A SURVEY OF TIME AND SPACE PARTITIONING FOR SPACE AVIONICS

Jan Bredereke · City University of Applied Sciences Bremen
mailing address: Flughafenallee 10, 28199 Bremen, Germany
email: jan.bredereke@hs-bremen.de · phone: +49 421 5905 5490

ABSTRACT

We present a survey of the current state of the reseach on time and space partitioning (TSP) for space avionics. The availability of ever more powerful computers allows to assign many control tasks to a single computer easily, in principle. But in its naïve form, this would mean too much effort and thus cost for demonstrating dependability. For aircraft, there is already an approach to solve this problem, the Integrated Modular Avionics (IMA) architecture. For spacecraft, the basic problem is similar. But in detail, the setting is different, though. This paper compiles a survey and identifies relevant research challenges.

Keywords: time and space partitioning, space avionics, integrated modular avionics

1 INTRODUCTION

Avionics is the set of electronic systems, in particular computers, on board of an aircraft or spacecraft. Such systems must be dependable. There are significantly different levels of dependability, for example for the cabin electronics of an aircraft and for its autopilot. The higher the level required, the higher the effort necessary for demonstrating dependability. The traditional approach therefore provisions separate computer hardware for each control task. The effort for demonstrating dependability then follows from the level targeted, for each task.

The availability of ever more powerful computers allows to assign many control tasks to a single computer easily, in principle. This would have the advantage of saving considerable weight and thus cost for the aircraft or spacecraft. But this idea, in its naïve form, would mean too much effort and thus cost for demonstrating dependability. A level demanded by one of the tasks must be applied to all of them.

For aircraft, there is an approach for this already. The Integrated Modular Avionics (IMA) architecture (see Sect. 3 below) is a software and hardware platform which isolates software programs from each other. The dependability needs to be demonstrated at the highest level only once, for the platform. This concept is used in practice already for the aircraft Airbus A380, Airbus A400M, and Boeing 787 Dreamliner.

For spacecraft, the basic problem is similar. We want to run several tasks on a single computer (or a few computers) for cost reasons, with dependability requirements that are high, too. But in detail, the setting is different, though. There is already some research on time and space partitioning of control tasks in space craft. But there is not yet a concept proven in practice.

The aim of our paper is to compile a survey and to identify some relevant research challenges. The paper is an updated version of a longer, more comprehensive technical report by us [Bre17].

2 SYSTEMS WITH MIXED DEPENDABILITY

In this section, we introduce to systems with mixed dependability. We briefly introduce to the notions of dependability and of mixed dependability first. We then describe two mechanisms to handle mixed dependability.

2.1 Dependability

The survey paper by Avižienis *et. al.* [Avi+04] provides definitions for dependability in the domains of computing and communication systems. The original definition of dependability is the ability to deliver service that can be justifiably trusted. This definition stresses the need for justification of trust. The alternate definition that provides the criterion for deciding if the service is dependable is: the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable. The means to attain dependability are fault prevention, fault tolerance, fault removal, and fault forecasting.

All of these means to attain dependability require effort. And this effort increases when a higher dependability is demanded.

2.2 Mixed Dependability

A system with mixed dependability is a system where components with different dependability levels coexist on the same execution platform. Crespo *et. al.* [Cre+14a] call this a "mixed criticality system" (MCS): Increasing processing power makes it possible to integrate more and more components on a single execution platform. However, if at least some of the components must be dependable, adequate validation (and often certification) is necessary, such that validation costs can become prohibitive for further integration. This trend can be observed in many different domains. Examples are the aeronautical domain, the space domain, the automotive domain, and industry automation.

2.3 Handling Mixed Dependability

A general approach to this is to separate the components on the single execution platform so well that only the separation mechanism and the high dependability components need to be validated for a high dependability. Mechanisms to achieve such a separation comprise a *separation kernel* and *virtualization*.

2.3.1 Separation Kernel

A separation kernel is a combination of hardware and software for allowing multiple functions to be performed on a common set of physical resources without interference [Cre+14a]. It was first proposed by Rushby [Rus81], aiming at security problems.

According to Crespo *et. al.* [Cre+14a], the MILS architecture, developed by the MILS (Multiple Independent Levels of Security and Safety) initiative [Alv+05], is a separation kernel. In addition, the ARINC-653 standard [Aer05] uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture. Wie will describe IMA in more detail in Sect. 3.

2.3.2 Virtualization

Crespo *et. al.* [Cre+14a] use virtualization as a separation mechanism. A hypervisor implements partitions or virtual machines that are isolated from each other in the temporal and spatial (i.e., storage) domains.

Different kinds of isolation must be considered [Cre+14a, Sect. 2.1]:

Fault isolation: a fault in an application must not propagate to other applications.

Spatial isolation: applications must execute in independent physical memory address spaces.

Temporal isolation: the real-time behaviour of an application must be correct independent of the execution of other applications.

Crespo *et. al.* [Cre+14a, Sect. 2.1] propose a predefined and static allocation of resources to partitions, in order to achieve a separation that is sufficiently simple to allow for separate validation. The resources comprise CPU time, memory areas, IO ports, etc. Static allocation of CPU time should be achieved by a cyclic scheduling policy for partition execution.

Crespo *et. al.* [Cre+14a, Sect. 3] state that there is some confusion of terminology on virtualization, and they propose the following definitions: A *type 1 hypervisor* (also named native or bare-metal hypervisor) runs directly on the native hardware, while a *type 2 hypervisor* is executed on top of an operating system. *Full virtualization* provides a complete re-creation of the hardware behaviour of a native system to a guest system, while *para-virtualization* requires the guest system to be modified: Some machine instructions are replaced by functions provided by the hypervisor. In full virtualization, certain "conflicting" machine instructions must be caught during runtime, in order to maintain the spatial and temporal separation. They are then handled by the hypervisor. With para-virtualization, in contrast, no catching is necessary, and the handling can use more information from the guest. This improves the performance greatly, and it simplifies the hypervisor. Of course, the source code of the guest must be available for recompiling. Since the latter usually is not a problem for mixed dependability systems, para-virtualization is preferable here.

Conceived by the same research group, XtratuM (see Sect. 4.4.1) is an open-source, type 1 hypervisor that uses para-virtualization. It was designed specifically for mixed-criticality systems.

Virtualization aims at presenting a virtual machine to an application software which is exactly like the real machine. However, it cannot hide one kind of difference: the machine instructions are not executed evenly in the time sense anymore. There are "holes" where other partitions get time. The application software can notice this when it interacts with the system's environment. This is relevant for real-time applications, for example for control applications where the controlled system does not stop while the application is on hold. Another example is the interaction with peripheral devices which change state by progress of time, such as timers.

The latency of an interrupt can become substantially higher, i.e., until the partition of the interrupt is scheduled again. This can break assumptions about the timing of interrupts made by an application or by an operating system. For example, Ripoll *et. al.* [Rip+10, Sect. 3.2] report that they used the RTEMS operating system in a partition, and that the timer tick of the RTEMS operating system was faster than the schedule period of the partition. Therefore, they had to take measures to ensure that accumulated clock ticks were presented to the partition at the beginning of its time slot.

2.3.3 Separation Kernel vs. Virtualization

Separation is a solution to the problem of mixed dependability, and virtualization is one possible mechanism to achive separation. Virtualization comprises more than necessary to achive separation. For example, the application in a (fully) virtualized machine cannot "see" any differences (besides "holes in time") to a dedicated real machine. This is not necessary for separation. Instead, we can provide some modification of a real machine to the application, as long as it guarantees separation. The approach of para-virtualization (compare Sect. 2.3.2 above) goes a step into exactly this direction.

A separation kernel imposes the use of the same operating system onto all applications, while virtualization allows for different operating systems (or even bare-metal applications without any operating system) in its partitions. The latter can be a substantial advantage if heterogeneous applications are to be integrated.

A separation kernel usually comprises more functionality (e.g., on communication, and maybe on multi-threading) than a hypervisor used for virtualization. Therefore, we suspect that the effort necessary for verifying the time and space separation property is usually higher for a separation kernel.

3 INTEGRATED MODULAR AVIONICS (IMA) FOR AIRCRAFT

In this section, we give an overview of the Integrated Modular Avionics (IMA) architecture which is used for aircraft. We describe why it was developed, and which are the key properties of its constituent data network and of its operating system interface. Furthermore, we outline the extension to Distributed Modular Electronics (DME).

Our overview draws on the well-written background chapter of the dissertation thesis of Efkemann [Efk14, Chap. 2]. A good source for further reading is Ott [Ott07]. She gives a broad view on both architectures and development processes, and there in particular on testing processes.

