From the Cloud to the Clouds: Taking Integrated Modular Avionics on a New Level with Cloud-Native Technologies

Christian Rebischke
Clausthal University of Technology
Student ID: 432108
E-Mail: Christian.Rebischke@tu-clausthal.de

June 27, 2021

Acknowledgement

Statutory Declaration

This master thesis is submitted in partial fulfilment of the requirements for the Clausthal University of Technology. I hereby declare that this dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements. The contributions of any other supervisors to this thesis are made with specific reference.

Clausthal-Zellerfeld, June 27, 2021

Christian Rebischke

Abstract

The goal of this scientific work is to apply transfer knowledge from the cloud computing area to avionics and to contribute to a more heterogeneous research picture. The focus of this work lies in particular in the transformation of avionics architectures from a federated system to an integrated system, as well as its future development. The challenges and solutions of known architectures will be analyzed and compared with new achievements in cloud computing. In particular the Service Orientated Architecture (SOA) approach plays a role in this comparison, as well as its reliable, secure and cost-effective deployment in airplanes, drones or spaceships. The master thesis is structured as follows: In the introduction, the classification of the thesis is repeated and put in context with the current state of the art. Then, in the second chapter, the path from a federated avionics architecture to an integrated system will be shown and its problems, challenges and ideas will get isolated. This gained information is subsequently being compared with current cloud native technologies and potential solutions for these subject will be proposed.

Zusammenfassung

Ziel dieser wissenschaftlichen Arbeit ist es Transferwissen aus dem Cloud Computing Bereich auf die Avionik anzuwenden und dazu zu einem heterogeneren Forschungsbild beizutragen. Im Fokus der Arbeit liegt insbesondere der Weg der Avionik Architekturen von einem föderierten System hin zu einem integrierten System, sowie dessen zukünftige Weiterentwicklung. Dabei sollen die Herausforderungen und Lösungen von bekannten Architekturen analysiert und mit neuen Errungenschaften aus dem Cloud Computing Bereich verglichen werden. Insbesondere der Service Orientierted Architecture (SOA) Ansatz spielt in diesem Vergleich eine Rolle, sowie dessen zuverlässige, sichere und kostengünstige Einsatzmöglichkeiten in Flugzeugen, Drohnen oder Raumschiffen. Die Masterarbeit ist wie folgt gegliedert: In der Einleitung wird die Einordnung der Arbeit wiederholt und in einen Zusammenhang mit der Gegenwart gestellt. Im Zweiten Kapitel wird dann der Weg von einer föderierten Avionik Architektur zu einem integrierten System beleuchtet und dessen Probleme, Herausforderungen und Ideen isoliert. Diese gewonnenen Informationen werden nachfolgend aktuellen Cloud Native Technologien gegenüber gestellt und potentielle Lösungen vorgeschlagen.

Contents

Chapter 1

Introduction

The number of performed flights by global airline industries increased from 23.8 million flights (2004) to 38.9 million flights (2019)[19]. This growing number of performed flights puts an enormous pressure on the global aviation industry as a whole. The permanent price pressure lead to demands of cheaper, lighter and smaller flight components[20]. Every inch and every gramm counts in the global business of civil aviation, because every inch less means one possible paying customer more on the plane and every gramm less means less expensive fuel demands for the flight. But it is not only the underlying architecture and the corresponding hardware that plays a big role in the aviation business. The software forming these architectures and running on these devices plays an equal important role in the aviation industry. The development of software is difficult, error-prone, tedious and expensive. This leads to the question why the aviation industry is not exploiting resources and development processes from other industry branches. The open source software movement provides a staggering amount of different technologies for solving problems that are not too different to the problems from the aviation industry. Reasons for this development paralysis are regulation and certification. The civil aviation sector is strictly regulated, thus experimenting with alternatives is expensive and difficult. Furthermore, the existing certification companies are not known for their disruptive technology announcements. Nevertheless this thesis tries to explore a few of these alternatives and tries to suggest topics that might be interesting for further research. Hopefully it will help justifying further research in this area and incite changes in the inflexible regulation and certification chain. The United States (US) military sector and the US space industry seem to be more willing to experiment with new or existing open source software. For example, the private US space company *SpaceX* had tremendous success with Linux as operating system on their *Dragon* spacecraft[11] and Linux is not only being used by SpaceX[15]. Of course this success is only possible, because the space industry is much more isolated and kept secret than the civil aviation industry with their international standards and guidelines. One of these standards is the DO-178B certification and its successor DO-178C from the Radio Technical

