Building a Time- and Space-Partitioned Architecture for the Next Generation of Space Vehicle Avionics*

José Rufino, João Craveiro, and Paulo Verissimo

University of Lisbon, Faculty of Sciences, LaSIGE
ruf@di.fc.ul.pt, jcraveiro@lasige.di.fc.ul.pt, pjv@di.fc.ul.pt

Abstract. Future space systems require innovative computing system architectures, on account of their size, weight, power consumption, cost, safety and maintainability requisites. The AIR (ARINC 653 in Space Real-Time Operating System) architecture answers the interest of the space industry, especially the European Space Agency, in transitioning to the flexible and safe approach of having onboard functions of different criticalities share hardware resources, while being functionally separated in logical containers (partitions). Partitions are separated in the time and space domains. In this paper we present the evolution of the AIR architecture, from its initial ideas to the current state of the art. We describe the research we are currently performing on AIR, which aims to obtain an industrial-grade product for future space systems, and lay the foundations for further work.

1 Introduction

Space systems of the future demand for innovative embedded computing system architectures, meeting requirements of different natures. These systems must obey to strict dependability and real-time requirements. Reduced size, weight and power consumption (SWaP), along with low wiring complexity, are also crucial requirements both for safety reasons and to decrease the overall cost of a mission. A modular approach to software enabling component reuse among the different space missions also benefits the cost factor. At the same time, such an approach allows independent validation and verification of components, thus easing the software certification process.

A typical spacecraft onboard computer has to host a set of avionics functions, such as the Attitude and Orbit Control Subsystem (AOCS), the Telemetry, Tracking, and Command (TTC) subsystem, and one or more payload subsystems [13]. The traditional approach to space computing systems was to grant dedicated hardware resources to each of these functions.

* This work was partially developed within the scope of the European Space Agency Innovation Triangle Initiative program, through ESTEC Contract 21217/07/NL/CB, Project AIR-II (ARINC 653 in Space RTOS — Industrial Initiative, http://air.di.fc.ul.pt). This work was partially supported by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology), through the Multiannual Funding and CMU-Portugal Programs and the Individual Doctoral Grant SFRH/BD/60193/2009.

However, the ongoing trend goes towards the integration of multiple functions, so that they share the same hardware resources. While satisfying SWaP requirements, this introduces potential safety risks, since the applications supporting those functions may have different degrees of criticality and predictability, and/or may originate from multiple sources. The architectural principle proposed to cope with function integration has onboard applications being separated in logical containers, called partitions. Partitioning allows achieving both fault containment and independent software verification/validation capabilities. The safety of this approach is enforced through robust temporal and spatial partitioning (TSP) [25, 28]. Temporal partitioning concerns partitions not interfering with the fulfilment of each other's real-time requisites, while spatial partitioning encompasses the usage of separate addressing spaces dedicated to each partition. The aeronautic industry went through a similar process [32], introducing the Integrated Modular Avionics (IMA) [1] and ARINC 653 [2, 3] specifications.

In this paper, we present our past, present, and future research work on the AIR (ARINC 653 in Space RTOS) architecture for TSP aerospace systems. AIR has been prompted by the interest of the space industry in the adoption of TSP concepts [25, 30], especially the European Space Agency (ESA), which is currently active in this matter within the TSP Working Group [33]. The National Aeronautics and Space Administration (NASA) has expressed a similar interest for its next generation of space exploration vehicles [14, 12].

The design of the AIR architecture incorporates state-of-the-art features, such as coexistence of real-time operating systems (RTOS) and generic non-real-time ones, advanced timeliness control and adaptation mechanisms, and flexible development and integration tools. It also foresees extensions to the architecture so as to take advantage of multicore platforms.

The remainder of this paper is organized as follows. Section 2 details the successive evolution steps of the AIR architecture, from its inception ideas to the current state of the art. Section 3 details further the AIR architecture. Section 4 describes the work currently being done on the AIR architecture. Section 5 presents open research issues which we plan to tackle. Finally, Section 6 closes the paper with some concluding remarks.

2 Evolution of AIR design solutions

The first steps of the AIR project were performed under commissioning of ESA, within the scope of the Innovation Triangle Initiative program.