3.1 From a Federated Architecture to the IMA Architecture

Efkemann [Efk14, Chap. 2] presents the following overview of Integrated Modular Avionics (IMA) in his dissertation thesis:

"The traditional *federated* aircraft controller architecture [Fil03, p. 4] consists of a large number of different, specialised electronics devices. Each of them is dedicated to a special, singular purpose (e. g. flight control, or fire and smoke detection) and has its own custom sensor/actuator wiring. Some of them are linked to each other with dedicated data connections. In the *Integrated Modular Avionics* (IMA) architecture this multitude of device types is replaced by a small number of modular, general-purpose component variants whose instances are linked by a high-speed data network. Due to high processing power each module can host several avionics functions, each of which previously required its own controller. The IMA approach has several main advantages:

- Reduction of weight through a smaller number of physical components and reduced wiring, thereby increasing fuel efficiency.
- Reduction of on-board power consumption by more effective use of computing power and electrical components.
- Lower maintenance costs by reducing the number of different types of replacement units needed to keep on stock.
- Reduction of development costs by provision of a standardised operating system, together with standardised drivers for the avionics interfaces most widely used.
- Reduction of certification effort and costs via incremental certification of hard- and software.

An important aspect of module design is *segregation*: In order to host applications of different safety assurance levels on the same module, it must be ensured that those applications cannot interfere with each other. Therefore a module must support resource partitioning via memory access protection, strict deterministic scheduling and I/O access permissions. Bandwidth limitations on the data network have to be enforced as well.

The standard aircraft documentation reference for IMA is ATA chapter 42. The IMA architecture is currently in use in the Airbus A380, A400M, the future A350XWB, and Boeing 787 Dreamliner aircraft. Predecessors of this architecture can be found in so-called fourth-generation jet fighter aircraft like the Dassault Rafale."

3.2 AFDX Data Network

Efkemann [Efk14, Chap. 2.1] continues with an overview of the AFDX data network employed by the IMA architecture:

"A data network is required for communication between (redundant) IMA modules as well as other hardware. This role is fulfilled by the *Avionics Full DupleX Switched Ethernet* (AFDX) network. It is an implementation of the ARINC specification 664 [Aer09] and is used as high-speed communication link between aircraft controllers. It is the successor of the slower ARINC 429 networks [Aer04].

AFDX is based on 100 Mbit/s Ethernet over twisted-pair copper wires (IEEE 802.3u, 100BASE-TX). This means it is compatible with COTS Ethernet equipment on layers 1 and 2 (physical layer and link layer). Ethernet by itself is not suitable for real-time applications as its timing is not deterministic. Therefore AFDX imposes some constraints in order to achieve full determinism and hard real-time capability."

Ott [Ott07, Chap. 1.6.3] provides a substantially more detailed overview of AFDX.

3.3 Operating System Interface ARINC 653

Efkemann [Efk14, Chap. 2.2] then introduces to the operating system interface ARINC 653 of the IMA architecture:

"[. . .], an IMA module can host multiple avionics functions. The interface between avionics applications and the module's operating system conforms to a standardised API which is defined in the ARINC specification 653 [Aer05]."

Figure 1 shows the system architecture of an IMA module.

Efkemann [Efk14, Chap. 2.2.1] details the partitioning as follows:

"On the IMA platform, a partition is a fixed set of the module's resources to be used by an avionics application. In particular, each partition is assigned a portion of the module's memory. The operating system ensures that other partitions can neither modify nor access the memory of a partition, similar to memory protection in a UNIX-like operating system. Each partition also receives a fixed amount of CPU time. The operating system's scheduler ensures that no partition can spend CPU time allotted to another partition [Aer05, p. 13].

Two kinds of partitions reside on a module: Application partitions contain application code that makes up (part of) the implementation of an avionics function. System partitions on the other hand provide additional module-related services like data loading or health monitoring.

Inside a partition there can be multiple threads of execution, called *processes*. Similar to POSIX threads, all processes within a partition share the resources allocated to the partition. Each process has a priority. A process with a higher priority pre-empts any processes with a lower priority. ARINC 653 defines a set of states a process can be in (Dormant, Waiting, Ready, Running) as well as API functions for process creation and management [Aer05, p. 18–25]."

3.4 Distributed Modular Electronics (DME)

Distributed Modular Electronics (DME) is an extension of the IMA concept. It was developed in the SCARLETT research project (SCAlable & ReconfigurabLe elEctronics plaTforms and Tool). SCARLETT was a joint European research and technology project of airframers, large industrial companies, SMEs, and universities [SCA13].

According to Efkemann [Efk14, Chap. 2.5], "the DME concept aims at the separation of processing power from sensor/actuator interfaces, thereby reducing the number of different component types to a minimum. This also makes DME suitable for a wider range of aircraft types by giving system designers the possibility to scale the platform according to required hardware interfaces and computing power."

Figure 2 shows an example of a network of components. The Core Processing Module (CPM) components provide the computing power and host the avionics applications, but apart from AFDX they do not provide any I/O hardware interfaces. Instead, the Remote Data Concentrator (RDC) and Remote Power Controller (RPC) components provide the required number of sensor/actuator and bus interfaces. [Efk14, Chap. 2.5]

Efkemann continues: "The project also investigates ways of increasing fault tolerance through different reconfiguration capabilities, for example transferring avionics functions from defective modules to other, still operative modules. Finally, the design of a unified tool chain and development environment has led to improvements of the avionics software implementation process. [. . .]"

The ASHLEY project [ASH17] is a follow-up project to the SCARLETT project.

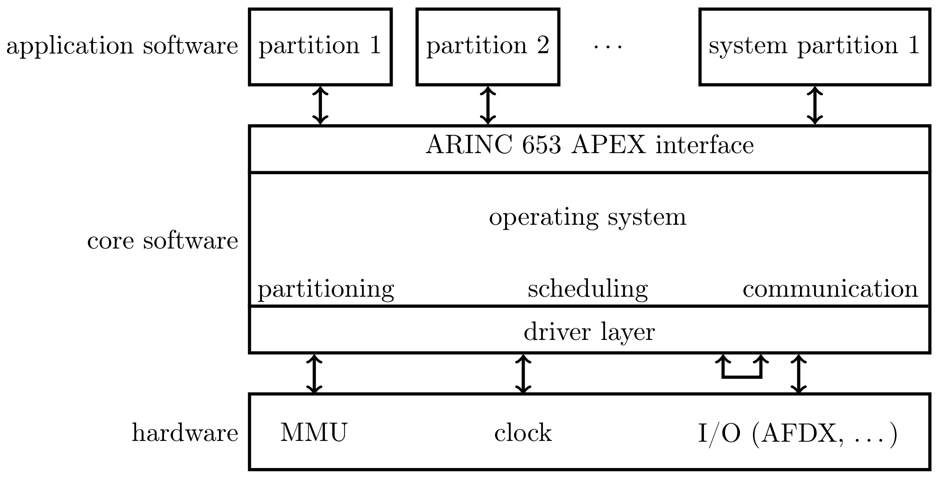

Figure 1: IMA module system architecture (after [Efk14, Fig. 2.2])

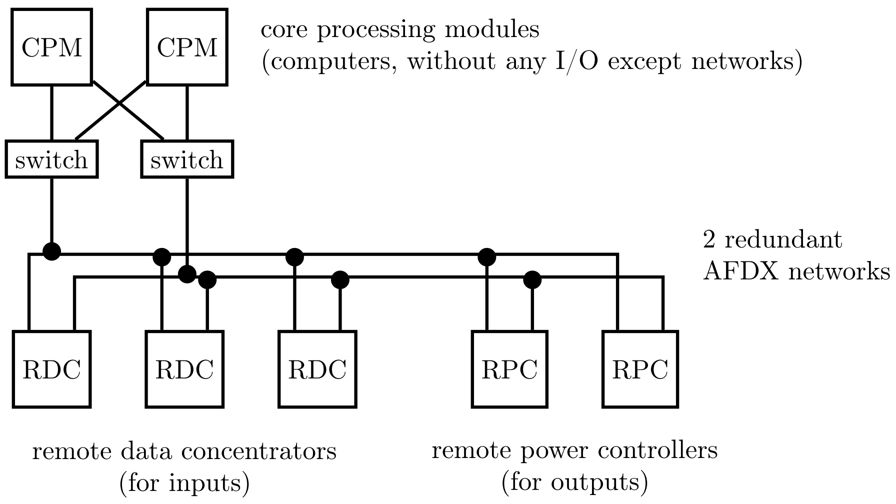

Figure 2: Distributed Modular Electronics (DME) architecture (after [Efk14, Fig. 2.3])

4 ADAPTION OF IMA FOR SPACE AVIONICS

4.1 Differences between the Aeronautical and the Space Domain

There are significant differences between the aeronautical and the space domain, with respect to time and space partitioning.

4.1.1 The Speed of Growth of Complexity

Windsor and Hjortnaes [WH09, Sec. IV.B] state that the space domain mastered integrating applications on a single CPU earlier than the aeronautical domain. This was achieved by validating everything at the highest integrity level. But this now becomes more and more expensive due to the increasing complexity of mission applications. The Central Flight Software (CFS) encompasses the Data Management (DMS) and the Attitude & Orbital Control (AOCS) functional chains. Recently, they are integrated on one computer instead of independent computers linked by a data bus ([WH09, Sec. V.A]). Similarly, the payload software consists of several components of differing criticality: command and control has a higher criticality than payload data processing. These payload functions co-exist on one or several on-board computers (OBCs) and are under the responsibility of one or several organizations ([WH09, Sec. V.B]).

The complexity problems hit the space domain later than the aeronautical domain. We suppose the reason is that the space domain must use slower and therefore less powerful computers due to the harsh radiation environment in space (compare Sect. 4.1.6 below). Therefore, the space domain only recently reached the critical region of complexity.