Commission for Aeronautics (RTCA). This certification has strict requirements on flight operating systems. A few of these requirements are real-time capabilities and a transparent and documented development process with design decisions and other documents. While real-time capabilities can be easily added via soft patches (*SpaceX* is exactly doing this with their *Dragon* spacecraft[11]), the documentation and development process seem to be an invincible obstacle for a successful certification process. Another prominent example of open source adoption is the operation of the Cloud Native container orchestration engine Kubernetes in military fighter planes, like the US military plane *U-2*[23]. Unfortunately there is no research on that topic. This thesis tries to change this as well and tries to connect existing research in both areas for creating synergies between them, but for connecting these two areas we need to understand both of these areas first. Therefore, the next chapter will give an introduction to the history of software architectures on planes and will highlight the most important challenges in these.

Chapter 2

Related Work

2.1 Federated Avionics

To understand the background of this thesis better it is recommended to understand the journey of flight system architectures. Around the 1970's avionics systems evolved from traditional point-to-point wiring to a standard data bus with a federated system architecture[28]. This federated system has been implemented as distributed collection of dedicated computing resources consisting of Line-Replacable Units (LRUs) or Line-Replacable Modules (LRMs)[26]. LRUs and LRMs are modular components that are specifically designed for predetermined tasks, such as interacting with certain flight sensors/effectors[14]. Sensors are reading data and effectors are executing certain actions, for example moving the flight gears. The main advantages of LRUs are their atomic behavior and their strict and easily certifiable system design. Each LRU or LRM contains one specific avionic workload and its required computing resources (processors, input and output (I/O) modules, main memory, hard disks and network cards). Figure 2.1 shows a simplified model of the federated architecture with distributed LRUs, sensors, effectors, and a global data bus connecting the components. It visualizes the enormous effort and the huge amount of cables. Duplicating the systems achieves service redundancy and ensures system reliability[20], for the price of duplicating the LRU as a whole. This does not only mean a duplicated hosted function (the actually functionality of the LRU), it also means double as much cables, processors, main memory, network cards and connectors. Weight and complexity disadvantages are not the only problems with the federated architecture approach. Having a dedicated hardware stack for each LRU means not fully saturated potential. Due to safety reasons the LRU will very unlikely use all of its resources. This means there will be always a spare amount of main memory, hard disk or network saturation. Added up over the whole federation architecture this means a huge amount of unused resource potential and unnecessary energy consumption that could be used for other functionality. The task-specific development of LRUs leads to problems