2.1 ARINC 653 interface in RTEMS

The initial idea was to perform a feasibility study to adapt the Real-Time Executive for Multiprocessor Systems (RTEMS) [18] to offer the application interface and functionality required by the ARINC 653 specification [2, 9]. The proposed solution involved an analysis of the RTEMS modules needing to be modified, extended, removed or added to cope with the missing ARINC 653 functionality. This approach was never strictly followed [23].

2.2 Single-executive core (SEC)

A more interesting solution, devised on the early stages of development, was to make the architecture design independent from the underlying RTOS kernel.

Stemming directly from the ARINC 653 specification, the original AIR design approach integrates a standard Application Executive (APEX) interface module, which maps into the service interface of a single RTOS kernel. The architecture of this Single-Executive Core (SEC) design, illustrated in Fig. 1, includes the RTOS kernel (providing functions such as process management, time and clock management, and interprocess synchronization and communication), and a module integrating the system-specific functions associated to the underlying processor infrastructure and to the specific platform hardware resources.

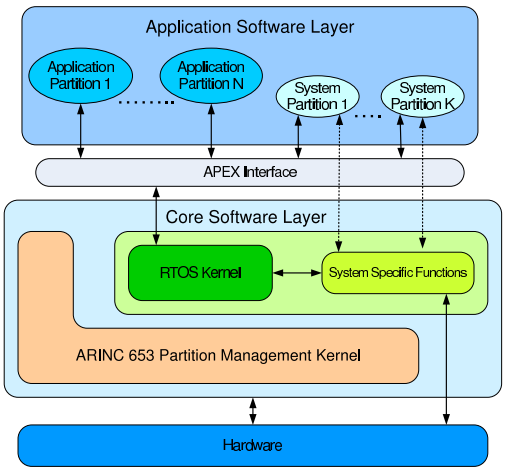

Fig. 1. Single-executive core design approach

The core functionality needed by the ARINC 653 specification (cyclic partition scheduling and priority-based process scheduling) was implemented by a specific module, the *ARINC 653 Partition Management Kernel*, shown in Fig. 1.

The SEC design exhibits an optimal memory footprint size, since there is only one instance of each component. Given that access to the APEX interface and RTOS kernel is shared among all partitions, the integrity and fault confinement attributes are restricted to the application software layer (see Fig. 1).

The SEC design and its proof-of-concept prototype have been very helpful to understand the realm of time and space partitioning and have proved the feasibility of implementing the ARINC 653 functionality making use of off-the-shelf operating systems. The proof-of-concept prototype was developed and demonstrated using RTEMS 4.6.6 on an Intel IA-32 platform.

2.3 Multi-executive core (MEC)

To improve the integrity and fault confinement properties of the AIR architecture, a Multi-Executive Core (MEC) design was approached. In MEC, the separation in logical containers is extended throughout all layers, as shown in Fig. 2. This is achieved by providing an APEX interface, an RTOS kernel and possibly system specific functions on a per-partition basis. From the SEC approach, the MEC design solution preserves the independence from a given operating system (OS) and permits a homogeneous integration of different instances of the same RTOS kernel or a hybrid approach, possibly integrating different RTOS kernels in different partitions, the *Partition Operating Systems* (POS).

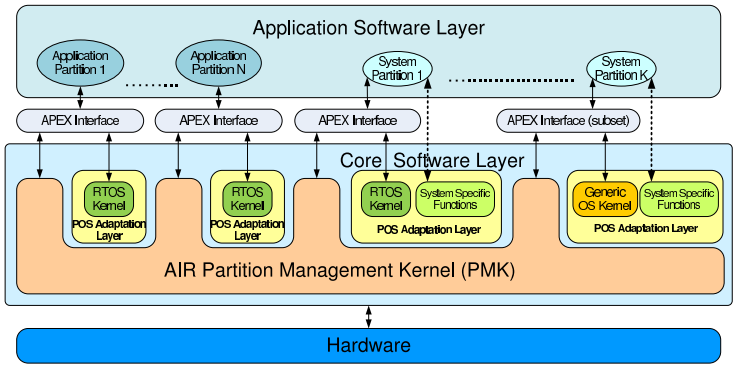

Fig. 2. Multi-executive core AIR architecture

The new *AIR Partition Management Kernel* (PMK) provides the core functionality needed for conformity with the ARINC 653 specification [2], separating the partition and process scheduling functions. Process scheduling is ensured by the native RTOS process scheduler of each partition. The AIR PMK also includes partition management and support for interpartition communication.

