systems. Of course, these are not the only features required by a real-time system (other features required may be e.g., fault tolerance, etc.); but these are some of the most important ones. As a consequence, any hardware/software sources that cause undesired effects regarding the reliability and predictability (or determinism) must be addressed efficiently in order to integrate safely COTS hardware/software components into hard real-time systems. Additionally, the whole development process should be short and the development cost should be kept low.

3.2. Design Methodology

The VIDAF provides a COTS-based framework to develop and implement reliable real-time applications. One of the key features of the design approach followed is based on the use of open-source and general-purpose hardware/software components, due to their immediate availability, rapid utilization and implementation, low-cost and wide-range of applicability. The competitive advantages of this approach in cost and time savings by using standardized readily available hardware/software components are evident. General-purpose components are more versatile and expandable over custom-built components since their functionality could also be upgraded and extended via custom development.

The design methodology extends this approach in incorporating real-time and open-source scheduling techniques, applied in system applications in education and manufacturing. Aspects such as availability, reusability, reliability, and cost-efficiency are taken into consideration during the development, for the purposes and context of specific critical and non-critical system applications. The methodology addresses all together such issues and challenges that pose the integration of standard commercially off-the-shelf hardware and software components, into the development of an architectural framework for control systems in real-time applications. It investigates the optimal design of operational real-time scheduling techniques for systems deployed in critical applications (e.g., in manufacturing). Schedulability analysis techniques are applied to specify the real-time requirements for such systems applications deployed in real-time critical applications. The techniques and approaches within this methodology build upon recent [32,33] and previous [34,35] research on developing real-time embedded control systems using standardized components and equipment. In particular, a model-based and component-based design methodology [36,37] is applied in combination with real-time multithreaded scheduling techniques [38] in analyzing the needs and requirements for building reliable systems, including the real-time performance of hardware components and software modules in control systems applications. Previous research has shown that this methodology is feasible and credible in real-time control system applications with excellent results [39]. The delivered architectural framework combines

formal verification techniques in simulation at an early prototyping stage and real-world validations, which promote innovation by facilitating the development, exploration, and implementation of such systems at reduced time and cost.

Upon this methodology, further on, a COTS-based development process is presented, which takes into consideration specific requirements (such as the flexibility, reusability, reliability, and cost-efficiency) of hardware and software COTS components. For validation purposes, this is applied to the development of a control system architecture in a manufacturing case study. The experimental results, based on extensive experiments on the actual machine under control and documented with system-specific measurements (e.g., accuracy and timing metrics), provide evidence that the COTS-based control system performs reasonably well and correctly (compared to previous custom-build solutions). In addition, the adequate operation of the COTS-based control application demonstrates that the system infrastructure of this architectural framework enables the integration and communication of COTS hardware and software components in real-time, making applications work together as one integrated system. Overall, the outcomes reconfirm the applicability and validity of the proposed framework methodology and approach. Such experimental results could be worthy and have a positive impact on emerging areas such as Industrial IoT.

3.3. Framework Structure

The above framework design methodology is implemented into the following specific substructures: design specifications, software-hardware co-development, and system implementation. The overall structure of the proposed framework is illustrated in Figure 1.

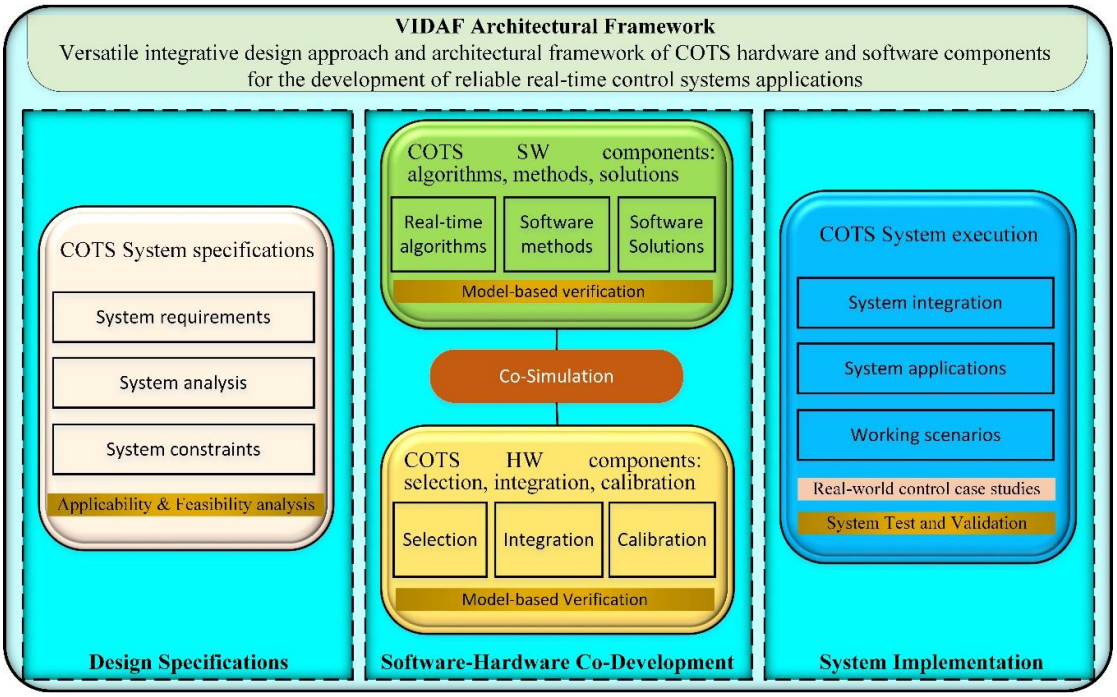

Figure 1. Versatile integrative design approach and architectural framework (VIDAF).

3.3.1. Design Specifications

This stage considers the requirements and constraints of COTS-based systems components regarding some of the important issues, such as availability and reliability (e.g., by using fault tolerance techniques), analyzes their applicability and feasibility (applicability and feasibility analysis), and defines the system specifications required in reliable development of COTS-based systems that use general-purpose hardware/software components.

3.3.2. Software-Hardware Co-Development

The development considers the techniques (e.g., real-time scheduling algorithms), methods (e.g., reliability and schedulability analysis), and system performance specifications and constraints necessary to develop COTS-based system architectures, software structures and solutions that guarantee requirements on availability, reliability, and timing in building reliable systems deployed in real-time control applications by integrating effectively general-purpose hardware and software components. Modeling is important to evaluate and ensure that the COTS-based system meets the design specifications. Co-simulation involves model-based development and verification of the delivered software infrastructure and hardware system, based on real-time scheduling experiments in simulation.

The development process runs in parallel to minimize the development time and costs, with the software development focusing on software structures and features and the hardware development including component selection, integration, and calibration. Despite this, the hardware and software developmental flow influences each other; e.g., software requirements may alter hardware components, and the hardware components architecture may influence software modules' design and development.

3.3.3. System Implementation

Implementation involves the integration and application of the developed COTS-based system in real-world control case studies for testing, validation, and exploration purposes. The overall system performance could be evaluated and improvements implemented.