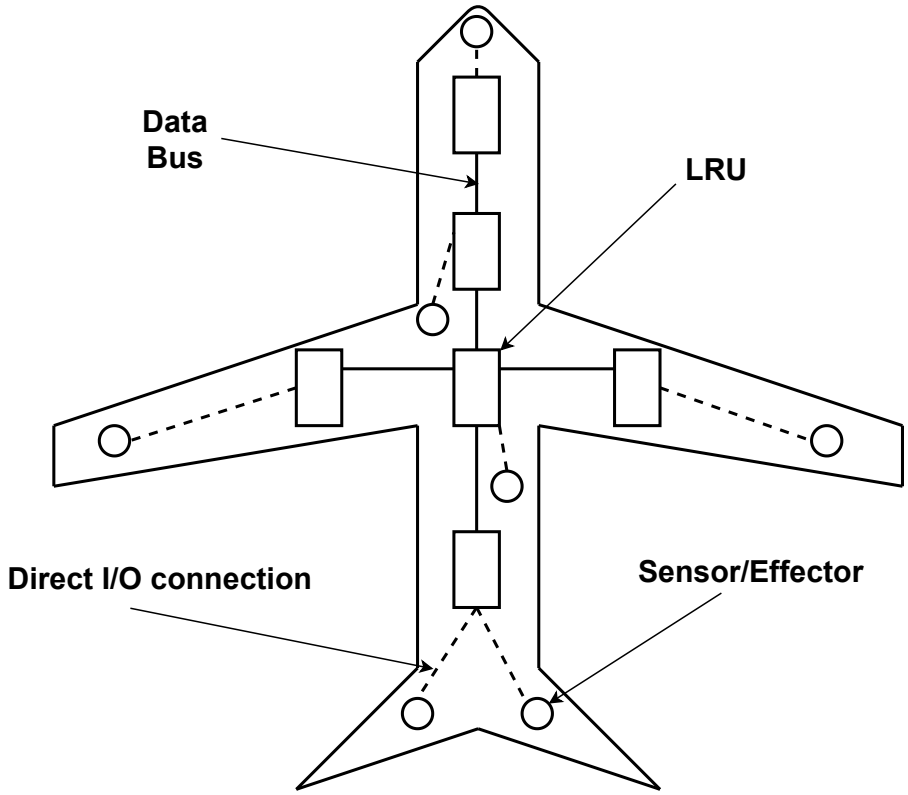

Figure 2.1: Simplified visualization of the federated avionics architecture, showing LRU, sensors, effectors, and the global data bus

with functionality extensions, meaning that LRUs are not easily upgradable or extendable and therefore adjustable to new tasks or functionality. Moreover in the past the development of federated architecture components has been closed source and very vendor specific. Specifications for federated architecture components were mostly hidden behind a paywall or non disclosure agreements leading to a decreased developer efficiency and less market competition due to monopolism. LRUs are not easily exchangable between vendors. Figure 2.2 displays the internal structure of the LRU and its interaction with other LRU. Each LRU provides its own hardware stack and hosts exactly one avionic function. It is possible for one LRU to speak with multiple sensors or effectors, but this does not change their core purpose of hosting exactly one function. In section 2.2 this thesis will investigate the next step in the evolution of flight system architectures and how the disadvantages of federated architectures can be fixed.

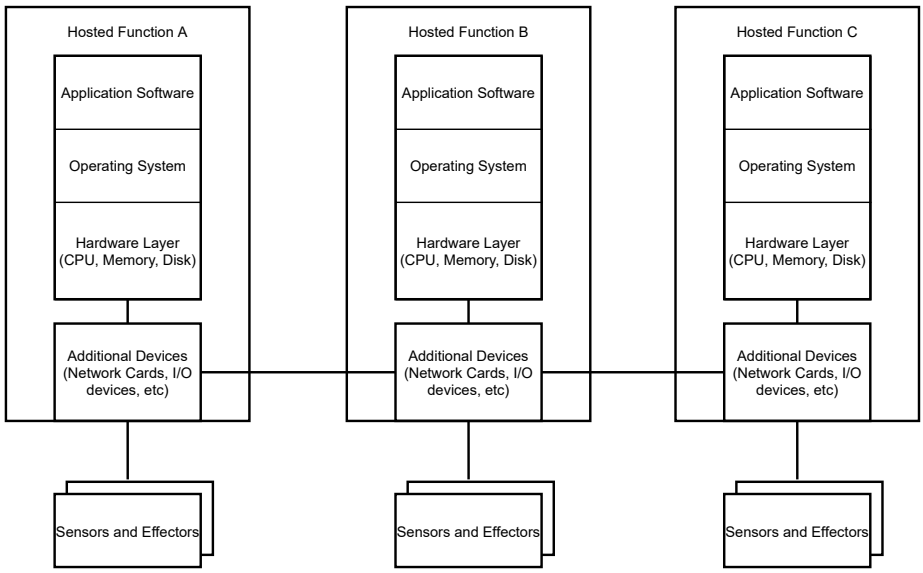

Figure 2.2: Internal system architecture and interaction between LRUs

2.2 Integrated Modular Avionics

Integrated Modular Avionics (IMA) is the direct successor of the federated avionics architecture. The idea behind Integrated Modular Avionics (IMA) is to consolidate the distributed hardware in one central flight cabinet. A Flight cabinet is very similar to a rack in a datacenter. They can host multiple processing units, each comes with its own hardware stack consisting of a Central Processing Unit (CPU), Random Access Memory (RAM), disk space and connectors for input and output[20]. These servers are then plugged-in into the flight cabinet. Each server can host more than one avionic function and each function is allocated on partitions. Partitions can be created on different ways and has been standardized in ARINC 653 standard[24]. The most common approach is the use of a hypervisor (how this is implemented is being discussed in section 2.4). The flight cabinet provides power and required network connection to the plane's global data bus as described in ARINC 629 standard[13] or ARINC 429 standard[9]. ARINC 629 is the successor of the data bus ARINC 429. Effectors and sensors communicate with the flight cabinet over ARINC 629[20]. Sensors or effectors that are incompatible with ARINC 629 may communicate over remote data concentrators. Remote data concentrators are gathering data from sensors or sending data to effectors over traditional I/O connections. The gathered data or the received actions are communicated via ARINC 629 or ARINC 429. Therefore, remote data concentrators act as bridges between such devices and the data bus. Using a central flight cabinet cannot replace all LRUs in the plane[26]. These LRUs needs to be either integrated into the flight cabinet or connected to the global data bus, for example via remote data con-