Windsor *et. al.* [WDD11] provide some concrete numbers on complexity for the Airbus A380: its IMA platform is composed of up to 30 IMA modules of 8 different types, hosting 21 avionics functions which were developed by 10 function suppliers. In contrast, the IMA-SP architecture envisioned by Windsor *et. al.* will have less computing nodes (e.g., central on board computer, payload computer, and intelligent sensors and actuators), connected to a less powerful network based on SpaceWire or MIL-STD-1553B), and hosting fewer functions developed by small numbers of suppliers.

4.1.2 Scale of Communication Demands

The space domain appears to have significantly smaller demands on communication. Even though the International Space Station (ISS) features a complexity similar to that of an airplane, a typical satellite or launcher is much simpler; compare the end of the previous section. Accordingly, only a few hardware nodes need to be connected, with less bandwith. Windsor *et. al.* [WDD11] even explicitly consider the communication bus optional for the IMA-SP platform, for the case of a single hardware node. We add that the necessary redundancy against hardware failures nevertheless demands at least two hardware nodes. Depending on the redundancy concept, there might be little demand for them to communicate, however.

4.1.3 Online/Offline Maintenance

All aeronautical software maintenance is performed offline, while the aircraft ist at the ground safely. In contrast, spacecraft operations require the system to be active ([WH09, Sec. VI.B].

Usually, a spacecraft is launched with the complete mission definition implemented in the flight software, from a pre-launch support mode to payload operations phase [WH09, Sect. V.D]. However, there is interest for adding new applications to extend the mission. This has been done in a few cases, but with extensive validation effort. And in the commercial satellite market, operators would like to have the in-flight capability to safely upload private payload applications to their spacecraft without the involvement of the manufacturer.

We add that the redundancy of multiple computer hardware may be used to take one of the computers offline in order to install a new software version. But this can be done for a short period of time only. And the new software version must be already validated fully. We have devised such an update process for the first in-space flight software update of the main computers of the European Columbus module which is part of the International Space Station [Bre08].

The IMA concept of the aeronautical domain requires substantial offline effort for reconfiguring communication paths. The virtual links (VLs) of the AFDX network are static. When the network routing shall be changed, each node affected must be accessed by a technician in order to apply this software update. This approach of IMA cannot be applied in the space domain, where physical access is impossible in nearly all cases.

4.1.4 Pronounced Mission Phases

Satellites have more pronounced mission phases than aircraft. Aircraft operate in different modes on ground, during ascent, cruise, and decent. Switches between these modes can occur rapidly. Long-lived satellites have longer-lasting and more predictable mission phases, such as ascent, orbit insertion, orbital payload operation, and deorbiting.

This offers the opportunity to reconfigure the computing resources between mission phases. At least, no computing time needs to be allocated to functions not active in the current mission phase. Since there is plenty of time after orbit insertion, the software even could be updated from ground, compare Sect. 4.1.3 above. See also [WDD11, page 8A6-4].

All of this does not apply to launchers, which are short-lived, of course.

4.1.5 Availability of a Hardware-Based Memory Protection Unit

In the space domain, only the most recent space qualified microprocessors (e.g., LEON2/3) have some kind of memory management unit (MMU) available [WH09, Sect. VI.E]. The Leon2 processor provides two memory write protection registers only. This is not sufficient if security is relevant, too. The Leon3 processor has a full MMU and also provides read-protection [Mas+10a]. (Sect. 4.4.1 below provides more details of the Leon processors.)

4.1.6 Radiation

In the space environment, the high level of hard radiation causes frequent malfunctions or even the permanent destruction of electronic circuits. This radiation could be shielded by a substantially massive casing only. Its launch weight usually prohibits this solution. The effect of radiation can be reduced by using electronic circuits with larger chip structures, too. However, larger chip structures also mean less functionality per chip and less speed. Therefore, less powerful computers can be used in space than on ground.

The atmosphere of the Earth shields most of this radiation, even for aircraft at high altitudes. Therefore, the restriction does not apply to the aeronautical domain. Accordingly, aeronautical hardware such as IMA modules cannot simply be taken and used in the space domain.

Higher computing power may be obtained in the space domain by giving up reliability and availability for some tasks, to a certain degree. For example, a radiation-hard, but slow computer can take care of the vital tasks of the spacecraft, while a much faster "number cruncher" computer can perform payload data processing, for example video encoding, even though it will crash a few times a day. In order to not affect the vital system adversely, such a number cruncher needs a bus interface with validated high dependability, only. However, such an approach with differentiated hardware does not match well the idea of interchangeable IMA hardware components.

4.2 The Original IMA-SP Project

Integrated Modular Avionics for Space (IMA-SP) was a project of the European Space Agency (ESA). It aimed at incorporating the benefits of time and space partitioning, based upon the aeronautical IMA concept, into the spacecraft avionics architecture. Windsor and Hjortnaes [WH09] motivate the benefits of IMA-SP in general, and they give an overview of the approach chosen, but yet without concrete experiences and without a defined, concrete architecture. Windsor *et. al.* [WDD11] provide such an architecture and some experimental applications of it.

The IMA-SP architecture is a two-layer architecture, consisting of a System Executive layer and an application layer [WH09]. The System Executive includes a software kernel responsible for partition scheduling and communication services as well as handling hardware signals. Memory partitioning is ensured either by a Memory Management Unit (MMU) or by a (simpler) Block Protection Unit (BPU). A BPU prohibits access to memory (at a minimum, write access) outside of a partition's defined memory areas. Partitions are scheduled on a fixed, cyclic basis. The order of partition activation is defined at configuration time using configuration tables. This provides a deterministic scheduling scheme. Tasks within a partition can be scheduled statically or dynamically. Temporal and spatial partitioning therefore ensures each partition uninterrupted access to common resources and non-interference during their assigned time period.

IMA-SP defines the role of the system integrator explicitly [WH09, Sect. III.D]. They are responsible for the system design including the detailed on-board resource allocation; and they are responsible for the final integration and configuration of the components. Furthermore, there are the role of the platform supplier and the role of the application supplier.

A communication bus is optional for the IMA-SP platform, for the case of a single hardware node [WDD11]. This reflects the significantly smaller demands on communication in the space domain compared to the aeronautical domain, see Sect. 4.1.2 above.

The IMA-SP approach customizes its architecture quite specifically to the requirements of the space domain, [WDD11]. We think that these "user requirements" must make the resulting architecture rather specific for a narrow application area. The IMA-SP project apparently did not do a generalization step by identifying common requirements of the aeronautical domain and the space domain first, before adding the space specific requirements. Instead, the project put an emphasis on preserving long-proven ideas, approaches, and even hardware from the space domain. Therefore, it has become less visible which conceptual changes are necessary for the transfer of IMA from the aeronautical domain to the space domain, and what are just customizations to a specific application area. The aeronautical domain dared a more radical change of its ways when introducing IMA, compared to what the IMA-SP project is prepared to do.

We even think that the approach is tailored more to satellites than to launchers. Launchers have no opportunity to do flight software maintenance, and often there is no time for recovery from a safe mode.

The services of the TSP abstraction layer were derived from the ARINC 653 [Aer05] API of the IMA architecture [WDD11]. In this, re-use of existing space concepts, which are similar to ARINC 653, took precedence over the (literal) incorporation of the ARINC 653 specification. Windsor *et. al.* describe the similarities and differences in detail [WDD11].

Windsor *et. al.* [WDD11] report on three use cases which the IMA-SP study planned to investigate. Windsor presents some results in a talk at ADCSS 2012 [Win12]. Silva *et. al.* [SCS12] report on the development of an input/output component for IMA-SP, by GMV, Portugal. The original IMA-SP project ended in December 2012.

4.3 IMA-SP Follow-Up Projects

Hardy *et. al.* [HHC14] report on a follow-up assessment study on partitioning and maintenance of flight software in IMA-SP.

The IMA-SP System Design Toolkit project is a follow-up project to the original IMA-SP project. Hann *et. al.* [Han+15] describe its first phases (2014 to 2015).

Hann *et. al.* [Han+16] describe preparation activities for the future IMA separation kernel qualification. Also for IMA separation kernel qualification, Butterfield *et. al.* [But+16] perform a case study on the formal verification of small aspects of an IMA separation kernel.

SAVOIR (space avionics open interface architecture) is an initiative by the European Space Agency (ESA) which aims at improving the ways in which the European space community builds avionics sub-systems. It it geared towards satellites, thus excluding launchers. The initiative defined a reference avionics architecture for spacecraft platform hardware and software in general. The reference architecture either uses a "classic" execution platform or an execution platform providing time and space partitioning. SAVOIR is organized in specialized working groups. Two of them are SAVOIR-FAIRE on the software reference architecture in general and SAVOIR-IMA on the TSP based software reference architecture. [Hjo14] Panunzio *et. al.* [PL16] provide their view regarding the way forward for the definition of the SAVOIR communication architecture. Sandin reports on the new generation SAVOIR on-board computer [DAS17].

4.4 Virtualization Solutions Suitable for Space Avionics

In this section, we survey virtualization solutions suitable for space avionics. They are the bare-metal hypervisor XtratuM and the partition management kernel AIR.