3.4. System Infrastructure Architecture

The system architecture refers to the hardware and software components, their relationships, attributes, and functions that constitute a COTS-based system application. A component-based approach is utilized in the design of a system application architecture within the VIDAF. Individual functional structures (COTS nodes) represent well-defined objects containing hardware/software components, tasks, and properties.

A VIDAF system is built using COTS nodes, as illustrated in Figure 2.

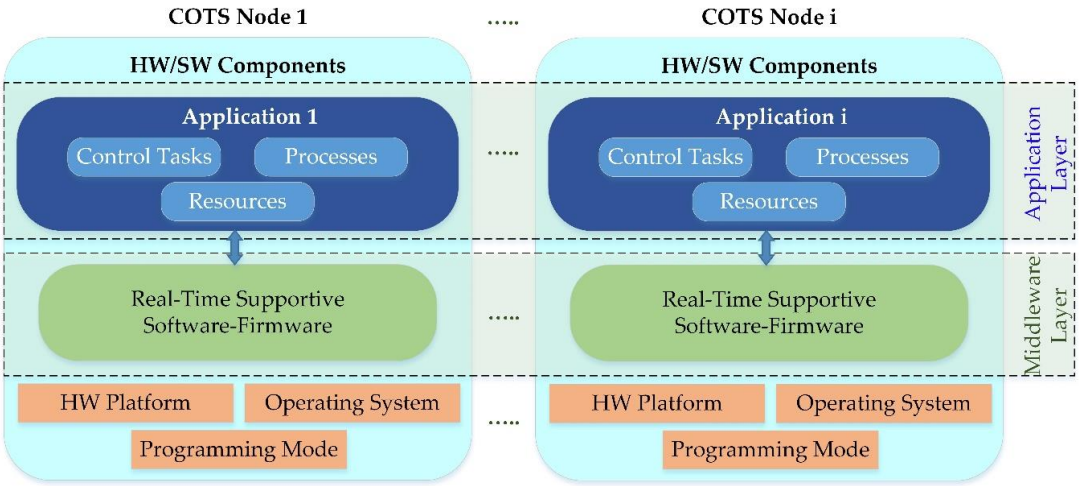

Figure 2. VIDAF System Architecture.

Each COTS node is constituted of various hardware/software components (e.g., from a standardized object's repository warehouse), within which applications could be executed. Each node may have its own hardware platform, operating system, and programming model. This ensures the diversity in COTS components and applications and different implementation and execution environments in each node. Within the system's infrastructure, a run-time middleware layer between the application and

the COTS nodes provides the support software needed for real-time application development. Such a software layer utilized in many cases is the Linux firmware, enhanced with real-time capabilities, with the real-time preemptive patch PREEMPT_RT [40]. One of the design specifications and essential requirements that should be satisfied by a COTS-based system architecture in real-time control is to ensure reliability and guarantee that worst-case behavior is kept within acceptable time limits (bounded by response time and latency). For this reason, in our implementation we configured a real-time variant of Linux with PREEMPT_RT patch, capable of minimizing both operating system overheads and latency. The resources that are crucial for the execution of real-time applications are directly assigned to the applications themselves, with no software layer in the middle.

Applications consist of tasks (in our implementations they are actually threads, each one having its own real-time scheduling policy SCHED_FIFO/SCHED_RR and high priority) and resources (hardware/software objects) within a single or several COTS nodes. Tasks communicate (through shared distributed memory) and interact in a preemptive multitasking way to perform reliably a real-time job. In our approach, tasks are enforced to predictable memory accesses (mlockall) that are current and future task's memory allocations are locked, in order to prevent being paged out. In our implementations, this feature is invoked immediately from the main master module to prevent the page out of memory for the real-time tasks. This results in lower latencies, a higher number of execution iterations, and improves the determinism of RPi3's real-time applications' execution. This application infrastructure ensures that a real-time task during its execution reacts appropriately in time to events that take place in the real world and that the overall system response time is within the desired real-time specifications limits.

Tasks from the same application can also be allocated to different nodes. Tasks could be related and even process the same data across the different nodes (application layer). However, in our implementation, this is not the preferred mode of operation to avoid bus interference (for memory data) or other potential conflicts between tasks running on different nodes. Additionally, in the real world, there may exist several software/hardware layers between the actual device under control and the real-time tasks, running other tasks or processes too, altogether competing for hardware/software resources that may lead to a non-deterministic execution. The functionality and operation of each node could also be changed or reconfigured at run-time, according to the application mode. Therefore, this component-based design approach does not impose restrictions on how applications are structured.

The implementation mechanism of each system node lies within the structure of each system hardware/software component.

4. Development

4.1. Traditional Development

Traditionally, the development process of a control system consists of two main stages: system design and its implementation. These are shown in Figure 3, where each step is presented within the so-called V-model.

In the first stage, a requirements analysis is performed to determine the important design specification parameters and ensure their inclusion in the design process. According to the requirements, the initial specifications are generated, upon which the design proceeds into the synthesis of the control system architecture. This step involves the design of control structures and algorithms and the design or determination of the hardware components necessary to support the development. Modeling plays a vital role in the description of these software structures, hardware components, functionality, and relationships. Further on, a system model that represents the entire system under development is constructed and verified in simulation, to ensure that the simulation model reflects all the system requirements and specifications. At the next stage, a system prototype is constructed and materialized as a working prototype, through a systems engineering process that may involve a variety of technologies and disciplines. This system prototype is trialed with extensive testing and

experimentation with real case data to assess that its performance meets the design objectives. Finally, system behavior and results are compared and assessed against those of any existing similar systems; therefore, validation provides a final assurance that the designed system fulfills as much as possible the design specifications and is ready to be applied.

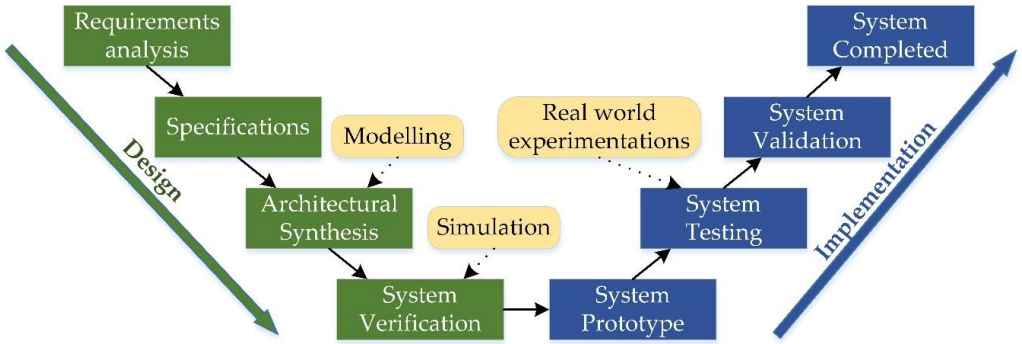

Figure 3. Traditional development process.

4.2. COTS-based System Development

The COTS-based system development methodology we have adopted is shown in Figure 4.

