AIR TECHNOLOGY: A STEP TOWARDS ARINC 653 IN SPACE

José Rufino*[,1], João Craveiro[1], Tobias Schoofs[2], Cássia Tatibana[2], and James Windsor[3]

[1]*Faculdade de Ciências da Universidade de Lisboa, Campo Grande, 1749-016 Lisboa, Portugal, E-mail: ruf@di.fc.ul.pt*
[2]*Skysoft Portugal, S.A. - Av. D. João II, Lote 1.17.02, 7, 1998-025 Lisboa, Portugal, E-mail: tobias.schoofs@skysoft.pt*
[3]*European Space Agency, ESA/ESTEC, 2200 AG Noordwijk, The Netherlands, E-mail: James.Windsor@esa.int*

ABSTRACT

The Integrated Modular Avionics and the ARINC 653 specifications are assuming a key role in the provision of a standard operating system interface for safety-critical applications in both the aeronautic and space markets. The AIR Technology, designed within the scope of an ESA initiative to develop a proof of concept, implements the notion of robust temporal and spatial partitioning. A different operating system kernel may be used per partition, furnishing the bare services to build the ARINC 653 application programming interface. This paper describes the advances done during AIR-II, an initiative to evolve the AIR Technology proof of concept towards an industrial product. Current prototype activities are based on RTEMS and on the SPARC V8 LEON3 processor and work is being done on the integration of Linux in the AIR Technology.

Key words: ARINC 653; Real-Time Kernels; RTEMS; Safety-critical embedded systems.

1. INTRODUCTION

The IMA (Integrated Modular Avionics) [1] concept emerged in modern avionics as a challenge to the federated avionics system architecture, where each avionics function owns and uses dedicated computer resources. The IMA architecture makes use of a high-integrity partitioned environment that may host multiple avionics functions of different criticalities on a shared computing platform [2]. In this realm, the ARINC 653 specification [3, 4] is a very important block from the IMA definition [1] and the partitioning concept emerges for protection and functional separation between applications, usually for fault containment and ease of validation, verification, and certification [5]. Examples of its application in the civil aviation industry include the operating systems shipped in the Airbus A380 and Boeing 787 commercial aircrafts.

In the last years the ARINC 653 specification and its concept of partitioning (temporal and spatial) have received considerable attention from the space community. Funded by the European Space Agency (ESA) several studies are currently analyzing the interest and usefulness of using temporal and space partitioning for space on-board software [5, 6, 7, 8].

The technological interest of ESA led to the support in the development of a proof of concept [7, 9] and a demonstration of feasibility of use, within the scope of the ESA innovation triangular initiative (projects AIR and AIR-II). This paper mainly discusses how the AIR architecture has evolved from the original AIR proof of concept towards the mature AIR-II technology demonstrator.

2. ARINC 653 FUNDAMENTAL CONCEPTS

The ARINC 653 specification [3], defined for aeronautical applications, has the goal of providing a standard interface between a given real-time operating system (RTOS) and the corresponding applications; this interface is designated as the APEX (Application Executive) interface. ARINC 653 also presents a concept of temporal and spatial segregation, which lies on confining each application (dubbed "partition" in ARINC 653 terminology) to its memory space and to its temporal window of possession of computing resources. The overall goal is to secure safety and timeliness in mission-critical systems, a concern which is also present in space applications [6, 7], and which led to the interest of the European Space Agency.

2.1. ARINC 653 System Architecture

The architecture of a standard ARINC 653 system is illustrated in Fig. 1. It comprises the application software layer, with each application running in a confined context its partition. The application software layer may include also a set of optional system partitions intended to manage the interactions with specific hardware devices. Ap-

*This work was partially supported by the European Space Agency (ESA), through ESTEC Contract 21217/07/NL/CB – Project AIR-II (ARINC 653 in Space – Industrial Initiative, URL: http://air.di.fc.ul.pt), and by FCT through the Multiannual Funding and the CMU-Portugal Programs.

propriate support from the core software layer (e.g. hardware interfacing and device drivers) is required.

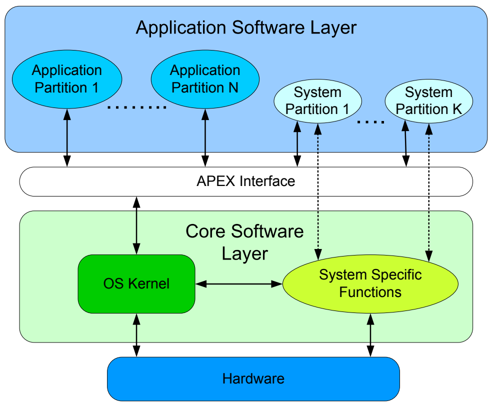

Figure 1. Standard ARINC 653 System Architecture

Application partitions generally consist of one or more processes, and can only use the services provided by a logical application executive (APEX) interface, as defined in the ARINC 653 specification [3]. However, a system partition may use also specific functions provided by the core software layer, thus being allowed to bypass the standard APEX interface.

The execution environment provided by the OS kernel module must furnish a relevant set of operating system services, such as process scheduling and management, time and clock management as well as inter-process synchronization and communication. Containment of possible faults inside the domain of each partition must be ensured by the core software layer [3].

2.2. Spatial and Temporal Partitioning

Spatial partitioning ensures that it is not possible to an application to access the memory space (both code and data) of another application running on a different partition. Temporal partitioning ensures that the activities in one partition do not affect the timing of the activities in other partition.