4.4.1 XtratuM: a Hypervisor for Safety-Critical Embedded Systems

XtratuM is a bare-metal hypervisor which implements para-virtualization and dedicated device techniques [Pei+10; Mas+09; Cre+09]. It was designed to achieve time and space partitioning for safety-critical embedded systems.

We already presented the notions of *virtualization in general*, of a *bare-metal hypervisor*, and of *para-virtualization* in Sect. 2.3.2 above.

Figure 3 shows an overview of the XtratuM architecture [Pei+10]. XtratuM executes in the supervisor mode of the processor. The applications execute in the user mode, each in its own partition. The hypervisor XtratuM applies a static scheduling scheme to grant execution time of the processor to the partitions. This achieves the separation of the partitions in the time domain. Each partition gets its statically determined share of the execution time. No user partition can overrun or change the schedule. Timer interrupts are caught and handled by the hypervisor.

Similarly, the hypervisor XtratuM allocates areas of memory to the partitions. It catches any illegal access to memory addresses outside a partition's memory space. To be precise, this is true only for processors providing suitable hardware support. The Leon2 processor does not have a memory management unit (MMU), which could translate virtual memory addresses into physical memory addresses. The Leon2 provides two memory write protection registers only. Therefore, XtratuM can enforce write protection among partitions, but not read protection. This is sufficient in order to meet safety requirements, but it is not sufficient to enforce security against malicious software in a partition. On processors providing a MMU, such as the Leon3 processor, XtratuM provides read protection, too [Mas+10a]. The Leon processors are designed for and used in a space environment, in particular with an increased level of radiation.

A peripheral device can be associated to a specific partition. In this case, no other partition may access this peripheral device. This is similar to memory protection.

Some partitions may be special, these are called system partitions or supervisor partitions, in contrast to the user partitions. The system partitions are allowed to manage the other partitions, for example by stopping and resuming them via calls to the hypervisor. However, since these partitions run in the user mode of the processor, too, they cannot break the time and space isolation described above.

The XtratuM hypervisor provides a hardware abstraction to the partitions similar to the aeronautical ARINC 653 standard [Aer05] API of the IMA architecture (compare Sect. 3.3). However, these two interfaces have several differences. The most notable difference is that communication in the IMA architecture is based on the fast AFDX data network (compare Sect. 3.2), while XtratuM does not prescribe any particular network technology. There even can be no inter-computer network at all. This is probably due to the difference between the aircraft and the spacecraft domain, that communication demands are often lower in the latter domain, compare Sect. 4.1.2 above.

Each partition appears as a normal, dedicated computer to the software running in it. Accordingly, a partition can contain a bare application (no operating system at all, just an infinite loop), a general-purpose operating system, or a real-time operating system. In case there is some operating system, an application in a partition may have several processes/threads, as usual. The

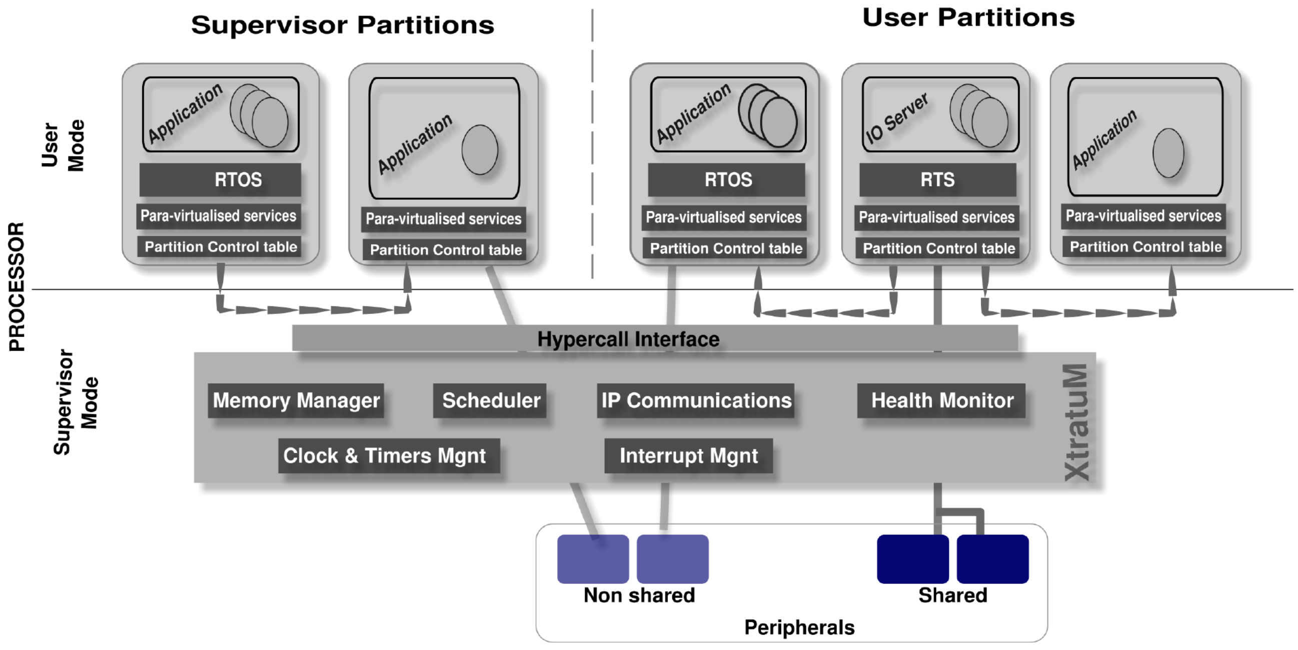

Figure 3: The XtratuM architecture, taken from [Pei+10, Fig. 1].

para-virtualization approach requires an adaption of the software inside the partitions, however: some privileged machine instructions in the lower layer of the operating system (or in the bare application) must be substituted by calls to the hypervisor, as discussed in Sect. 2.3.2.

Operating systems that have been ported to XtratuM include LithOS and RTEMS (see Sect. 4.4.2 on RTEMS below), and also Linux, PaRTiKle, and ORK+ [FEN+13]. The real-time operating system LithOS [Mas+10b] was designed to provide an ARINC 653 inspired API. LithOS adds multi-process support, communication between processes, and a process scheduler to the services provided by XtratuM. However, there is still no mandatory AFDX network with LithOS. Similarly, PaRTiKle is an open source real-time kernel for embedded systems, distributed under the terms of the GNU Public License; PaRTiKle has been initially developed by the University of Valencia, Spain [FEN+13]. ORK+ (Open Ravenscar Kernel) is a small, high performance real-time kernel that provides restricted tasking support for Ada programs [FEN+13].

The processors on which XtratuM has been implemented include the Intel x86 processor family, supporting multiprocessors [Mas+09; Cre+09], the Leon2 processor [Pei+10], the Leon3 processor [Mas+10a], the Leon4 processor [Cre+14a; MCC12], and the ARM Cortex R4 [Cre+14a]. The MultiPARTES project adapted XtratuM to deal with heterogeneous multicore architectures, see Sect. 4.6 below.

XtratuM is open software distributed under the Gnu Public Licence version 3. Some ancillary tools are sold under a proprietary licence by FentISS [Fen18], a spin-off of the University of Valencia, Spain.

The configuration of a specific software image must be described in a central configuration file. XtratuM's system integration tools compile this configuration file together with the application software images into the system software image. This image then is loaded onto the processor an run.

The central XtratuM configuration file is in XML format and can be written by hand. Alternatively, the configuration file can also be generated using the Xoncrete tool. Xoncrete is a graphical tool to assist the system designer when configuring the resources (memory, communication ports, devices, processor time, etc.) allocated to each partition. It has two parts: a resource editor and a scheduling analysis tool [Bro+10]. The scheduling analysis tool provides support for the complexities of hierarchical scheduling (i.e., scheduling the partitions and scheduling tasks inside the partitions) [Bro+10; Rip+10].

4.4.2 AIR: A Partition Management Kernel Based on RTEMS

The AIR and AIR-II projects developed a partition management kernel based on the RTEMS operating system kernel.

RTEMS RTEMS (Real-Time Executive for Multiprocessor Systems) is a real-time operating system kernel.

Originally designed for military applications, it is now used in a wide area of application domains, including the space domain, in particular. It is open software distributed under a license close to the GNU General Public License. RTEMS provides multi-tasking, inter-task communication, different kinds of scheduling, and support for homogeneous and heterogeneous multiprocessor systems. [RTE16]

RTEMS is not a partitioning kernel. Thus, it does not support time and space partitioning. However, RTEMS can be and has been used as a real-time operating system inside a partition. (For example, Ripoll *et. al.* report on a use of RTEMS inside XtratuM [Rip+10], and Hardy *et. al.* [HHC14] report on another such use of RTEMS in XtratuM.)

AIR The European Space Agency studies AIR and AIR-II (**A**RINC **I**nterface in **R**TOS – Industrial Initiative) developed several components for time and space partitioning [RF07b; RF07a]. They comprise [Sch11]

- a partition management kernel,
- support libraries, drivers, etc.
- operating system APIs to be used inside of a partition,
- a configuration and compilation tool chain, and
- analysis tools.