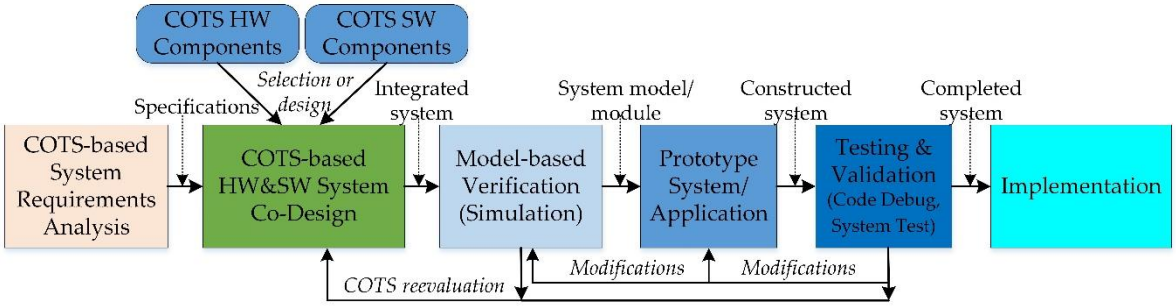

Figure 4. Commercial off-the-shelf (COTS)-based system development process.

At first, COTS-based requirements analysis is performed that determines the important design specification parameters that correspond to the needs of the application environment. According to the COTS-based specifications generated, the development proceeds into the synthesis of the COTS-based system architecture. At this stage, optimal integration of the hardware (HW) and software (SW) COTS components is very important. Therefore, appropriate selection of existing COTS HW components (e.g., commercially available low-cost development boards, open-source hardware, etc.) and SW components (e.g., available open-source software and real-time operating systems), or the design and development of a new one (e.g., real-time multithreaded control software, as it is the case in our implementations), is a necessity. At this step, we must primarily consider those hardware/software components that most fulfill the real-time specifications and requirements, irrespective of the fact that their real-time features could be further improved during the development. For instance, in embedded systems for real-time control, an advisable and wise choice would be for a system based on a COTS processor (such as ARM), instead of other specific CPUs, micro-controllers, or Digital Signal Processors (DSPs). Many hardware vendors develop system-on-chip (SoC) devices, which usually include on the same integrated circuit one or more COTS processors, together with other specialized peripherals. In our implementation, we make use of such a device (e.g., the RPi3 which integrates an ARMv8 Cortex-A53 quad-core processor unit within a Broadcom BCM2837 SoC), since it is cheaper, more reliable, and consumes less power than other equivalent systems. The simultaneous co-design of system hardware and software components and their integrations keeps low the development

time and cost. That will ensure achieving optimal performance for the entire system and satisfy all the requirements.

Model-based techniques play an important role in systems engineering for verification purposes and diagnosis. Model-based reasoning is the symbolic processing of an explicit representation of the internal workings of a system, in order to describe and examine (simulate) its behavior from its structural components. Many different techniques and tools have been developed to support such modeling tasks in systems design and development. In our case studies, we adopt a model-based methodology for systems modeling using a qualitative modeling and simulation tool (QMTOOL) [41] that utilizes both conventional numerical methods and more advanced qualitative techniques to deal efficiently with the description of control systems, mechanisms, and processes. Qualitative models are introduced at a high-level abstraction form, using a relatively small amount of information, similar to human reasoning in studying a complex system's behavior. Thus, by using this model-based checking approach, the performance of the system model is verified through extensive simulations against the predefined requirements and specifications. For any design errors found, appropriate modification requests are made (COTS components reevaluation) and the system model is rebuilt and verified again. One of the major advantages of this approach in developing control systems applications in manufacturing is the increased flexibility provided in the description of system parameters, components, and their functional relationships in symbolic qualitative terms and forms, easily understood by system engineers and non-specialists in the industry.

Further on, based on a credible and well balanced integrated system model, a prototype system is physically constructed. The realization process may involve the assembly of specified COTS HW and SW components as well as the design and development of new hardware/software components if required. This prototype system will be tested, possibly many times until its credible performance is trusted and accepted, and finally, its implementation validated, prior to its wide-scale production. At this stage, any structural and functional errors found may lead to simple model or prototype modifications, or even to more complex, time-consuming and costly modifications in the system's design.

5. Case Study: COTS-based Embedded Real-Time Control of a Lime Mash Delivery Machine

Applying the research results on a real system is often a challenge, due to different working conditions than those of an academic environment, and the considerable efforts needed in various engineering aspects (implementation, operation, validation) in order to be successful.

The feasibility and applicability of the proposed architectural framework to handle the development and implementation of a COTS-based real-time control system application in manufacturing is demonstrated with a case study in the industry: the control of a lime mash/slurry (milk-of-lime) storage and delivery machine (ADAMs Machines Constructions Company, www.adam.com.gr). Lime mash/slurry is hydrated slaked lime that is quicklime reacted with water to form calcium hydroxide. The machine under control (Figure 5) is a system that stores and delivers lime mash volumes with accuracy at the appropriate speed. The lime mash storage tank is a cylindrical silo (10 m^3, ~3 tons) equipped with an agitation (mixing) and scrapping system for delivery of lime mash. The machine can be operated in a manual mode (using the joystick panel) or an automatic mode (based on a custom-build 80C188EB controller) by setting the value of the required lime slurry volume to be dispensed.

The operation of the machine is based on a hydraulic pump system driven by electro-hydraulic valves (Duplomatic, Pmax 320Bar, 24VDC) and a speed regulative three-phase asynchronous motor (11HP), which in turn drives a gearbox (motor) mechanism that regulates the speed (rotation) of the single-shaft agitation system (paddles shaft) that regulates the actual flow of lime slurry out of the silo (tank). The proportional control valves can adjust the flow rate from 0 to 1500 L/min according to the input voltage (0–10 V) by changing the spool position. In addition to the valve control by the custom-build controller, the valves can be opened and closed for the delivery of lime slurry by the joystick panel. The actual flow volume of lime slurry is measured with an electromagnetic flowmeter

(Danfoss Magflo MAG3500/MAG6000) installed in the tank's lime mash outlet. A custom-built controller (based on 80C188EB microprocessor) developed within the company, monitors and regulates the delivered lime slurry volume according to preset values.

Figure 5. Lime mash/slurry storage and delivery machine under control.

5.1. The System under Control

The gearbox system driven by the motor is the actual mechanism that rotates the single-shaft agitation system. The shaft's rotation generates a relative velocity that regulates the lime flow. The main control parameter that regulates the quantity of the lime slurry flow is the reference angular speed (ω) of the single gear shaft agitation system (paddles shaft), which forces the lime volume to flow out of the silo. The outlet volumetric flow rate Q_v of the lime slurry is proportional to the angular (rotational) speed $\dot{p}_\omega$ of the gear shaft according to the following simplified equation:

$$Q_v = K\dot{p}_\omega \quad (1)$$

where K is the lime constant, which depends on the viscosity and velocity of the lime.

A schematic diagram of the lime tank controller is shown in Figure 6.