A proof-of-concept prototype for the MEC architecture was developed and demonstrated using RTEMS 4.8 as partition operating system. Two processor platforms have been approached: an Intel IA-32 platform, and SPARC ERC32 and SPARC LEON-based platforms.

2.4 Comparison between SEC and MEC design solutions

The MEC design solution mostly improves on a set of relevant attributes, in comparison with the SEC solution. A system-wide suboptimal footprint size is outweighed by the benefits obtained in terms of flexibility, configurability, integrity and fault confinement. The MEC design is further flexible in the sense that it allows the integration of different RTOS kernel instances in different partitions. A comparison detailing these improvements is presented in Table 1.

Table 1. Comparison between the AIR design solutions' attributes

	Single-executive core (SEC)	**Multi-executive core (MEC)**	
RTOS Integration	shared	per partition	
Flexibility	fair	good	
Configurability	system-wide	per partition	
Integrity and Fault Confinement	application layer	all layers	
Footprint Size	optimal (system-wide)	optimal (per partition)	
Development Tools	canonical	canonical	canonical and specific
Bootstrap Method	single image	single image	multiple images

The remaining design attributes in Table 1 do address the requirements of the development tools and of the application bootstrap methodology. One limitation of the application production toolchain concerns a common inability of canonical link editors to combine in a single object the different instances of the same RTOS kernel, since they use the same naming references. A methodology is needed to tackle this problem. A tag filter utility, which appends a partition identifier to each public symbol, is used. The objects of each partition may afterwards be combined by a canonical link editor into a single linked object. The MEC design solution, allowing multiple objects to be specified for bootstrapping, opens room for the dynamic update of individual partition applications.

3 AIR System Architecture

Figure 2 also shows some later additions to the AIR architecture definition, which we will now discuss.

3.1 Temporal and spatial partitioning

The robust partitioning approach defined in the AIR architecture implies the temporal and spatial separation of the different operating systems and its applications in integrity and criticality containers, defined by partitions. Temporal partitioning is achieved through a two-level hierarchical scheduling scheme pictured in Fig. 3. In the first level, partitions are scheduled according to a cyclic sequence of fixed time slices. Inside each partition, processes compete with each other according to the native process scheduler of the partition; in the case of RTOSs, this is typically a dynamic priority-based scheduler [24].

Partitions spatially encapsulate the addressing spaces of the contained POS and applications. No component of a given partition can directly access the addressing space of other partitions, thus guaranteeing that partitions do not interfere with each other [22, 24]. A highly modular design approach in the support

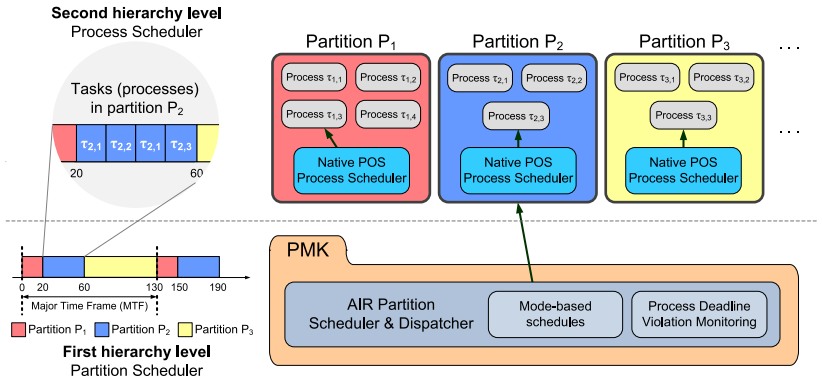

Fig. 3. AIR two-level hierarchical scheduling

of AIR spatial partitioning was followed, with requirements (specified in configuration files with the assistance of development tools support) being described in runtime through a high-level processor independent abstraction layer [22]. Hardware-mapping descriptors are provided per partition, primarily corresponding to the several levels of execution of an activity (e. g., application, POS kernel and AIR PMK) and to its different memory blocks (e. g., code, data and stack).

3.2 Advanced timeliness control and adaptation mechanisms

A basic partition scheduling scheme, with a single partition scheduling table defined at integration time, is very limiting regarding the different temporal characteristics a space mission can adopt in distinct phases of its operation (e. g., takeoff, flight, exploration) or regarding the accommodation of component failures. The AIR advanced design addresses this issue by introducing support for multiple *mode-based partition schedules*. The basic partition scheduling scheme is extended to allow multiple schedules to be defined. At execution time, authorized partitions may request switching between the different partition schedules [5, 24].