Figure 4 shows the AIR architecture. It allows to sepate applications into different partitions, each with their own memory space and own time budget. Inter-partition communication is by queueing ports and sampling ports. All this is similar to XtratuM, see Sect. 4.4.1 above. But there are differences, too.

The preferred operating system to be used inside a partition ("personality") is RTEMS, possibly extended by the ARINC 653 API ("APEX"). The partition management kernel internally uses RTEMS as a hardware abstraction layer, too. Therefore, there are close links from AIR to RTEMS, even though any operating system can be run inside a partition, or even a bare application. [Sch11]

The AIR project used RTEMS version 4.6.6 [RF07a].

Only the so-called "system" partitions may access hardware devices and thus perform input/output. The corresponding drivers run in kernel mode, but are scheduled as "co-partitions". Co-partitions share execution windows with their client partitions, up to a pre-defined percentage ("sharing quota"), iff critical tasks have terminated ("sharing barrier"). Efficient inter-partition communication via shared memory is possible, too. [Sch11]

The partition management kernel is a micro-kernel. There are no kernel threads. Instead, the concurrency of drivers is implemented through the co-partitions, which are scheduled with their client partitions. [Sch11]

Silva *et. al.* [SCS12, Sect. 3.1] note that the co-partition approach is more efficient than having regular partitions for input/output, but that this comes not without a cost. A system that allows a partition to be pre-empted before the end of its execution window in order to perform I/O operations is more difficult to analyse, qualify, and in perspective, to fully guarantee as predictable.

The studies were performed by GMV, Portugal, together with the University of Lisbon, Portugal, and Thales-Alenia Space. The initial target processor architecture was Sparc-Leon. [Sch11]

Santos *et. al.* [San+08] describe in detail the implementation of an ARINC 653 API on an underlying POSIX layer. The AIR-II consortium decided not to implement an entirly new in-partition operating system to provide ARINC 653 API services to hosted applications, but instead to build the ARINC 653 API on top of available real-time operating systems.

4.5 Separation Kernels Suitable for Space Avionics

In this section, we survey separation kernels suitable for space avionics. These comprise PikeOS, VxWorks 653, and LynxSecure.

4.5.1 PikeOS

PikeOS is a separation kernel for real-time systems in safety-relevant and security-relevant domains, such as the aeronautical domain and the space domain [SYS18]. Alternatively, it also can be viewed as a virtualization solution plus an optional guest operating system. Figure 5 shows the architecture of PikeOS. PikeOS is a commercial product by Sysgo, Germany.

According to the product overview [SYS18], the PikeOS Hypervisor runs on x86 as well as ARM, PowerPC, SPARC, or V8/LEON and can be adapted to other CPU types. The virtualization concept supports multi-core architectures. PikeOS is completely developed according to safety standards such as DO-178B/C, IEC 61508, EN 50128, ISO 26262 or IEC 62304. The available guest operating systems, runtime enviroments and APIs are: PikeOS, Linux, Android, ARINC 653, AUTOSAR, RTEMS, legacy RTOS, POSIX, Realtime Java, ADA, and others. Optimized implementations such as ARINC 664 (AFDX) and CAN are available for PikeOS Native partitions. There is a development tool chain called CODEO. It is an Eclipse-base IDE with configuration tools, remote debugging with operating system

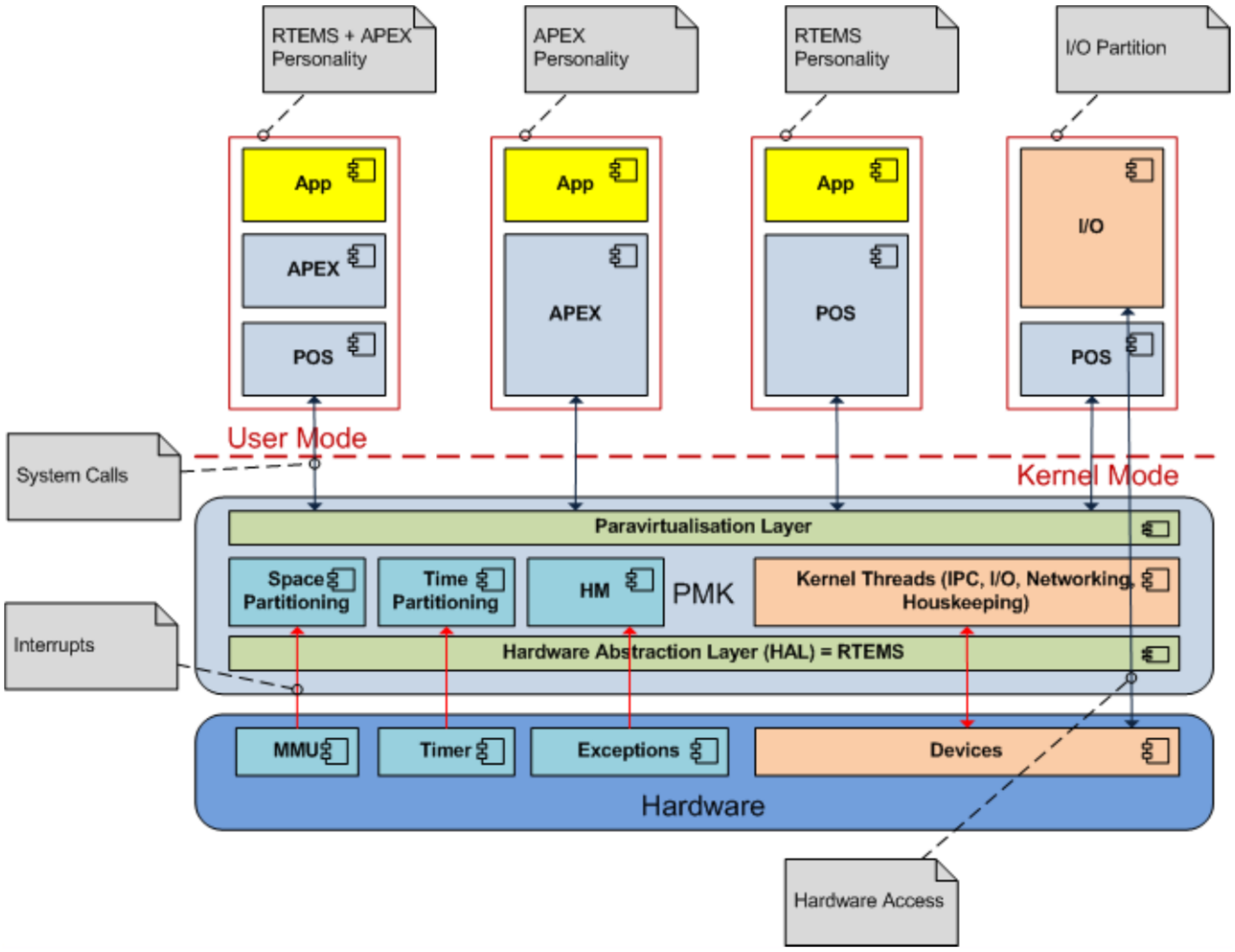

Figure 4: The AIR architecture (taken from [Sch11]).

awareness, target monitoring, remote application deployment, and timing analysis tools.

4.5.2 VxWorks 653

VxWorks 653 is a real-time operating system for safety-critical domains, such as the aeronautical domain, supporting IMA standards, in particular the ARINC 653 standard [PK15]. VxWorks 653 is that member of the VxWorks real-time operating system family that provides a separation kernel. Figure 6 shows the architecture of VxWorks 653. VxWorks 653 is a commercial product by Wind River, which is a subsidiary of Intel.

According to a white paper by the manufacturer [PK15], there is a *module operating system* providing global resource management, scheduling, and health monitoring. There is also a VxWorks *partition operating system* providing scheduling and resource management within a partition. There is no mention of running bare-metal applications without any operating system inside a partition. Accordingly, a guest operating system inside a partition appears to need adaptions to run under VxWorks 953. VxWorks 653 provides an option for priority preemptive scheduling of partitions. This permits *slack stealing* by allowing designated partitions to consume what would otherwise be idle time in the defined ARINC schedule. The VxWorks 653 3.0 Multi-core Edition supports multi-core processors. However, certification of this is still under review by authorities in both the FAA and EASA. The operating system comes with a tool chain including an Eclipse-based workbench, the simulator Simics, and further development, system configuration, and debugging tools. The white paper also mentions some security related mechanism of VxWorks 953, but no comprehensive security concept. VxWorks 653 is used for many avionics systems and safety-critical applications, including systems of the Boeing 787 Dreamliner and of the Airbus A330.