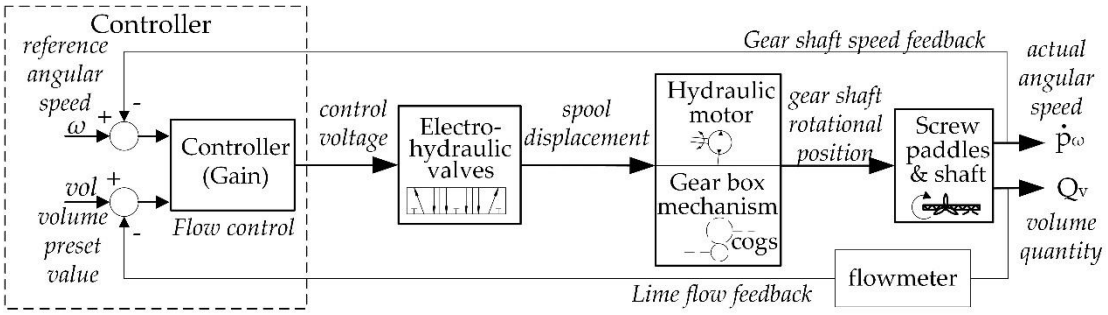

Figure 6. Schematic diagram of the lime tank controller.

The flow meter monitors the flow of lime slurry and this data is used by the controller to automatically adjust the speed of the single gear shaft agitation system to maintain the accurate flow rate. The custom-design of the machine's controller was based upon 80C188EB Intel microprocessor (25 MHz, 20bits address bus, 1M address space, 8 bits data bus, 16 I/O pins, 3 timers/counters, 5V). The single-board controller basically contains four 62,256 (32K × 8) RAMS that compose a 128K × 8 bit memory, two 27C512 (64K × 8) EPROMs that compose a 128K × 8-bit memory, two RS232 (PC) communication ports and a programmable peripheral interface circuit that directly connects to the actuators' control equipment (electro-valves and motor relays). The control software was written in assembly language (80 × 86 family MASM Assembler). A simplified block diagram of the 80C188EB microprocessor-based custom-built controller is shown in Figure 7.

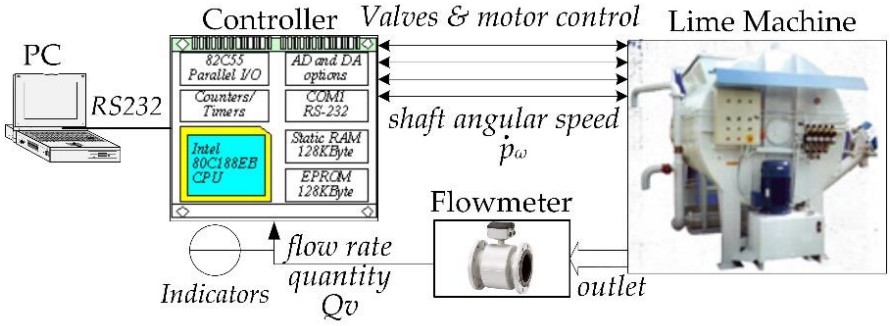

Figure 7. Lime mash/slurry machine custom-build control system.

5.2. The COTS-Based Controller

The design and implementation flow of the control system development follows the described COTS-based development process and involves the initial specification of COTS hardware and software modules and units (COTS node), their applicability analysis, their modeling and simulation-based verification, the actual programming and implementation steps, and finally, real-time experimentations of the applied control system for performance validation purposes.

The objective was to replace the custom-built controller by a COTS-based, easily customizable, reusable, low-cost, low-power, and open-source embedded control system. For this purpose, a Raspberry Pi3 Model B microcontroller board was chosen as the development platform (RPi3 microcontroller integrates an ARMv8 Cortex-A53 quad-core processor unit within a Broadcom BCM2837 SoC [42]). The choice of RPi3 is based on the fact that it has been extensively used for embedded low-powered control applications due to its computing power andcomparative price, its ease of use with peripheral equipment, and low-energy consumption [43]. In addition, research shows that such a system is capable of supporting adequate real-time timing for most measurement purposes [44]. The system runs UBUNTU 18.04 LTS Linux patched with PREEMPT_RT for real-time processing. This architectural upgrade brings several advantages regarding reconfigurability, versatility, availability, and cost. All the specific operation constraints that apply in operating the lime machine under control, are handled by this COTS component, the RPi3.

5.2.1. COTS-based Control Architecture

The COTS-based control architecture is shown in Figure 8. This consisted of: (i) the RPi3 COTS component, (ii) an RS232 serial module component, (iii) the lime slurry flowmeter, and (iv) the 5V solid state relay module and triac circuit for the control of the AC motor drive frequency regulator.

The overall monitoring and control are performed on a central control board, the setting of which is based on an RPi3 panel that monitors the machine's operation. The RPi3 controller is connected to a Danfoss Magflo MAG6000 electromagnetic flowmeter, through an RS232 serial module (MODBUS RTU), in order to receive the lime flow rate volume (as pulses count). Accordingly, it regulates with PWM (Pulse Width Modulation) the speed of the variable frequency ac motor, through a specific triac circuit (integrated into the motor unit), in order to set the required volume. The RPi3 controller operates the status (on/off) of the machine under control (motor's run/stop status) through a 5V relay module (with high-level switching voltage 380V) that enables/disables (1/0 signals) the motor's relay.

The lime mixing tank is fitted with an agitation system and actuators to continuously maintain the lime slurry in suspension prior to its delivery. The operation of the machine is initiated and terminated by on/off (1/0) signals from the RPi3 (through 1-channel 5V relay module with high-level switching voltage 380V) that sets the motor's run/stop status. This could also be accomplished manually through the on/off button provided (for practical and security reasons too). The speed of this AC motor, with variable frequency driver/regulator (VFD) is automatically controlled (increase/decrease) by the RPi3 with PWM (Pulse Width Modulation) signals (through the triac circuit) to a set value, by using as a

comparative input the corresponding flow rate volume (vol), which is proportional to the velocity of the flow, from the feedback flow control loop (flowmeter volume pulse). The RPi3 receives this flow rate volume through an RS232 unit connected (MODBUS RTU) to an electromechanical counter within the flowmeter, configured as a volume pulse output unit. Then, the RPi3 controller will reduce/increase the speed (ω) (actually voltage frequency) of the VFD AC motor until the desired volume is registered by the flowmeter. Thus, flow control is modulated accordingly to the flow rate of lime slurry out of the tank. This could also be accomplished manually through the feed buttons (+,−), or by using the up and down keyboard arrows keys, or from the GUI (Graphical User Interface) of a specific application developed for this purpose (running and displaying at a PiTFT touch screen). On the spot, on-line commissioning and configuring is performed by using auxiliary peripherals, a PiTFT Plus 480 × 320 3.5″ TFT display with touchscreen, and a 2.4G mini wireless keyboard with touchpad. In addition, the RPi3-based controller has the ability to wirelessly connect to other mobile devices (e.g., Android or iOS iPhone) for monitoring or configuring purposes using standard connection protocols (Wi-Fi, Bluetooth).