Another timeliness control mechanism introduced in AIR is *process deadline violation monitoring*. During the execution of the system, it may be the case that a process exceeds its deadline; this can be caused by a malfunction or because that process's worst-case execution time (WCET) was underestimated at system configuration and integration time. Other factors related to faulty system planning (such as violation of the partitions' timing requirements) can be predicted and avoided using offline tools [7], addressed in Sect. 4.1. The process deadline violation monitoring procedure is optimized regarding deadline violation detection latency and regarding computation complexity so as not to have minimal temporal interference with the rest of the system, since it is performed inside the system clock interrupt service routine: the earliest deadline is checked; following deadlines may subsequently be verified until one has not been missed [5, 24].

3.3 Flexible partition operating system integration

The *AIR POS Adaptation Layer* (PAL) encapsulates each POS providing a common interface to the surrounding components (AIR PMK, APEX). This way, the necessary changes to support using a new family or version of POS are circumscribed to a smaller component, and previous or ongoing verification, validation and/or certification efforts on more complex ones are not hindered [5, 6].

3.4 Integration of generic operating systems

The foreseen heterogeneity between POSs is also being extended to include generic non-real-time OSs, such as Linux, answering to a recent trend in the aerospace industry. This is motivated by the lack of relevant functions in most RTOSs, which are commonly provided by generic non-real-time operating systems. Porting these functions (e. g., scripting language interpreters) to RTOSs can be a complicated and error-prone task [16]. An embedded variant of Linux has been approached, and wields a fully functional OS with a minimal size compatible with typical space missions requirements [8, 5].

To ensure that a non-real-time OS cannot undermine the overall time guarantees of the system by disabling or diverting system clock interrupts, instructions that could allow this are wrapped by low-level handlers (paravirtualized) [5, 6].

3.5 Flexible Portable APEX

The ARINC 653 specification defines a standard interface between applications and the core software layer [2], the APEX interface. The *AIR APEX interface* component (Fig. 2) supports this feature, exploiting the availability of AIR PAL-related functions and implementing the advanced notion of *Portable APEX* [26].

3.6 AIR Health Monitoring (HM)

The AIR Health Monitor is responsible for handling hardware and software errors (like deadlines missed, memory protection violations, or hardware failures). The aim is to isolate errors within its domain of occurrence: process level errors will cause an application error handler to be invoked, while partition level errors trigger a response action defined in a system configuration table. Errors detected at system level may lead the entire system to be stopped or reinitialized [22].

3.7 Interpartition communication

Interpartition communication aims to support the transfer of information between partitions, and its relation with spatial partitioning implies the use of specific executive interface services encapsulating and providing the transfer of data from one partition to another without violating spatial segregation constraints. The core of AIR interpartition communication mechanisms are integrated at the

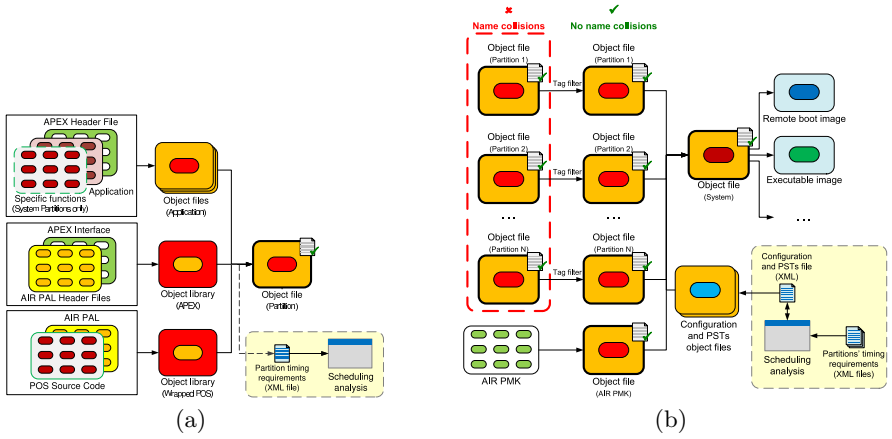

Fig. 4. Introduction of scheduling analysis features for (a) application developers, and (b) system integrators

Portable APEX interface level. Memory protection and, if required, memory-to-memory copy mechanisms are managed at the AIR PMK level [22, 24].