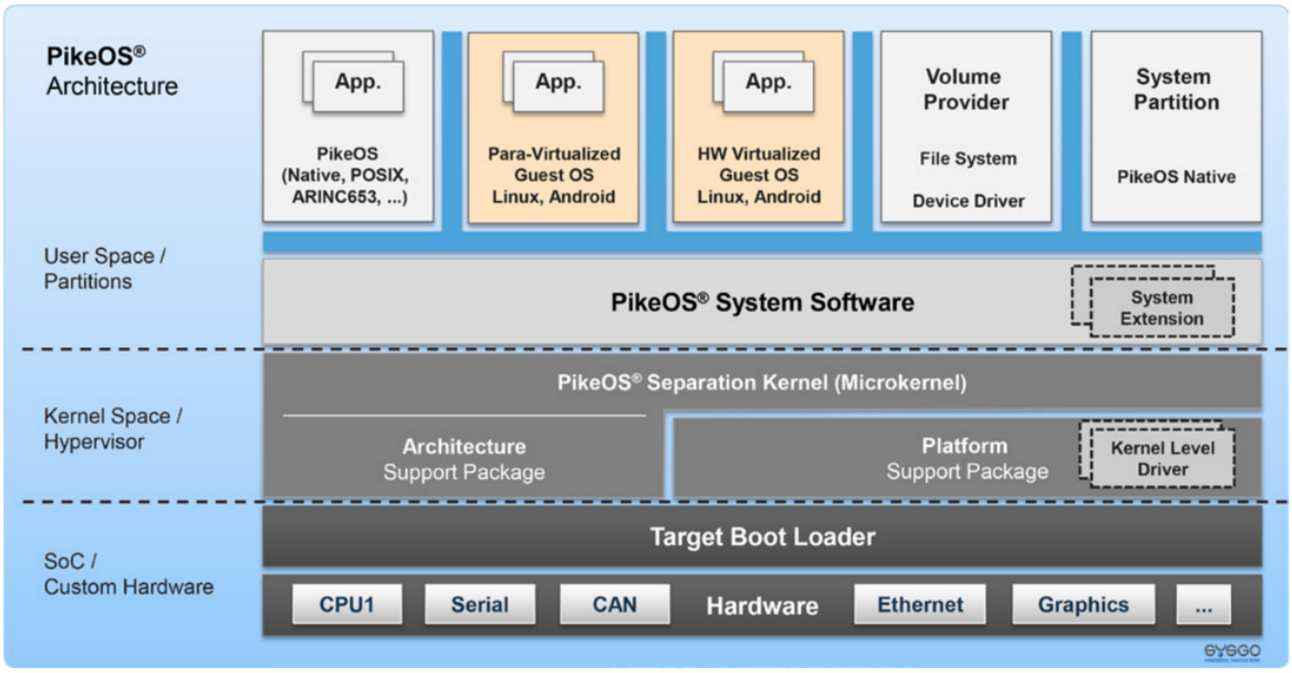

Figure 5: PikeOS architecture (taken from [SYS18, Fig. 1])

4.5.3 LynxSecure

LynxSecure is a real-time operating system for security-relevant systems with a separation kernel [Lyn17]. Its emphasis is mainly on security and only to a lesser extent on safety. It supports multi-core processing. Figure 7 shows the architecture of LynxSecure. LynxSecure is a commercial product by Lynx Software Technologies, CA, USA.

4.6 Extensions for Multi-Core Processors

The MultiPARTES project (Multi-cores Partitioning for Trusted Embedded Systems) adapted the hypervisor XtratuM [Cre+09] to deal with heterogeneous multicore architectures [TCA13]. Besides the adaption of XtratuM, the project also defined a development methodology and provided supporting tools.

The hardware platform consists of two different systems: a dual core x86 based processor (Atom Core Duo at 1.7 GHz) and an FPGA with several synthesized LEON3 processors. The x86 subsystem provides comparably high computation capabilities, and the LEON3 processors provide a hardware base for time-predictable computations.

Therefore, the challenges of multi-core CPUs to time partitioning (compare Sect. 5 below) have been responded to by simply using several independent LEON3 CPUs for the time critical computations. However, these LEON3 CPUs have been integrated onto a single FPGA chip (and under a single hypervisor) at least, thus reducing weight and other resource demands.

Crespo *et. al.* [Cre+14b] describe the multi-core version of the hypervisor XtratuM, and they report on its performance. However, they measure the speed of the x86 cores only; and they don't investigate the worst-case impact one core might have on the other through the shared cache. This would be required to predict the worst-case execution time safely. Therefore, the x86 cores don't appear to be intended for hard real-time computations.

The DASIA 2017 conference dedicated an entire panel session and several papers to multicore processors [DAS17].

5 RESEARCH CHALLENGES

In this section, we collect some open problems in the area of time and space partitioning. They deserve further research.

5.1 CPU-Related Challenges for Time Partitioning

Current CPUs pose challenges to time partitioning. They are, in particular, the intricacies of multi-core CPUs and of direct memory access.

Current multi-core CPUs make it difficult to achieve

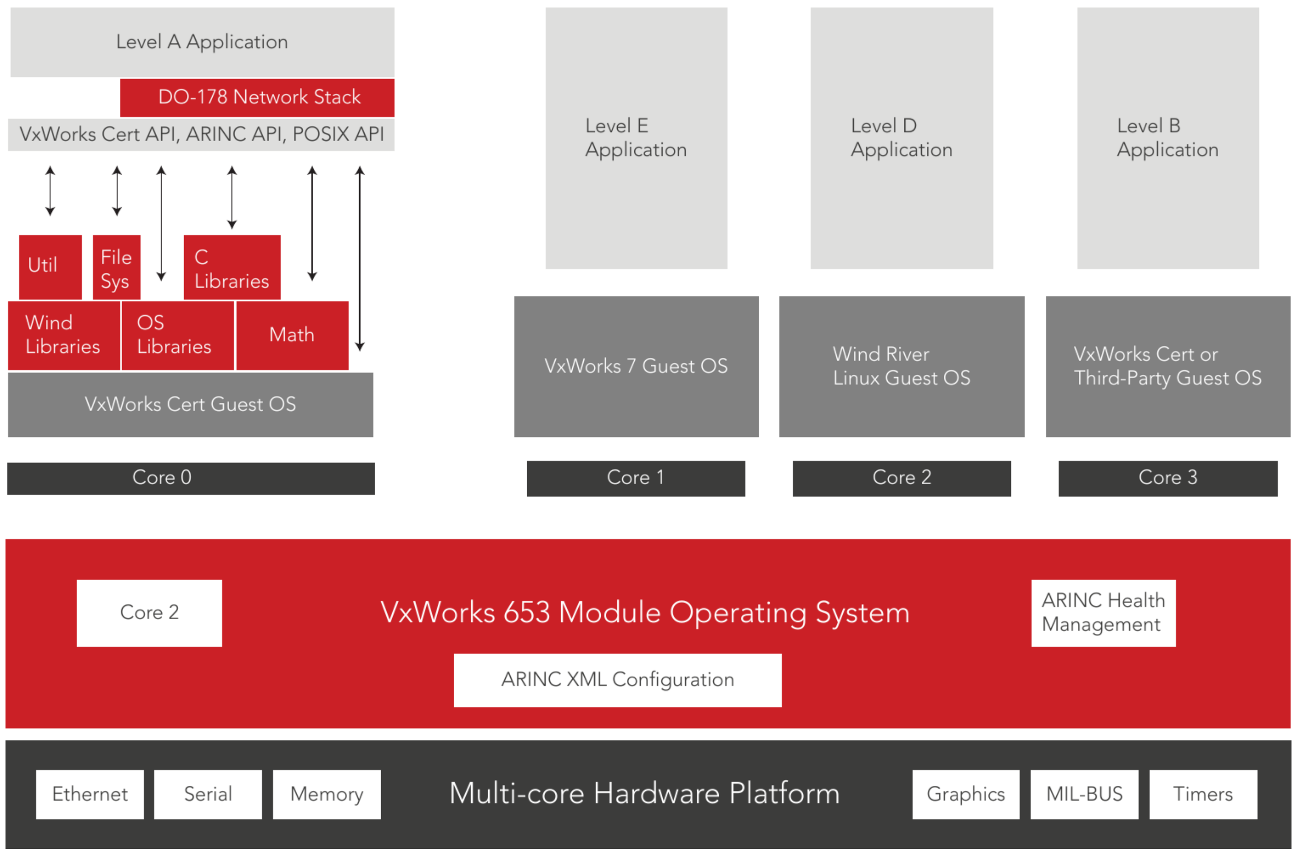

Figure 6: VxWorks 653 architecture (taken from [PK15, Fig. 2])

true time partitioning. A cache shared between CPU cores causes a dependency of the worst-case execution time of a piece of code running on one core on the behaviour of code running on another core. Several research projects already investigate this subject. Some of them are listed by Crespo *et. al.* [Cre+14a, Sect. 2.3] (even though none of them primarily addresses the space domain).

A DMA controller can slow down the CPU by contending for the memory bus. Therefore, a non-real-time partition can affect a real-time partition adversely in this way. As Crespo *et. al.* [Cre+14b] demonstrate, even CPU cores can contend for the memory bus in a similar way.

5.2 Challenges for Real-Time Property Proofs

Proving real-time properties poses many challenges, including those from processor architecture, virtualization, and distributed computing.

Conventional processor architectures aim to optimize the average-case performance. However, when validating real-time properties, the worst-case performance matters. Consequently, hard real-time tasks require an entirely different processor architecture design.

Timing anomalies occur in complex processor architectures, and they complicate real-time property proofs greatly. Selecting or designing a processor architecture for space use that is not susceptible to timing anomalies may prove useful for achieving acceptable real-time performance and validating it.