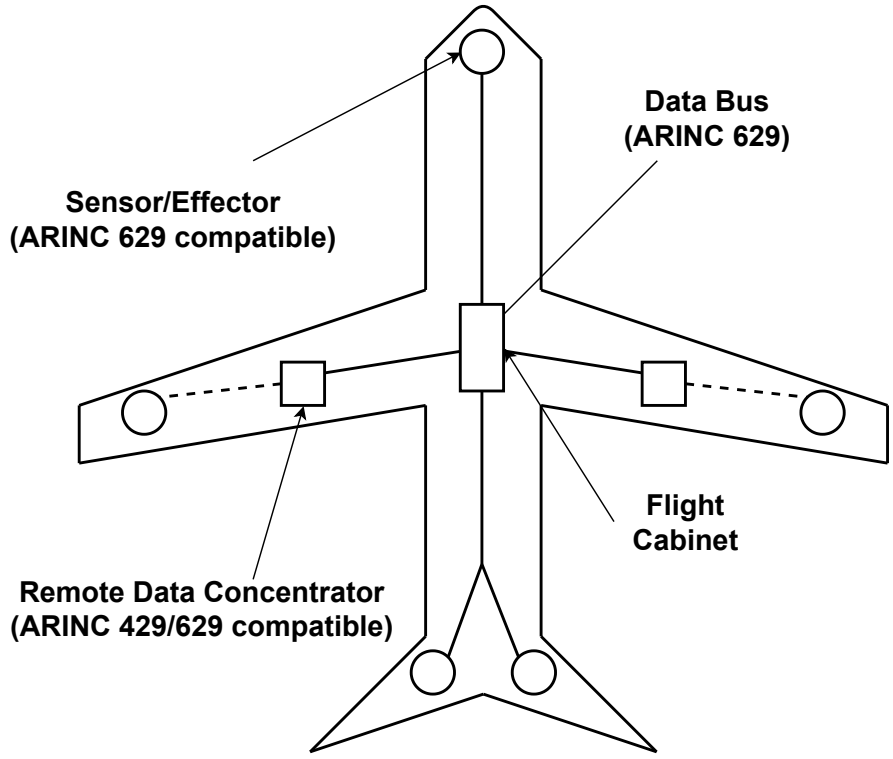

Figure 2.3: Simplified visualization of the integrated modular avionics architecture, showing sensors, effectors, cabinets and the data bus LRUs

centrators. Figure 2.3 depicts a simplified view on integrated modular avionics architecture in a plane. The number of LRUs has been reduced and one central flight cabinet has been introduced. Remote data concentrators are working as bridges between sensors and effectors incompatible with the data bus standard and the hosted functions in the flight cabinet. Figure 2.4 shows the modules inside of such a flight cabinet. The flight cabinet possesses multiple processing units. Each processing unit is comparable to a dedicated computer with its own hardware and operating system. These processing units are being connected via a network layer and each processing unit hosts one or more hosted functions. Hosted functions are isolated from each other and have a fixed predetermined set of resources and execution time.

This system architecture has numerous advantages over the federated avionics architecture. Due to the more centralized approach IMA is able to reduce weight via better cable management and less distributed processing units. Computing resources can be used more efficiently via hosting multiple avionics functions on one processing unit. This leads to a higher system saturation. A positive side effect is less energy consumption and a smaller ecological foot-