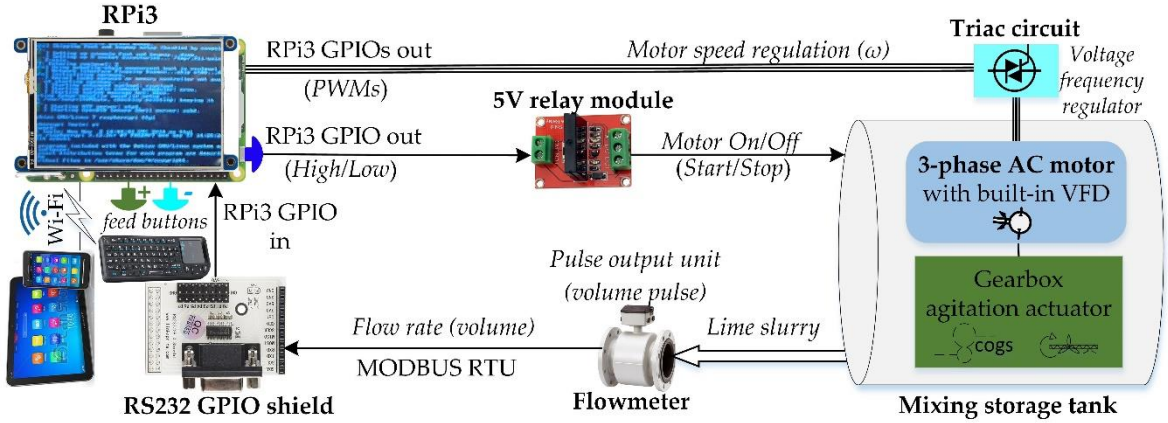

Figure 8. COTS-based control system.

5.2.2. COTS-Based Control Software

Within this COTS's hardware infrastructure, the real-time supportive software (middleware layer) is based upon a customized version of the open-source UBUNTU 18.04 LTS Linux distribution available for embedded IoT devices and boards such as RPis, PDAs etc. UBUNTU 18.04.2 LTS (kernel version 4.15.0.1041-raspi2 #44 Ubuntu SMP PREEMPT armv7l) supports real-time interface circuitry with peripherals (e.g., sensors, actuators) and system applications written in standard high-level languages such as C/C++. The Linux kernel is a low-latency preemptible kernel by default, capable of satisfying soft real-time requirements of control applications. Thus, the default scheduler cannot provide the fixed and predictable latency required e.g., for real-time data sampling. For this reason, PREEMPT_RT real-time patch was installed, to provide Linux OS with deterministic scheduling and hard real-time capabilities of a fully preemptible kernel. This ensures reliable real-time performance in the controls of electronic devices and other control gears and sensors, which have more strict timing requirements in the transmission of signals. In many cases, the PREEMPT_RT patched Linux is a suitable alternative for proprietary real-time operating systems.

The system's application control software (developed in C) consisted of tasks (implemented as software threads) that perform all the processing and machine control operations (including motor on/off, motor speed regulation). A master software module generates the PWM signals for the control of the AC motor, thus regulating the flow rate of lime slurry delivered. The RPi3 supports a PWM hardware peripheral (through general-purpose onput/output GPIO18) but is also capable of generating more software PWM signals needed for the frequency control of the AC motor. In our approach, a

single process is created that handles three software threads (tasked as user-level threads, based on pthread library) to generate multiple PWM signals.

At least one processor core is dedicated (CPU affinity is set for each task) to run each multithreaded task that generates the PWM signal, in order to ensure its timing execution. For this purpose, any task attempts to execute non-preemptible (real-time) code in other cores are not allowed. In this way, tasks running on these cores perform predictably with minimum latencies. The majority of latencies on the RPi3 with real-time support OS are considerably lower and below 50usecs. The average maximum observed latency is still significantly lower than the one observed under the default Linux (without real-time support). Even in the worst-case scenario, the results indicate that a value of about 150usecs, as a lower bound, could be an acceptable safety margin for such low frequencies in most real-time systems structured in this way, as long as the frequency-time step value is higher. However, an embedded systems developer should consider any application latency specific requirement, and not rely on this indicative latency bound. Overall, the latencies and particularly the maximums are reduced in the RPi3 with real-time support, thus proving that is suitable for use in time-sensitive (real-time) embedded control systems.

The flow of the system control software routines is shown in Figure 9.

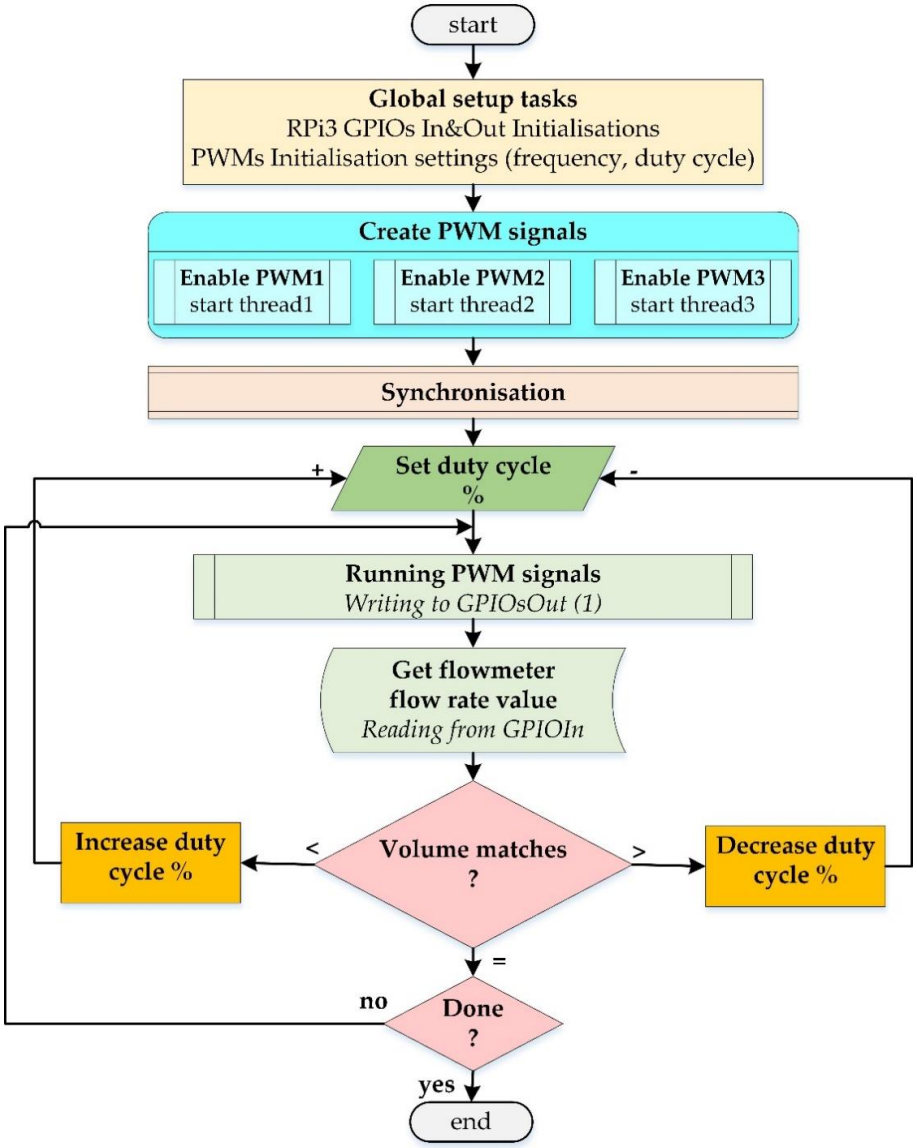

Figure 9. The flowchart of the system control software.