The interpartition communication abstractions required for conformity with ARINC 653, sampling ports and queuing ports, model each partition's way to communicate (send *or* receive messages) through a communication channel.

4 Present lines of work

4.1 Scheduling and composability

In [7], we discuss how we can profit from composability properties inherent to the build and integration process of AIR-based systems to allow validating scheduling requirements independently at different levels of the build and integration process. We proposed the development of rules, techniques and tools to support this purpose, so as to provide both schedulability analysis capabilities but also tool-assisted generation of partition scheduling tables.

Schedulability results for TSP systems allow extensions to the software build and integration processes of AIR-based systems, for the benefit of both application developers (Fig. 4(a)) and system integrators (Fig. 4(b)). This benefit obviously depends on software tools, using such results, being made available to them. The goal of application developers having a scheduling analysis phase introduced in their production cycle is for them to be able to independently analyse the feasibility of their applications, provided the timing requirements (period, WCET, deadline, etc.) of the composing processes. The information of these timing requirements can be either estimated by the developers or tentatively determined through static code analysis [19].

4.2 Multicore

Multicore processors are paving their way into the realm of embedded systems [17], but their use in TSP platforms has not been addressed in detail [5]. In this sense, we pursue the extension of TSP concepts to allow multicore-enabled AIR-based systems, which feasibility shall be backed up by both schedulability and safety considerations. The applicability of multicore includes strengthening overall safety through adaptive fault tolerance mechanisms, and augmenting the integration potential of a single system with the contribute of different facets of parallelism. The tools proposed in [7] should in this case be extended to accommodate multicore support.

This research line includes the profound analysis of the impact of parallelism, both intrapartition parallelism (i. e., between processes) and between partitions. The approach to intrapartition parallelism aims to understand the advantages and drawbacks in distributing processes in a partition among processor cores. Concerning parallelism between partitions, two scheduling approaches will be studied: *(i)* static (extending system configuration mechanisms, to allow explicit definition of when and how parallelism between partitions occurs), and; *(ii)* semi-dynamic (extending configuration mechanisms, to allow expressing restrictions and dependencies that will guide the activity of a dynamic partition scheduler with support for parallelism between partitions) [5].

4.3 Remote and online application update

The Mars Rover Pathfinder is an example of a mission where the (in this case fortuitous) possibility to modify the mission's configuration remotely was crucial for its survival [15]. Due to the impossibility of direct access to the spacecraft, it is extremely important to have the possibility for the onboard system to remotely receive software updates [20].

At this stage, AIR abstracts from issues inherent to the communication between the spacecraft and the ground station, focusing on the management and treatment of the update information (integrity, correctness, domain of application). Remote software update may have to cope with critical software components that must be updated without interruptions to their execution.

Other interesting issues include remote system monitoring and modification of system-wide control parameters (such as partition scheduling tables). These will increase the extent to which the advanced timeliness control and adaptation mechanisms of AIR are taken advantage of.

5 Future

5.1 Operating system integration

The work mentioned in Sect. 3.4 shall be extended in two ways. On the one hand, the study involving embedded Linux [8, 5] will evolve into a fully functional Linux integration. On the other hand, the principles should be applied for the

integration of other generic non-real-time operating system, such as Windows through the Windows Research Kernel [27]. Furthermore, for specific application support, the integration of other RTOS kernels, such as eCos, is also envisaged.

5.2 Sensors, actuators and networks

Any spacecraft needs interfacing with surrounding environment, through sensors and actuators. For instance, the AOCS function needs to get information from star/Sun sensors and reference gyroscopes. On the other hand, AOCS needs to actuate on reaction wheels and propulsion drive thrusters. The safety of these interactions with the environment should be supported by extending the spatial partitioning mechanisms to input/output (I/O) addressing spaces.

A particular case of I/O functions is network communication. This may include wired network interfaces, such as: legacy MIL-STD-1553 [10] systems; *dependable* Controller Area Network (CAN) buses [21]; high-rate SpaceWire [11] and TTEthernet [31] links. For some space systems, such as planetary robotic explorers, wireless sensor networks with improved dependability and timeliness may be of the utmost importance for sensing and coordination actions [29].