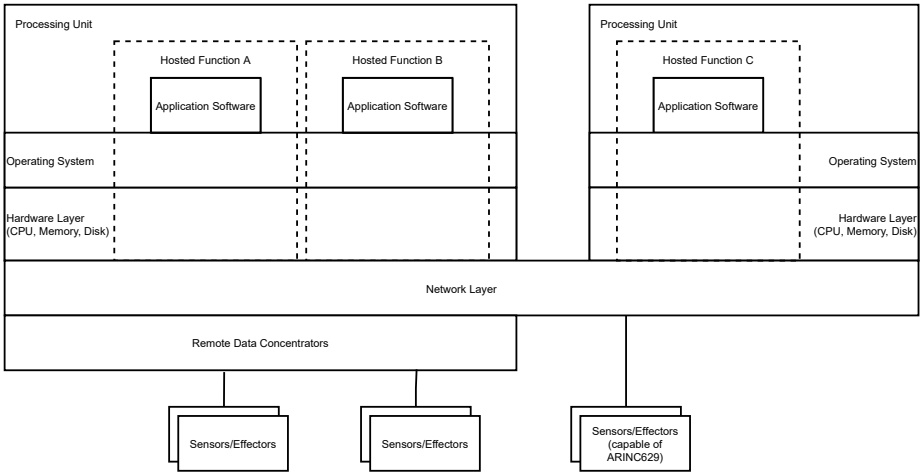

Figure 2.4: View into a flight cabinet

print. The reduced weight creates more space for cargo, fuel or passengers. Hardware consolidation leads to a consolidation of development efforts, which achieves cost and time savings[26]. The common processor allows the developer to focus on the hosted avionic function, enabling a better development experience and less error-prone flight software. Separating software and hardware is a benefit during the certification process, because the certifying authority can certify software and hardware separately. This has just another huge impact on cost and time savings. Additionally, upgrading the hosted function becomes a lot easier, because of the hardware and software separation and less expensive hardware, due to standardized and more common processing units. Another important benefit can be achieved via adopting the idea of open software and hardware. An open system architecture with open standards can lead to a more competitive market due to industry-wide participation and exchangeability between hardware or software applications. This way development and hardware costs can be reduced, because development cost gets distributed among all contributing companies and mass production of standardized and open hardware shrinks marginal costs[3]. Moreover, the decoupling between hardware and software can have a positive effect on new emerging companies, considering the lower costs[26]. Software virtualization makes it easier to develop the software, without buying expensive physical development kits. The software is being virtualized, tested and can be much later evaluated on real hardware, speeding up the development process and time to market.

2.3 Distributed Integrated Modular Avionics

While IMA introduced a central architecture via consolidating computing resources into one central flight cabinet Distributed Integrated Modular Avionics

(DIMA) takes a different approach. IMA has shown that it is able to successfully reduce the number of components in the plane with increasing number of hosted functions, because of its shared hosting infrastructure[10]. Although this had positive effects on weight and cost management, there is room to improve in form of cable management. Due to the centralized architecture it is necessary to connect every sensor and effector in the plane with the central flight cabinet in the fuselage[17]. This possible increase of cable length can be prevented via using ideas from the federated avionics and IMA. DIMA suggests a distribution of processing units, while taking into account the advantages of a central flight cabinet. Instead of one central flight cabinet it is possible to redistribute the avionics functions over multiple flight cabinets distributed in the plane's fuselage (see also Figure 2.5). This way it is possible to successfully exploit the advantages of the integrated architecture, while achieving further improvements in the cable management[16]. Cable management is not the only possible adjustment for improvement. The commercial cloud computing industry experienced a huge success over the last couple of years and these achievements can be used for creating synergies between these two areas and applying cloud computing ideas in DIMA. According to the National Institute of Standards and Technology (NIST) cloud computing is defined as:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[18]

Although not all aspects of cloud computing are applicable on aviation, some of these aspects are. One of these aspects is the separation into different service layers:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

SaaS is handling all applications. PaaS is responsible for the platform, where these applications are running on (for example a hypervisor or services like webserver or databases). IaaS is the underlying infrastructure (server, storages, networking). These three layers can be mirrored on the aviation world as follows:

- SaaS: Hosted functions
- PaaS: The virtualized processing units or separation kernels
- IaaS: flight cabinets, remote data concentrators, the plane's data bus

Via separating each of these layers the aviation industry gains stronger standardization, more reliability and more effective development workflows. Instead