Virtualization poses additional challenges when performing a real-time property proof. Virtualization cannot hide that the machine instructions are not executed evenly in the time sense anymore. Any real-time property proof must account for this. If the hypervisor employs a static cyclic scheduling, this can be done.

The interplay between the scheduling of a hypervisor or of a separation kernel on the one hand with the scheduling of a communication bus on the other hand poses its own difficulties, with respect to the according real-time property proofs. See, e.g., Silva *et. al.* [SCS12].

Distributed Modular Electronics (DME, compare Sect. 3.4) and other approaches which combine time and space partitioning with distribution add corresponding challenges to proving real-time properties. In general, there will not even exist a global scheduling cycle, such

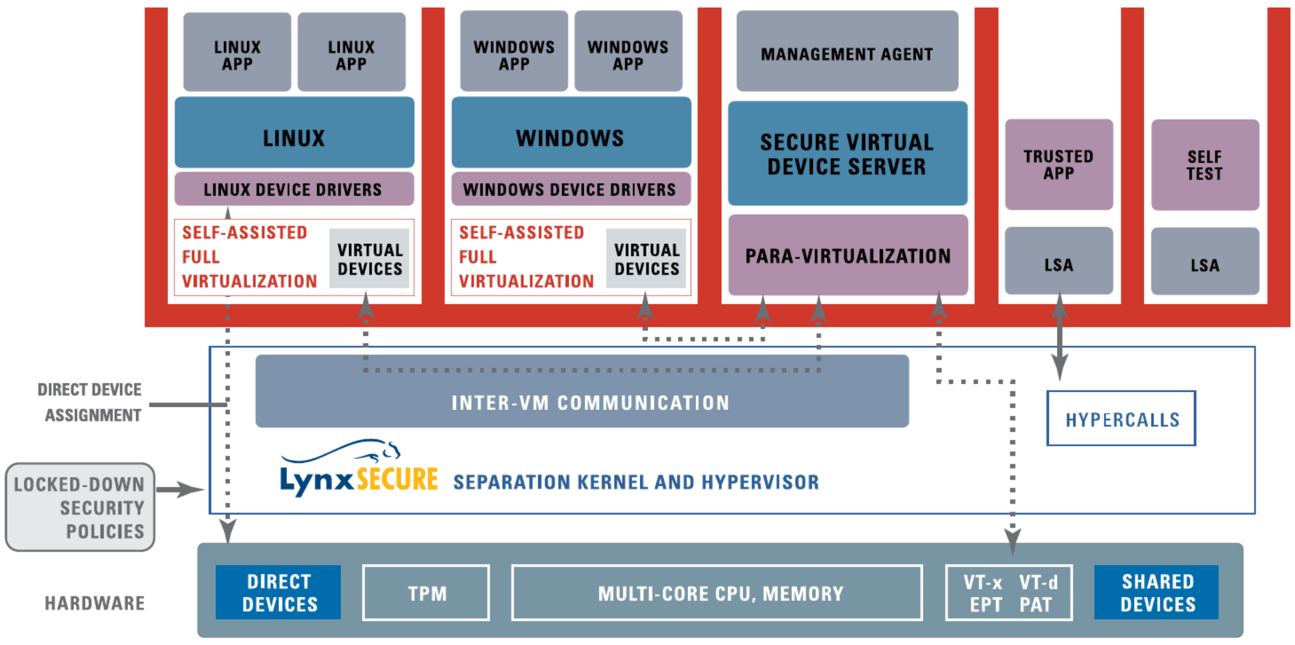

Figure 7: LynxSecure architecture (taken from [Lyn16, Fig. 3])

that the local scheduling latencies will add up with their worst-case values. Brocal *et. al.* [Bro+10] and Ripoll *et. al.* [Rip+10] present work on a special case, in the context of the Xoncrete tool for the XtratuM hypervisor.

We performed a case study on grid computing using space hardware [B+14]. However, grid computing intrinsically is a best effort approach, and therefore probably is no viable way for providing high-speed real-time computing.

6 SUMMARY

We presented a survey of the current state of the reseach on time and space partitioning for space avionics. There is already a body of existing work, it has been presented in Sect. 4. But substantial research challenges remain to be tackled, as shown in Sect. 5.

REFERENCES

[Aer04] Aeronautical Radio, Inc. *Digital Information Transfer System (DITS), Part 1, Functional Description, Electrical Interface, Label Assignments and Word Formats. ARINC Specification 429P1-17 Mark 33.* May 2004.

[Aer05] Aeronautical Radio, Inc. *Avionics Application Software Standard Interface, Part 1 – Required Services. ARINC Specification 653P1-2.* Dec. 2005.

[Aer09] Aeronautical Radio, Inc. *Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet. ARINC Specification 664P7-1.* Sept. 2009.

[Alv+05] Jim Alves-Foss et al. 'The MILS Architecture for High-Assurance Embedded Systems'. In: *Intl. Journal of Embedded Systems* 2 (3-4 Feb. 2005), pp. 239–247.

[ASH17] *ASHLEY project website.* ASHLEY Consortium. 2017. URL: `http://www.ashleyproject.eu/` (visited on 20/06/2018).

[Avi+04] Algirdas Avižienis et al. 'Basic Concepts and Taxonomy of Dependable and Secure Computing'. In: *IEEE Trans. on Dependable and Secure Computing* 1.1 (Jan.–Mar. 2004).

[B+14] Andreas Bergmeier, Peer Kampa, Jan Bredereke et al. *Grid-Computing in der Raumfahrt.* German. Project report. Univ. of Applied Sciences Bremen, Germany, 2nd Feb. 2014.

[Bre08] Jan Bredereke. *Flight SW Transition to Cycle 11.* ESO-IT-TN-0116. Version 2. Internal technical note. Astrium GmbH. 26th Aug. 2008.

[Bre17] Jan Bredereke. *A Survey of Time and Space Partitioning for Space Avionics.* Tech. rep. Version 1.1. City University of Applied Sciences Bremen, 16th Feb. 2017. urn:nbn:de:gbv:46-00105751-11. URL: `http://homepages.hs-bremen.de/~jbredereke/downloads/bredereke-tsp-space-avionics-tr-2017.pdf`.

[Bro+10] Vicent Brocal et al. 'Xoncrete: a scheduling tool for partitioned real-time systems'. In: *5th Intl. Conf. and Exhibition on Embedded Realtime Software and Systems, ERTSS-2010.* (Toulouse, France). Ed. by Gérard Ladier and Jean-Luc Maté. 19th–21st May 2010. URL: `http://www.fentiss.com/documents/xoncrete_overview.pdf`.

[But+16] Andrew Butterfield et al. 'Towards Formal Verification of Interrupts and Hypercalls'. In: *Proc. of DASIA 2016 Data Systems In Aerospace.* (Tallinn, Estonia, 10th–12th May 2016). Ed. by L. Ouwehand. SP-736. ESA Spacebooks Online, Aug. 2016.

URL: http://adsabs.harvard.edu/abs/2016ESASP.736E..22B.

[Cre+09] A. Crespo et al. ‘XtratuM: an open source hypervisor for TSP embedded systems in aerospace’. In: *Proc. of DASIA 2009 Data Systems In Aerospace.* (Istanbul, Turkey, 26th–29th May 2009). Ed. by L. Ouwehand. SP-669. ESA Spacebooks Online, Aug. 2009. URL: http://www.fentiss.com/documents/dasia09.pdf.

[Cre+14a] Alfons Crespo et al. ‘Mixed Criticality in Control Systems’. In: *Preprints of the 19th World Congress of the International Federation of Automatic Control.* (Cape Town, South Africa). 24th–29th Aug. 2014.

[Cre+14b] A. Crespo et al. ‘Multicore partitioned systems based on hypervisor’. In: *19th IFAC World Congress.* (Cape Town, South Africa, 24th–29th Aug. 2014). Ed. by E. Boje et al. Vol. 47. Elsevier, Dec. 2014, pp. 12293–12298.

[DAS17] *Proc. of DASIA 2017 Data Systems In Aerospace.* (Gothenburg, Sweden, 30th May–1st June 2017). Still in preparation. Eurospace.

[Efk14] Christof Efkemann. ‘A Framework for Model-based Testing of Integrated Modular Avionics’. PhD thesis. Univ. of Bremen, Germany, 6th Aug. 2014. urn:nbn:de:gbv:46-00104131-10.

[FEN+13] FENTISS et al. *Tool chain implementation.* Version v1.0. MultiPARTES project. Feb. 2013. URL: https://alfresco.dit.upm.es/multipartes/public/MPT-D4.3-V10-final.pdf.

[Fen18] *Homepage of FentISS.* Fent Innovative Software Solutions. 2018. URL: http://www.fentiss.com/ (visited on 20/06/2018).

[Fil03] Bill Filmer. ‘Open systems avionics architectures considerations’. In: *Aerospace and Electronic Systems Magazine, IEEE* 18 (9 Sept. 2003), pp. 3–10. DOI: 10.1109/MAES.2003.1232153.

[Han+15] Mark Hann et al. ‘System Design Toolkit for Integrated Modular Avionics for Space’. In: *Proc. of DASIA 2015 Data Systems in Aerospace.* (Barcelona, Spain, 19th–21st May 2015). ESA Special Publication ESA SP-732. Sept. 2015.