5.3 Information security

The AIR architecture still requires the incorporation of security concerns. One option for partitioned security is the notion of Multiple Independent Levels of Security and Safety (MILS) [4]. This implies that, at the application level, execution is confined to the application's partition, with controlled communication with the remaining partitions. All communication passes through the security components, which can include monitoring and cryptographic mechanisms.

To fulfil MILS requisites, the AIR architecture will incorporate the provision of privacy- and authenticity-capable interpartition communication services, using the cryptographic mechanisms and algorithms most adequate to the characteristics of TSP systems (for encryption and, possibly, key distribution). This extension includes the analysis of the architectural, hardware and cryptographic algorithm requirements for this functionality.

6 Conclusion

This paper presents the evolution of the AIR architecture, from its initial ideas to the current state of the art. AIR targets space systems of the future, and current work aims to turn it into an industrial-grade product. The first approach to AIR was a single-executive core design, but soon evolved in to a multi-executive one. Subsequent research work provided AIR with flexible support to different partition operating systems (both real-time and non-real-time) and advanced timeliness control and adaptation mechanisms. Current work focuses on schedulability issues, taking advantage of multicore platforms, and remote online update of applications. Research directions for the future include expansion of support to new

operating systems, interfaces with input/output devices (including networking), and security (privacy and authenticity) of information exchanges.

Acknowledgment

The authors would like to thank all the researchers, engineers and collaborators who worked throughout the time in the AIR Technology (AIR and AIR-II projects), at FCUL, GMV Portugal, Thales Alenia Space, and ESA–ESTEC.

References

1. AEEC: Design guidance for Integrated Modular Avionics. ARINC Report 651-1 (Nov 1997)
2. AEEC: Avionics application software standard interface, part 1 - required services. ARINC Specification 653P1-2 (Mar 2006)
3. AEEC: Avionics application software standard interface, part 2 - extended services. ARINC Specification 653P2-1 (Dec 2008)
4. Alves-Foss, J., Harrison, W.S., Oman, P., Taylor, C.: The MILS architecture for high-assurance embedded systems. Int. J. of Embedded Systems 2, 239–247 (2006)
5. Craveiro, J.: Integration of generic operating systems in partitioned architectures. M.Sc. thesis, Faculty of Sciences, University of Lisbon, Lisbon, Portugal (2009)
6. Craveiro, J., Rufino, J., Schoofs, T., Windsor, J.: Flexible operating system integration in partitioned aerospace systems. In: Actas do INForum - Simpósio de Informática 2009. pp. 49–60. Lisbon, Portugal (Sep 2009)
7. Craveiro, J., Rufino, J.: Schedulability analysis in partitioned systems for aerospace avionics. In: Proc. 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA 2010). Bilbao, Spain (Sep 2010)
8. Craveiro, J., Rufino, J., Almeida, C., Covelo, R., Venda, P.: Embedded Linux in a partitioned architecture for aerospace applications. In: Proc. 7th ACS/IEEE Int. Conf. on Computer Systems and Applications (AICCSA 2009). pp. 132–138. Rabat, Morocco (May 2009)
9. Diniz, N., Rufino, J.: ARINC 653 in space. In: Proc. DASIA 2005 "DAta Systems In Aerospace" Conf. Edinburgh, Scotland (Jun 2005)
10. ECSS: Space engineering: Interface and communication protocol for MIL-STD-1553B data bus onboard spacecraft. Standard ECSS-E-50-13 Draft C, ESA Requirements and Standards Division (May 2008)
11. ECSS: Space engineering: SpaceWire — links, nodes, routers and networks. Standard ECSS-E-ST-50-12C, ESA Requirements and Standards Division (Jul 2008)
12. Fletcher, M.: Progression of an open architecture: from Orion to Altair and LSS. Tech. rep., Honeywell International (May 2009)
13. Fortescue, P.W., Stark, J.P.W., Swinerd, G. (eds.): Spacecraft Systems Engineering, 3rd edition. Wiley (2003)
14. Hodson, R., Ng, T.: Avionics for exploration. In: NASA Technology Exchange Conference, Galveston, TX, USA. (Nov 2007)
15. Jones, M.: What really happened on Mars Rover Pathfinder. The RISKS Digest - Forum on Risks to the Public in Computers and Related Systems 19(49) (Dec 1997), `http://catless.ncl.ac.uk/Risks/19.49.html`