PWM involves the modulation of the signal's pulse. Initially, after experimentation, the operating PWM frequency (period of the cycle) was set at a high-frequency rate of 1000 Hz. The percentage of time the signal is ON of the total cycle (duty cycle) spans from a minimum (0%) to a maximum value (100%), and initially was set to 100% (no modulation). While running the control software, this ratio (ON_time/100) will be changing (increasing/decreasing), thus changing the speed (frequency) of the motor's rotation according to the desired flow rate value of the lime slurry delivered volume. Next in the software flow, three software threads are created and synchronized (joined) within their parent process. Further on, these threads are enabled to generate the PWM signals (writing 1s to GPIOs outputs) with the given PWM frequency and duty cycle. At the same time, the flowmeter's flow rate reading is checked against the desired volume, and accordingly, the duty cycle is changed (increased/decreased). This process runs repeatedly in a loop until it is done. The algorithm that describes the basic functionality of these system control structures is shown as pseudocode in Algorithm 1:

Algorithm 1: Master module control structures functionality

```
structure {gpio_output [3]; freqeuncy; dutycycle_ratio, threads[3]} pwm; ← PWMs structure
function PWM_initiate
   pwm = malloc(sizeof *pwm); ← allocate memory
   pwm->gpio_output[i] = gpio; ← enable GPIO outputs
   pwm->frequency = frequency; ← set frequency (initially set to 1000Hz)
   pwm->dutycycle_ratio = 1; ← set duty cycle (initially set to 100%—no modulation)
   return pwm;
end function PWM_initiate
function PWM_thread
   pthread_setschedparam = SCHED__RR, 99 ← set real-time scheduling policy and high priority
   pthread_setaffinity = 1:3 ← set each thread to run on a separate CPU core (actually cores 2 to 4)
   while (!done)
         period = 1.0/pwm->frequency; ← define run period
         ratio = pwm->dutycycle_ratio; ← define ratio
         time_ON = period * ratio; ← define time run
         for threads_count = 1:3 starts PWMs signals execution/generation
             gpio_write (pwm->gpio_output[1:3],1); ← logical bit 1; time_wait(time_ON);
             gpio_write(pwm->gpio_output[1:3],0); ← logical bit 0; time_wait(period-time_ON);
         end for
   end while
end function PWM_thread
function PWM_master
   mlockall(MCL_CURRENT | MCL_FUTURE); ← lock current & future task's memory allocations
   sched_setscheduler = SCHED_FIFO; ← set process real-time scheduling policy
   PWM_initiate; ← PWMs structures creation and initialisation
   gpio_init; ← GPIOs input & output initialisations
   for threads_count =1:3 ← PWMs signals creation
         pthread_create(pwm->thread[1:3],PWM_thread,pwm); ← threads creation and execution
         pthread_join(pwm->thread[1:3], NULL); ← threads synchronisation
   end for
   gpio_read (volume, pwm->gpio_input); ← get flowrate volume
    if (volume<setvalue) pwm->dutycycle_ratio++; ← increase flowrate
   else if (volume>setvalue) pwm->dutycycle_ratio–; ← decrease flowrate
             else if (done) { free(pwm); exit();}
end function PWM_master
```

5.3. Experimental Results

A simple graphic application (GUI), shown in Figure 10, was also created by using the Tkinter module in Python, to ease this experimentation process by providing a user-friendly interaction with the control system and for future remote control purposes.

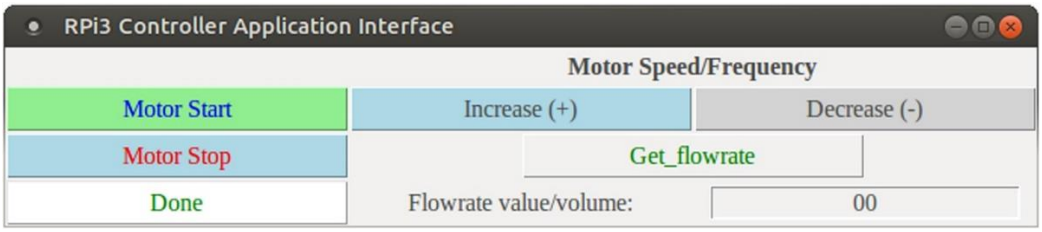

Figure 10. COTS-based controller simple application interface.

This Python GUI module uses ctypes foreign function library to communicate with the RPi3 controller by calling appropriate C functions of the control software in C. For this purpose, these C functions are defined within a shared library generated as a shared object that the Python GUI module calls to execute. The simplicity of its use allows any user to perform easily the control operations.

Experiments were carried out with the actual lime mash/slurry storage and delivery machine. Overall, the performance results provided evidence that the proposed control system performs correctly and can efficiently and effectively manage the machine's operations. In particular, the performance of the COTS-based control system (RPi3 controller) was evaluated under real-time operating conditions. Certain tests and measurements were performed and the results were compared against those obtained from the custom-built controller (80C188EB controller). During the testing, internal data regarding the flow quantity (volume) measured by each of the controllers, were collected and examined for their accuracy, based on the flowmeter's nominal values for the corresponding volume load. During all the experiments, the operating conditions (e.g., tank internal pressure, total lime mash quantity, temperature, etc.) were kept constant and the same for both controllers. In both cases, the internal diameter of the flowmeter's metering tube was 5 inches (sensor dimension of DN 125), and the frequency was set to 1 pulse/unit (e.g., m^3). The results obtained are shown in Table 1.

Table 1. Experimental results of volume measurements.

Min Volume Per Pulse ($m^{3)}$	Pulse Width (ms)	Time Run (min)	Nominal Volume ($m^{3)}$	80C188EB Controller (m^3)	RPi3 Controller (m^3)	Accuracy (±0.5%)
0.0011	5	1	13.2	12.5	13	0.066
0.0022	10	5	66	64	65	0.33
0.011	50	10	132	130.5	131.5	0.66
0.022	100	30	396	394	393.5	1.98
0.111	500	60	792	790.5	789	3.96

The measurements were performed at a variable pulse width (the flowmeter's measuring signal) and time of execution (time of operation), recording the response of each controller in measuring the flow quantity (volume in m^3). The volume measurements of each controller are quite close to each other, and to the nominal value. These are within the flow rate range, a minimum (13.2 m^3/min) and maximum volume (792 m^3/h). The COTS-based controller tends to be more accurate for shorter running times and pulse widths. Taking into consideration that the flowmeter's measuring error (accuracy) is ±0.5% of the actual flow rate, it is evident that both controllers' results are in general within this range. The obtained results suggest that our COTS-based design approach is competitive with that of the custom-build controller, at a shorter development time and a lower cost.

6. Discussion

The proposed framework and design methodology primarily aim to present the development flow and determine the key architectural attributes and hardware/software components in advance of the implementation of particular COTS-based system architecture. This architectural prototype is based on methods and approaches, which do not necessarily involve extensive quantitative descriptions. The results produced from the implementation of this architectural framework in a case study in manufacturing, verify and validate the applicability of this specific structural framework and methodological flow, particularly from a software and hardware co-design point of view. This is documented with system-specific measurements (e.g., accuracy and timing metrics), which are examined and compared against the previous implementation, as indicated in Table 1, and other approaches examined earlier in related work.