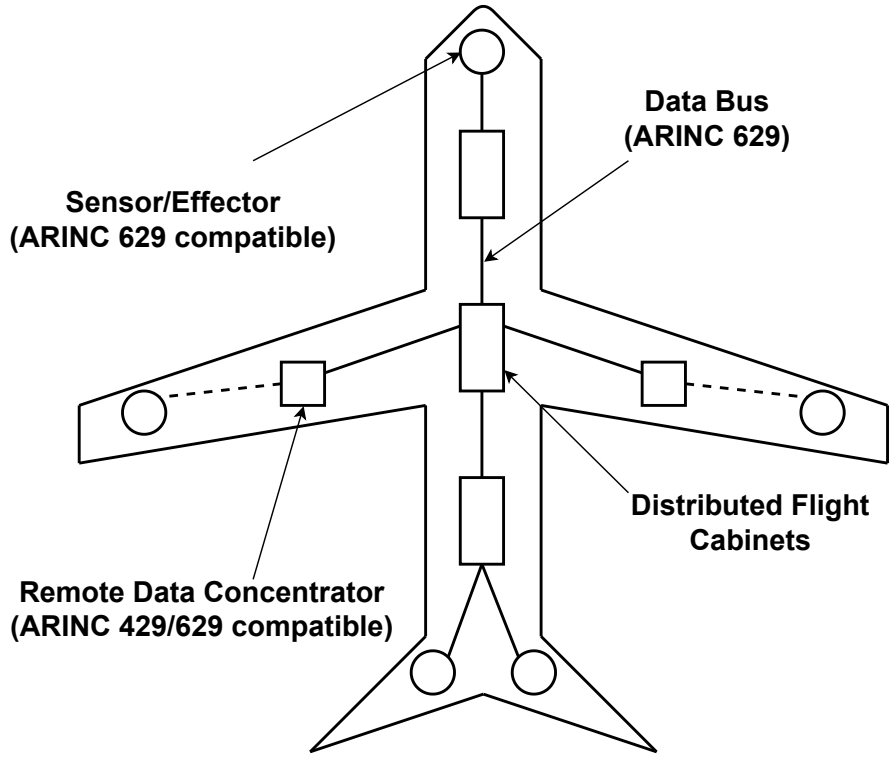

Figure 2.5: Visualization of distributed flight cabinets

of selling one big monolithic system, separating makes it possible to develop products for specific layers and interconnect them with the other layer via standards. These standards can be, as shown in section 2.2, drivers for competition and exchangeability. Another aspect of cloud computing is the free and configurable allocation of applications[16]. Dedicated storage, processor or sensor clusters would allow on-demand access from applications. Nevertheless it is a challenge to make this access reliable and safe. Furthermore using standardized software may allow interconnection between the plane and ground units (for example: real-time weather data transmitted from the ground to the plane). This is partly comparable to the tactical data link of military units that get real-time radar data on their Horizontal Situation Display (HSD).

2.4 Workload Partitioning Strategies

Planes are composed of complex distributed systems with different tasks, requirements and environmental influences. Every system has its own set of risks and possible impacts. These different risks are known as Design Assurance Level (DAL) or Item Development Assurance Level (IDAL) and have been defined in

the Aerospace Recommended Practic (ARP) 4754 as follows[1]:

A Catastrophic system failures

B Hazardous system failures

C Major system failures

D Minor system failures

E No safety effect

Due to these different risk levels it becomes important to isolate systems from each other. A system with a low risk level should never have direct nor indirect impact on another system. In conclusion the following measures are necessary to satisfy the safety and security requirements in planes:

- Reserved CPU time
- Reserved memory
- Reserved disk space
- Qualityf of Service (QoS)
- Network isolation
- Process isolation
- Privilege dropping

Reserved CPU time means that every task has a strict time partition to work with. Normally, cpu time is being shared on single processor systems. The operating systems simulates parallelism via fast context switches. Multiple tasks are being executed consecutively switching between them. This creates the impression that the system behaves parallel, but in reality it does not. On multi processor systems real parallelism is possible. Reserved memory refers to a fixed memory partition in the RAM. Reserved disk space refers to guaranteed space on a hard disk for storing persistent data. Network isolation and Process isolatio ensures a noisefree and secure runtime environment. Processes should not be disturbed by other processes either on process or network level. Furthermore, each process has to drop privileges. Dropping privileges increases the safety and security of a service, because it is less likely to interfere with other processes if the process has no privileges to do so. QoS gives certain processes wit higher importance or risk level a higher priority. For example, a system that controls the radar should have a higher priority than a system that just controls the ambient lights in the passenger cabin. Many of the above mentioned measures reveal other measures as direct dependencies. For example, it is not possible to do proper QoS without an observable system that gets properly monitored. Another example is the importance of security. On the first view security might