16. Kinnan, L.: Application migration from Linux prototype to deployable IMA platform using ARINC 653 and Open GL. In: Proc. 26th IEEE/AIAA Digital Avionics Systems Conference. pp. 6.C.2–1–6.C.2–5. Dallas, TX, USA (Oct 2007)
17. Mignolet, J.Y., Wuyts, R.: Embedded multiprocessor systems-on-chip programming. IEEE Software 26(3), 34–41 (May/Jun 2009)
18. OAR - On-Line Applications Research Corporation: RTEMS C Users Guide, 4.8 (Feb 2008)
19. Pushner, P., Koza, C.: Calculating the maximum execution time of real-time programs. Journal of Real-Time Systems 1, 160–176 (Sep 1989)
20. Rosa, J., Craveiro, J., Rufino, J.: Exploiting AIR composability towards spacecraft onboard software update. In: Actas do INForum - Simpósio de Informática 2010. Braga, Portugal (Sep 2010)
21. Rufino, J., Almeida, C., Verissimo, P., Arroz, G.: Enforcing dependability and timeliness in Controller Area Networks. In: Proc. 32nd Ann. Conf. of the IEEE Industrial Electronics Society (IECON'06). Paris, France (Nov 2006)
22. Rufino, J., Craveiro, J., Schoofs, T., Tatibana, C., Windsor, J.: AIR Technology: a step towards ARINC 653 in space. In: Proc. DASIA 2009 "DAta Systems In Aerospace" Conf. Istanbul, Turkey (May 2009)
23. Rufino, J., Filipe, S., Coutinho, M., Santos, S., Windsor, J.: ARINC 653 interface in RTEMS. In: Proc. DASIA 2007 "DAta Systems In Aerospace" Conf. Naples, Italy (Jun 2007)
24. Rufino, J., Craveiro, J., Verissimo, P.: Architecting robustness and timeliness in a new generation of aerospace systems. In: Casimiro, A., de Lemos, R., Gacek, C. (eds.) Architecting Dependable Systems 7. LNCS, Springer-Verlag, Berlin Heidelberg (2010), accepted for publication
25. Rushby, J.: Partitioning in avionics architectures: Requirements, mechanisms and assurance. NASA Contractor Report CR-1999-209347, SRI International, California, USA (Jun 1999)
26. Santos, S., Rufino, J., Schoofs, T., Tatibana, C., Windsor, J.: A portable ARINC 653 standard interface. In: Proc. IEEE/AIAA 27th Digital Avionics Systems Conf. (DASC '08). St. Paul, MN, USA (Oct 2008)
27. Schöbel, M., Polze, A.: Kernel-mode scheduling server for CPU partitioning: a case study using the Windows Research Kernel. In: Proc. 2008 ACM Symp. on Applied Computing (SAC 2008). pp. 1700–1704. ACM, Fortaleza, Ceará, Brazil (2008)
28. Seyer, R., Siemers, C., Falsett, R., Ecker, K., Richter, H.: Robust partitioning for reliable real-time systems. In: Proc. 18th Int. Parallel and Distributed Processing Symp. pp. 117–122 (Apr 2004)
29. Souza, J.L.R., Rufino, J.: Characterization of inaccessibility in wireless networks: a case study on IEEE 802.15.4 standard. In: Proc. IESS International Embedded Systems Symposium '09. Langenargen, Germany (Sep 2009)
30. Terraillon, J.L., Hjortnaes, K.: Technical note on on-board software. European Space Technology Harmonisation, Technical Dossier on Mapping, TOSE-2-DOS-1, ESA (Feb 2003)
31. TTTech: TTEthernet specification. Document D-INT-S-10-002, TTTech Computertechnik AG (Nov 2008)
32. Watkins, C., Walter, R.: Transitioning from federated avionics architectures to Integrated Modular Avionics. In: Proc. 26th IEEE/AIAA Digital Avionics Systems Conf. (DASC 2007). Dallas, TX, USA (Oct 2007)
33. Windsor, J., Hjortnaes, K.: Time and space partitioning in spacecraft avionics. In: Proc. 3rd IEEE Int. Conf. on Space Mission Challenges for Information Technology (SMC-IT 2009). pp. 13–20. Pasadena, CA, USA (Jul 2009)