The goal of the case study system application was to verify the validity and feasibility of the research concept and methodology in applying the proposed open architectural framework and infrastructure into the development and exploration of reliable real-time control systems, based primarily on standardized/general-purpose hardware/software architectural components.

The case study system application infrastructure proves the efficient integration and communication of COTS hardware and software components in real-time control, making applications work together as one integrated system. The COTS-based control system developed is shown to behave according to the design criteria and essential requirements. The RPi3 system controller is capable of achieving accurate flow rate control within an optimal working range. The most pronounced performance improvement, compared to the previous controller, is the advanced but at the same time friendly, way of reliable control.

The proposed framework architecture, design, and development methodology take into consideration specific requirements regarding the availability, reusability, reliability, and cost efficiency of hardware and software COTS components. As an outcome of this case study, a summary of such notable architectural features of COTS-based systems versus specialized or custom-built, and their qualitative metrics, is provided in Table 2.

Table 2. Architectural features of COTS-based systems versus specialized or custom-built.

Feature	Specialized System	COTS-Based System
Flexibility	Specific functionality, low adaptability to change	Ease of modifications or upgrades of commercial HW/SW components
Redundancy	Highly fault-tolerant architectures, low probability of unsafe failures	Relevant critical components and functions are built-in
Integrity	Specific interconnections with other components	Ease of integration with other COTS components
Reliability	Often highly assured	Additional requirements needed to assure reliability
Reusability	Low versatility, specific low-level assembly code	Software and high-level code reusability
Time	Higher development time	Lower development time
Cost	Considerable variable costs	Lower development cost and system life cycle cost

Furthermore, the choice of open-source and general-purpose components, both in respect to COTS hardware (e.g., RPi3) and software (e.g., use of open-source Linux OS patched with PREEMPT_RT, and standard C and Python programming), broadens the capabilities of the control system. In this direction, a future objective is to substitute the expensive flowmeter with a simpler low-cost general-purpose fluid meter.

7. Conclusions

This paper presents an open architectural framework and infrastructure for reliable and low-cost real-time control systems and applications, based primarily on standardized/general-purpose (COTS) hardware/software architectural components, and its implementation. The system infrastructure of this architectural framework enables the integration and communication of hardware/software components

in real-time, making applications work together as one integrated system, as demonstrated with a case study in manufacturing.

The applicability and validity of the proposed framework are tested with the implementation of a COTS-based control system, which utilizes a Raspberry Pi3 Model B microcontroller board running a Linux-based distribution (UBUNTU 18.04.2 LTS) patched with PREEMPT_RT for real-time processing. The system control software developed in C performs all the processing, communication, and control operations of a specific lime mash/slurry storage and delivery machine used in the case study experimentations. Software applications written in C and Python were developed for operation and performance testing purposes. The validity of the proposed system was evaluated through extensive experiments on the actual machine. The experimental results of the architectural working prototype provide evidence that the system performs correctly. Some of the key features of the implemented COTS-based control system can be summarized in the following:

- Open-source, low-cost, low-power, COTS-based control architecture, easily reprogrammed and reconfigured for integration with other automation systems (e.g., electromagnetic flowmeter).
- Linux-based supportive software with real-time capabilities due to the real-time patch PREEMPT_RT in UBUNTU, capable of supporting future advanced timing control requirements (e.g., actual flowrate measurements).
- Real-time control software, based on multithreaded modules in C, ensure reliable operation of the machine controller and provide reconfiguration flexibility of the embedded device.

The proposed architectural framework and control system infrastructure could be used for a variety of control purposes and needs in manufacturing, as well as for experimental research in control of other control devices and machinery. Future research is already undertaken (https://github.com/gadam2018/RPi3-PREEMPT_RT) into the advanced real-time requirements in professional and practical issues regarding the reliable and low-cost real-time control of large-scale networked production machines.

Author Contributions: Conceptualization, G.K.A.; Methodology, G.K.A. and G.G.; Software, G.K.A. and G.G.; Validation, N.P. and T.S.; Investigation, G.K.A.; Data curation, G.K.A.; Writing—original draft preparation, G.K.A.; Writing—review and editing, G.G. and N.P.; Visualization, T.S.; Supervision, G.K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors wants to acknowledge the ADAMs Machines Constructions Company for the technical support and the provision of the machinery and materials used for experiments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xia, Y. Cloud Control Systems. *IEEE/CAA J. Autom. Sin.* **2015**, *2*, 134–142.
2. Buyya, R.; Dastjerdi, A. *Internet of Things, Principles and Paradigms*, 1st ed.; Morgan Kaufmann: Cambridge, MA, USA, 2016.
3. Wolf, M. *Computers as Components, Principles of Embedded Computing System Design*, 3rd ed.; Morgan Kaufmann: Cambridge, MA, USA, 2012.
4. Gan, J. Tradeoff Analysis for Dependable Real-Time Embedded Systems during the Early Design Phases. Ph.D. Thesis, Technical University of Denmark, Copenhagen, Denmark, 2014.
5. Pan, W.; Li, Z.; Zhang, Y.; Weng, C. The New Hardware Development Trend and the Challenges in Data Management and Analysis. *Data Sci. Eng.* **2018**, *3*, 263–276. [CrossRef]
6. Berra, E.; Gaulton, R.; Barr, S. Commercial Off-the-Shelf Digital Cameras on Unmanned Aerial Vehicles for Multitemporal Monitoring of Vegetation Reflectance. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 1–9. [CrossRef]
7. Chiabrando, F.; Teppati Losè, L. Performance evaluation of COTS UAV for architectural heritage documentation. A test on S. Giuliano Chapel in Savigliano (CN) Italy. In Proceedings of the International Conference on Unmanned Aerial Vehicles in Geomatics, Bonn, Germany, 4–7 September 2017.