[Han+16] Mark Hann et al. ‘Qualification Strategy and Plan for Integrated Modular Avionics for Space Separation Kernel’. In: *Proc. of DASIA 2016 Data Systems In Aerospace.* (Tallinn, Estonia, 10th–12th May 2016). Ed. by L. Ouwehand. SP-736. ESA Spacebooks Online, Aug. 2016. URL: http://adsabs.harvard.edu/abs/2016ESASP.736E..23H.

[HHC14] Johan Hardy, Martin Hiller and Philippe Creten. ‘Partitioning and Maintenance of Flight Software in Integrated Modular Avionics for Space’. In: *TEC-ED & TEC-SW Final Presentation Days 2014.* ESA ESTEC. 21st–22nd May 2014. URL: https://indico.esa.int/indico/event/57/contribution/4/material/slides/0.pdf.

[Hjo14] Kjeld Hjortnaes. ‘Introduction to SAVOIR’. In: *8th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2014.* (Noordwijk, The Netherlands). Ed. by Kjeld Hjortnaes, Alain Benoit and Philippe Armbruster. 27th–29th Oct. 2014. URL: https://indico.esa.int/indico/event/53/session/1/contribution/3/material/1/.

[Lyn16] *LynxSECURE. Software security driven by an embedded hypervisor.* Product Datasheet. Last version with figure on architecture. Lynx Software Technologies, Inc. San Jose, CA, USA, 2016. URL: http://www.lynx.com/pdf/LynxSecureDatasheetFinal.pdf (visited on 09/06/2016).

[Lyn17] *LynxSECURE. Software security driven by an embedded hypervisor.* Product Datasheet. Lynx Software Technologies, Inc. San Jose, CA, USA, 2017. URL: http://www.lynx.com/pdf/LynxSecureDatasheetFinal.pdf (visited on 20/06/2018).

[Mas+09] M. Masmano et al. ‘XtratuM: a Hypervisor for Safety Critical Embedded Systems’. In: *11th Real-Time Linux Workshop.* (Dresden, Germany). 2009.

[Mas+10a] Miguel Masmano et al. ‘XtratuM for LEON3: an Open Source Hypervisor for High Integrity Systems’. In: *5th Intl. Conf. and Exhibition on Embedded Realtime Software and Systems, ERTSS-2010.* (Toulouse, France). Ed. by Gérard Ladier and Jean-Luc Maté. 19th–21st May 2010. URL: www.fentiss.com/documents/xtratum-leon3.pdf.

[Mas+10b] M. Masmano et al. ‘LithOS: a ARINC-653 guest operating [sic!] for XtratuM’. In: *12th Real-Time Linux Workshop.* (Nairobi, Kenia). 27th Oct. 2010. URL: http://www.xtratum.org/files/lithos-2010.pdf.

[MCC12] Miguel Masmano, Alfons Crespo and Javier Coronel. *XtratuM Hypervisor for LEON4. Volume 2: XtratuM User Manual.* Polytechnical University of Valencia, Spain. Nov. 2012. URL: http://microelectronics.esa.int/gr740/xm-4-usermanual-047d.pdf (visited on 20/06/2018).

[Ott07] Aliki Ott. ‘System Testing in the Avionics Domain’. PhD thesis. Univ. of Bremen, Germany, Oct. 2007. urn:nbn:de:gbv:46-diss000108814.

[Pei+10] S. Peiró et al. ‘Partitioned Embedded Architecture based on Hypervisor: the XtratuM approach’. In: *8th European Dependable Computing Conference, EDCC-8 2010.* (Valencia, Spain). IEEE Computer Society. 28th–30th Apr. 2010.

[PK15] Paul Parkinson and Larry Kinnan. *Safety-Critical Software Development for Integrated Modular Avionics.* Rev. 10/2015. White Paper. Wind River Systems, Inc. Oct. 2015. URL: http://www.windriver.com/whitepapers/safety-critical-software-development-for-integrated-modular-avionics/wp-safety-critical-software-development-for-integrated-modular-avionics.pdf (visited on 20/06/2018).

[PL16] Marco Panunzio and Patricia Lopez-Cueva. ‘The SAVOIR Communication Architecture: Current Results and Way Forward’. In: *Proc. of DASIA 2016 Data Systems In Aerospace.* (Tallinn, Estonia, 10th–12th May 2016). Ed. by L. Ouwehand. SP-736. ESA Spacebooks Online, Aug. 2016. URL: http://adsabs.harvard.edu/abs/2016ESASP.736E..44P.

[RF07a] José Rufino and Sérgio Filipe. *AIR Project Final Report.* Tech. rep. DI-FCUL TR–07–35. Comp. Sce. Dept. of the University of Lisbon, Portugal, Dec. 2007. URL: http://air.di.fc.ul.pt/air/downloads/07-35.pdf.

[RF07b] José Rufino and Sérgio Filipe. *AIR Project Summary Report.* Tech. rep. DI-FCUL TR–07–36. Comp. Sce. Dept. of the University of Lisbon, Portugal, Dec. 2007. URL: `air.di.fc.ul.pt/air/downloads/07-36.pdf`.

[Rip+10] I. Ripoll et al. 'Configuration and Scheduling Tools for TSP Systems Based on XtratuM'. In: *Proc. of DASIA 2010 Data Systems in Aerospace.* (Budapest, Hungary). 1st–4th June 2010. URL: `http://www.xtratum.org/files/dasia2010.pdf`.

[RTE16] *RTEMS C User's Guide.* Version RTEMS 4.10.2. 2016. URL: `https://docs.rtems.org/releases/rtemsdocs-4.10.2/share/rtems/html/c_user/index.html` (visited on 20/06/2018).

[Rus81] John Rushby. 'The Design and Verification of Secure Systems'. Reprint of a paper presented at the 8th ACM Symposium on Operating System Principles, Pacific Grove, CA, USA, 14–16 Dec. 1981. In: *ACM Operating Systems Review* 15.5 (1981), pp. 12–21.

[San+08] Sérgio Santos et al. 'A Portable ARINC 653 Standard Interface'. In: *27th Digital Avionics Systems Conference, DASC 2008.* (St. Paul, MN, USA). Ed. by John Moore. 26th–30th Oct. 2008, 1.E.2-1–1.E.2-7. URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4702767`.

[SCA13] *SCARLETT project website.* SCARLETT Consortium. 2013. URL: `http://www.scarlettproject.eu/` (visited on 20/06/2018).

[Sch11] Tobias Schoofs. *AIR – Overview.* Presentation. Lisbon, Portugal: GMV, 2011. URL: `http://www.gmv.com/export/sites/gmv/DocumentosPDF/air/Presentation_GMV-AIR-1.1.pdf`.

[SCS12] Cláudio Silva, João Cristóvão and Tobias Schoofs. 'An I/O Building Block for the IMA Space Reference Architecture'. In: *Proc. of DASIA 2012 Data Systems In Aerospace.* (Dubrovnic, Croatia, 14th–16th May 2012). Ed. by L. Ouwehand. SP-701. ESA Spacebooks Online, Aug. 2012. URL: `http://www.gmv.com/export/sites/gmv/DocumentosPDF/air/Paper_DASIA_2012.pdf`.

[SYS18] *PikeOS 4.2. RTOS with Hypervisor-Functionality.* Rel. 1.3. Product Overview. SYSGO. Klein-Winternheim, Germany, 2018. URL: `https://www.sysgo.com/fileadmin/user_upload/www.sysgo.com/redaktion/downloads/pdf/data-sheets/SYSGO-Product-Overview-PikeOS.pdf` (visited on 20/06/2018).

[TCA13] Salvador Trujillo, Alfons Crespo and Alejandro Alonso. 'MultiPARTES: Multicore for Mixed-criticality virtualization Systems'. In: *16th Euromicro Conference on Digital System Design.* (Santander, Spain). 4th–6th Sept. 2013. URL: `http://www.dit.upm.es/~str/papers/pdf/trujillo&13a.pdf` (visited on 20/06/2018).

[WDD11] James Windsor, Marie-Hélène Deredempt and Regis De-Ferluc. 'Integrated Modular Avionics for Spacecraft – User Requirements, Architecture and Role Definition'. In: *30th Digital Avionics Systems Conference, DASC 2011.* (Seattle, WA, USA). IEEE/AIAA. 16th–20th Oct. 2011, 8A6-1–8A6-16.

[WH09] James Windsor and K. Hjortnaes. 'Time and Space Partitioning in Spacecraft Avionics'. In: *Third IEEE Int'l Conf. on Space Mission Challenges for Information Technology, SMC-IT 2009.* (Pasadena, CA, USA). 19th–23rd July 2009, pp. 13–20. DOI: `10.1109/SMC-IT.2009.11`.

[Win12] James Windsor. 'Integrated Modular Avionics for Space. IMA4Space'. In: *6th ESA Workshop on Avionics, Data, Control and Software Systems, ADCSS 2012.* 23rd Oct. 2012. URL: `http://congrexprojects.com/docs/12c25_2310/sa1440_deredept.pdf?sfvrsn=2` (visited on 20/06/2018).