be untangled of safety, but in reality safety and security have a direct effect on each other. If the system is insecure it cannot be safe and reliable, because every possible security incident could undermine all reliability promises. The same applies to safety. An unsafe and unstable system cannot be considered secure, because these weak points in the reliability might weaken the security. Just imagine a system that controls permissions or security related functions. The system itself may be secure, but if it is unreliable it may has a direct effect on the security of other systems that depend on this particular system. Additionally, planes have further constraints and requirements on software. One of these requirements is a guarantee on response. Certain software systems in planes must respond before a given deadline. This is called real-time communication. Real-time communication breaks down into the following types[27]:

Hard The system has to satisfy all constraints. Responses are always on time. If a response would come too late it would inflict damage.

Firm Executed tasks are worthless when their response does not satisfy the time constraint. No damage happens.

Soft There are time constraints for requests and responses, but a tolerance exist. Delays are acceptable (as long as they fit in a specific time frame).

With all of these requirements it is obvious that there need to be technical implementations to satisfy all constraints. The real-time constraints can be satisified via implementing real-time capabilities on operating system or kernel level. Prominent examples are VxWorks' real-time operating system[25], FreeRTOS[21] or the real-time patches for the linux kernel[22]. All other requirements are being solved either via virtualization or via containerization. There are two types of virtualization: Full virtualization and paravirtualization. While full virtualization creates an almost full virtualized hardware including virtualized memory or I/O devices, the paravirtualization does only virtualize the software layer. The hardware stack will not be virtualized. Although this approach increases performance it is considered as less secure. With virtualization it is possible to successfully isolate processes from each other and provide a dedicated workload partition for a specific software with its own RAM, CPU or security requirements. The keye element in this approach is the hypervisor. The hypervisor is the abstraction layer between the hardware and the software and their virtualized counterpart. Type 1 hypervisors run directly on the hardware, while type 2 hypervisors are running on a dedicated host OS. Both hypervisors can usually virtualize a finite number of guests. Guests can be either applications or whole operating systems. Many hardware provide direct support for virtualization, such as special instruction sets for CPUs. The containerization uses operating system or kernel level features to isolate processes or resources. In the Linux kernel this usually works via making use of namespaces, kernel capabilities, control groups and union file systems.

2.5 Summary

Bibliography

[1] SAE ARP4754A. "Guidelines for development of civil aircraft and systems". In: *SAE International* (2010).

[2] Tjerk Bijlsma et al. "A distributed safety mechanism using middleware and hypervisors for autonomous vehicles". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1175–1180.

[3] Randy Black and Mitch Fletcher. "Open systems architecture-both boon and bane". In: *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*. IEEE. 2006, pp. 1–7.

[4] Todd Carpenter et al. "ARINC 659 Scheduling: Problem Definition." In: *RTSS*. 1994, pp. 165–169.

[5] Marcello Cinque and Gianmaria De Tommasi. "Work-in-Progress: Real-Time Containers for Large-Scale Mixed-Criticality Systems". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2017, pp. 369–371.

[6] Marcello Cinque et al. "Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets". In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019.

[7] Luigi De Simone and Giovanni Mazzeo. "Isolating Real-Time Safety-Critical Embedded Systems via SGX-based Lightweight Virtualization". In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, pp. 308–313.

[8] Abhishek Dubey et al. "Enabling strong isolation for distributed real-time applications in edge computing scenarios". In: *IEEE Aerospace and Electronic Systems Magazine* 34.7 (2019), pp. 32–45.

[9] Christian M Fuchs and Advisors Stefan Schneele. "The evolution of avionics networks from ARINC 429 to AFDX". In: *Innovative Internet Technologies and Mobile Communications (IITM), and Aerospace Networks (AN)* 65 (2012), pp. 1551–3203.

[10] Rudolf Fuchsen. "Preparing the next generation of IMA: A new technology for the scarlett program". In: *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*. IEEE. 2009, 7–B.

[11] Jim Gruen. "Linux in space". In: *The Linux Foundation* (2012).

[12] Sanghyun Han and Hyun-Wook Jin. "Kernel-level ARINC 653 partitioning for Linux". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 2012, pp. 1632–1637.

[13] Yasemin Isik. "ARINC 629 data bus standard on aircrafts". In: *Recent Researches in Circuits, Systems, Electronics, Control & Signal Processing* (2010), pp. 191–195.