8. Real-Time Innovations Co. COTS DO-254 and DO-178C Solution for Airborne Systems. Available online: www.rti.com/news/abaco-rti-and-wind-river-partner-to-deliver-first-cots-do-254-and-do-178c-solution-for-airborne-systems (accessed on 4 April 2020).
9. Angelov, C.; Ke, X.; Sierszecki, K. A Component-Based Framework for Distributed Control Systems. In Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, Cavtat, Dubrovnik, Croatia, 29 August–1 September 2006; pp. 20–27.
10. Garcia-Valls, M.; Estevez-Ayres, I.; Basanta, P.; Delgado-Kloos, C. CoSeRT: A Framework for Composing Service-Based Real-Time Applications. In *Lecture Notes in Computer Science*; Bussler, C.J., Haller, A., Eds.; Springer: Berlin, Germany, 2006.
11. Zhang, P. *Advanced Industrial Control Technology*, 1st ed.; Elsevier: Amsterdam, The Netherlands, 2010. [CrossRef]
12. Raavi, J.K. Usage of Third Party Components in Heterogeneous Systems: An Empirical Study. Master's Thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2016.
13. Alford, L.D. The problem with aviation COTS. *IEEE Aerosp. Electron. Syst. Mag.* **2001**, *16*, 33–37. [CrossRef]
14. Finnegan, R. Requirements engineering methodologies for COTS systems. In Proceedings of the IEEE International Conference on Electro/Information Technology, Ames, IA, USA, 18–20 May 2008; pp. 475–481. [CrossRef]
15. Tarawneh, F.; Baharom, F.; Yahaya, J.H.; Ahmad, F. Evaluation and selection COTS software process: The state of the art. *Int. J. New Comput. Archit. Their Appl.* **2011**, *1*, 344–357.
16. Javed, F.; Afzal, M.K.; Sharif, M.; Kim, B. IoT Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2062–2100. [CrossRef]
17. Boniol, F.; Casse, H.; Noulard, E.; Pagetti, C. Deterministic execution model on COTS hardware. In Proceedings of the 25th International Conference on Architecture of Computing Systems, Munich, Germany, 28 February–2 March 2012; pp. 98–110.
18. Breivold, H.P.; Sandstrm, K. Internet of things for industrial automation—Challenges and technical solutions. In Proceedings of the IEEE International Conference on Data Science and Data Intensive Systems, Sydney, Australia, 11–13 December 2015; pp. 532–539. [CrossRef]
19. Mu, J.; Huang, D.; Ma, L.; Cao, Y. Safety Analysis Method for COTS Software Components in Train Control System. In Proceedings of the International Conference on Artificial Intelligence and Engineering Applications, Hong Kong, China, 12–13 November 2016. [CrossRef]
20. Tang, L.; Jiang, Y.; Lou, J. Reliability Architecture for Collaborative Robot Control Systems in Complex Environments. *Int. J. Adv. Robot. Syst.* **2016**, *13*. [CrossRef]
21. Berger, C.; Hansson, J. COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle. In Proceedings of the 32nd International Conference on Computer Safety, Reliability and Security, Toulouse, France, 24–27 September 2013; pp. 411–422.
22. Konys, A. A framework to COTS component selection and evaluation processes. *Electron. Rev.* **2015**, *91*, 86–90. [CrossRef]
23. Bali, V.; Madan, S. COTS evaluation & selection process in design of component based software system: An overview and future direction. *Glob. J. Comput. Sci. Technol.* **2015**, *15*, 1–8.
24. Mathopo, S.; Marnewick, A. Selection process for commercial-off-the-shelf products used as defense equipment. In Proceedings of the IEEE AFRICON, Cape Town, South Africa, 18–20 September 2017; pp. 682–687.
25. Rachucki, J.; Rugo, P. Analysis of scalable distributed on-board computer architecture for suborbital rockets and micro launchers. In Proceedings of the 8th European Conference for Aeronautics and Space Sciences, Madrid, Spain, 1–4 July 2019. [CrossRef]
26. Sanfilippo, F.; Helgerud, E.; Stadheim, P.A.; Aronsen, S.L. *Serpens*: A Highly Compliant Low-Cost ROS-Based Snake Robot with Series Elastic Actuators, Stereoscopic Vision and a Screw-Less Assembly Mechanism. *Appl. Sci.* **2019**, *9*, 396. [CrossRef]
27. Ma, L.; Xia, F.; Peng, Z. Integrated Design and Implementation of Embedded Control Systems with Scilab. *Sensors* **2008**, *8*, 5501–5515. [CrossRef] [PubMed]

28. Sarkar, D. Optimizing COTS selection process using prototype framework approach: A theoretical concept. In Proceedings of the 5th International Conference on Advanced Computing & Communication Technologies, Rohtak, Haryana, India, 7–8 February 2015; pp. 521–523.
29. Bundalo, Z.; Bundalo, D. Embedded Systems Based on Open Source Platforms. In *Introduction to Data Science and Machine Learning*; IntechOpen: London, UK, 2019. [CrossRef]
30. Tesanovic, A.; Nystrom, D.; Hansson, J.; Norstrom, C. Aspects and components in real-time system development: Towards reconfigurable and reusable software. *J. Embed. Comput.* **2005**, *1*, 17–37.
31. Pop, T. Component-Based Development for Real-time Embedded Devices: Current Obstacles and Next Steps Forward. In Proceedings of the 19th Annual Conference of Doctoral Students, Prague, Czech Republic, 1–4 June 2010; pp. 76–81.
32. Adam, G.K. DALI LED Driver Control System for Lighting Operations Based on Raspberry Pi and Kernel Modules. *Electronics* **2019**, *8*, 1021. [CrossRef]
33. Adam, G.K.; Kontaxis, P.A.; Doulos, L.T.; Madias, E.D.; Bouroussis, C.A.; Topalis, F.V. Embedded Microcontroller with a CCD Camera as a Digital Lighting Control System. *Electronics* **2019**, *8*, 33. [CrossRef]
34. Adam, G.K.; Kontaxis, P.A.; Bouroussis, C.A.; Ventzas, D.E.; Topalis, F.V. Embedded computer communication and control of DALI LED drivers. In Proceedings of the Balkan Light Conference, Athens, Greece, 16–19 September 2015; pp. 125–130.
35. Adam, G.K.; Garani, G.; Ventzas, D. FPGA design of a camera control system for road traffic monitoring. In Proceedings of the 6th International Conference from Scientific Computing to Computational Engineering, Athens, Greece, 9–12 July 2014; pp. 191–197.
36. Adam, G.K.; Karapoulios, C. Model-based qualitative studies of complex manufacturing systems. *WSEAS Trans. Inf. Sci. Appl.* **2004**, *1*, 88–93.
37. Petrellis, N.; Adam, G.; Ventzas, D. A real-time self-calibrated current mirror for wide range current reference generation, signal reproduction and delaying. *Elsevier Microelectron. J.* **2012**, *43*, 225–234. [CrossRef]
38. Adam, G.K. Process scheduling evaluations using multithreaded software modules. *AMSE Adv. Model.* **2005**, *10*, 13–26.
39. Garani, G.; Adam, G.K. Design, Simulation and Development of Software Modules for the Control of Concrete Elements Production Plant. In *New Approaches in Automation and Robotics*; I-Tech Education and Publishing: Vienna, Austria, 2008; pp. 261–282. [CrossRef]
40. The Linux Foundation. Real Time Linux. Available online: https://www.linuxfoundation.org/ (accessed on 14 January 2020).
41. Adam, G.K.; Grant, E. QMTOOL—A qualitative modelling and simulation CAD system for designing automated workcells. In Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, USA, 8–13 May 1994; pp. 1141–1146. [CrossRef]
42. Raspberry Pi Foundation. Raspberry Pi3 Model B. Available online: https://www.raspberrypi.org/products/raspberry-pi-3-model-b (accessed on 5 February 2020).
43. Ardito, L.; Torchiano, M. Creating and evaluating a software power model for linux single board computers. In Proceedings of the 6th International Workshop on Green and Sustainable Software, Gothenburg, Sweden, 27 May 2018; pp. 1–8. [CrossRef]
44. Membrey, P.; Veitch, D.; Chang, R.K.C. Time to Measure the Pi. In Proceedings of the ACM Internet Measurement Conference, Santa Monica, CA, USA, 14–16 November 2016; pp. 327–334. [CrossRef]