[14] SJ Lemke. *A Comparative Evaluation of the Reliability Improvement in Line Replaceable Units Warranted under the F-16 Reliability Improvement Warranty.* Tech. rep. AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYSTEMS and LOGISTIC S, 1985.

[15] Hannu Leppinen. "Current use of Linux in spacecraft flight software". In: *IEEE Aerospace and Electronic Systems Magazine* 32.10 (2017), pp. 4–13.

[16] Zheng Li, Qiao Li, and Huagang Xiong. "Avionics clouds: A generic scheme for future avionics systems". In: *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*. IEEE. 2012, 6E4–1.

[17] M McCabe, C Baggerman, and D Verma. "Avionics architecture interface considerations between constellation vehicles". In: *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*. IEEE. 2009, 1–E.

[18] Peter M Mell and Timothy Grance. *Sp 800-145. the nist definition of cloud computing*. 2011.

[19] *Number of flights performed by the global airline industry from 2004 to 2021*. Statista. 2021. URL: https://www.statista.com/statistics/564769/airline-industry-number-of-flights/ (visited on 04/03/2021).

[20] Paul J Prisaznuk. "Integrated modular avionics". In: *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference@ m_ NAECON 1992*. IEEE. 1992, pp. 39–45.

[21] *Real-time operating system for microcontrollers*. FreeRTOS. 2021. URL: https://www.freertos.org/index.html (visited on 06/27/2021).

[22] *The Linux Foundation - Real-Time Linux*. The Linux Foundation. 2021. URL: https://wiki.linuxfoundation.org/realtime/start (visited on 06/27/2021).

[23] *U-2 Federal Lab achieves flight with Kubernetes*. Air Combat Command Public Affairs. 2020. URL: https://www.af.mil/News/Article-Display/Article/2375297/u-2-federal-lab-achieves-flight-with-kubernetes/.

[24] Steven H VanderLeest. "ARINC 653 hypervisor". In: *29th Digital Avionics Systems Conference*. IEEE. 2010, 5–E.

[25] *VxWorks, the leading RTOS for the intelligent edge*. VxWorks. 2021. URL: https://www.windriver.com/products/vxworks (visited on 06/27/2021).

[26] Christopher B Watkins and Randy Walter. "Transitioning from federated avionics architectures to integrated modular avionics". In: *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*. IEEE. 2007, 2–A.

[27] Heinz Wörn. *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*. Springer-Verlag, 2006.

[28] HG Xiong and ZH Wang. "Advanced avionics integration techniques". In: *National Defense Industry Press, Beijing, China* (2009).

Listings

List of Tables

List of Figures

Glossary

AEEC Airlines Electronic Engineering Committee.

APEX APplication/EXecutive Interface.

ARINC Aeronautical Radio Incorporated.

ARINC 429 ARINC standard for a global data bus in aviation. 10

ARINC 629 ARINC standard for a global computer bus in aviation. 10

ARINC 650 ARINC standard for IMA packaging and interfaces.

ARINC 651 ARINC report that provides guidelines for a maintenance strategy for IMA-equipped airplanes.

ARINC 653 ARINC standard for space and time partitioning. 10

ARINC 659 ARINC standard for a backplane data bus.

ARP4754 Guideline for the development of aircraft systems by SAE International. 15

COEX COre/EXecutive Interface.

CPU The CPU consists of registers for fast computation and an Algorithmic Logic Unit (ALU). 10, 15, 16

DAL Design Assurance Level. 14

DIMA Distributed computer network airborne system. 12, 13

DO-178B Certification for safety-critical software by the RTCA. 6

DO-178C Certification for safety-critical software by the RTCA (replaces DO-178B). 6

HSD . 14

I/O input and output. 8, 10, 16

IaaS . 13

IDAL Item Development Assurance Level. 14

IMA Computer network airborne system. 10–13

LRM Line-Replaceable Module. 8

LRU Line-Replaceable Unit. 8–11, 23

NIST United States institute for promoting innovation and industrial competitiveness. 13

PaaS . 13

QoS Quality of Service. 15

RAM RAM is very fast memory for temporary storing data. 10, 15, 16

RTCA Radio Technical Commission for Aeronautics (RTCA). 6

SaaS . 13

US United States of America. Short: United States. 6, 7