2.3. Health Monitoring

The Health Monitoring (HM) functions consist in a set of mechanisms to monitor system resources and application components. The HM helps to isolate faults and to prevent failures from propagating. Within the scope of the ARINC 653 standard specification the HM functions are defined for process, partition and system levels [3]. The fundamental issue regarding the migration of the HM services from the aeronautic to the space environment is the impossibility of providing human assistance on space devices. As such the HM mechanisms on space must provide much stronger recovery mechanisms as to assure a failure will not produce permanent losses.

2.4. ARINC 653 Service Interface

The ARINC 653 service requests define the application executive APEX interface layer (Fig. 1) provided to the application software developer and the facilities the core executive shall supply. A required set of services is mandatory to claim strict compliance with the ARINC 653 standard [3]. Those services are grouped in the following major categories: partition and process management, time management, intra- and inter-partition communications, and health monitoring.

3. AIR TECHNOLOGY EVOLUTION

The AIR innovation initiative represents a first but significant step towards the usage of off-the-shelf open-source operating systems, such as RTOS kernels or even generic operating systems (e.g. embedded and real-time Linux), in the definition and design of ARINC 653 based systems.

3.1. AIR System Architecture Overview

The key idea in the definition of the AIR system architecture is a simple solution for providing the ARINC 653 functionality missing in off-the-shelf (real-time) operating system kernels, which implies encapsulating those functions in components with a well-defined interface and adding them to the bare operating system architecture. That means, the AIR architecture in essence preserves the hardware and operating system independence defined in the ARINC 653 specification [3, 9].

Three fundamental components are identified in the bare architecture of AIR (cf. Fig. 2): the AIR Partition Management kernel (PMK), a simple microkernel that efficiently handles partition scheduling and dispatching, and inter-partition communication; a (real-time) operating system kernel per partition, generally referred as Partition Operating System (POS); an application executive (APEX) interface, defining a set of services in strict conformity with the ARINC 653 standard [3]. For generic operating systems (e.g. embedded Linux) only a subset of the APEX primitives is needed, primarily for management and monitoring purposes [10].

3.2. Versatile Operating System Integration

The AIR architecture allows operating systems integration without any fundamental change to a given POS Kernel. In essence, only the OS initialization process and the system clock handler need to be adapted. This was established in the course of designing the original AIR architecture. To a given extent, this corresponds to some form of paravirtualization [11]. In AIR-II this approach was formalized, extended and exploited to allow a versatile and flexible integration of partition operating systems

The lessons learned from a preliminary work on generic operating systems, such as (embedded) Linux [10], were extremely valuable to establish a uniform methodology for the integration of both real-time and generic operating systems.

The solution adopted for the AIR-II architecture wraps the POS and, if applicable, the system specific functions, through the use of a POS Adaptation Layer (PAL), facilitating the integration at the low and at the high level domains, i.e. with respect to the AIR PMK and APEX compenents, as can be seen in Fig. 2.

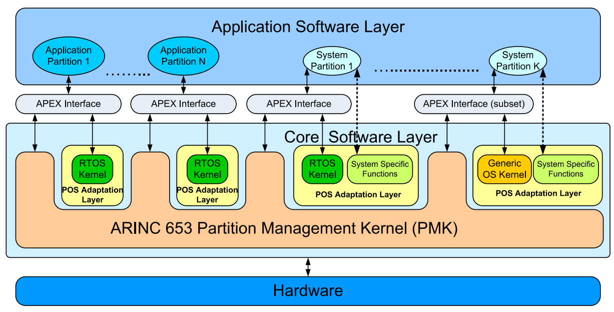

Figure 2. Overview of the AIR System Architecture

3.3. AIR Robust Temporal and Spatial Partitioning

In the original AIR architecture, temporal segregation is secured through a fixed cyclic partition scheduling scheme over a major time frame (MTF), in conformity with the ARINC 653 standard [3]. This is secured at the PMK level. A dedicated component, the AIR PMK Partition Scheduler, checks at each system clock tick whether a partition pre-emption is to occur; if it is so, the AIR PMK Partition Dispatcher has to save the context of the running partition and restore the context of the heir partition. The POS native process scheduler of the heir partition is then notified of the amount of clock ticks elapsed since it was last pre-empted, thus adjusting the heir partition system time to a common partition-wise time referential.

A two-level hierarchical approach was followed in the integration of AIR PMK components, namely the AIR PMK Partition Scheduler and the POS native process scheduler, as show in Fig. 3.

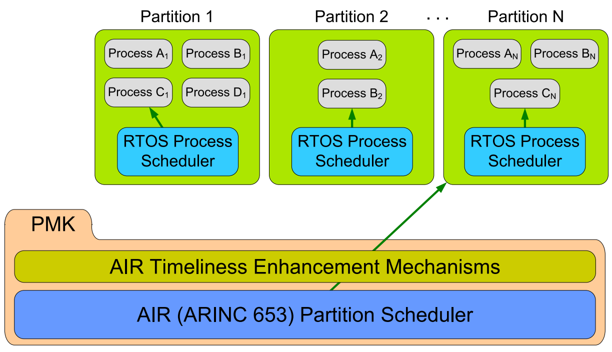

Figure 3. Integration of AIR PMK Partition Scheduler and POS Process Scheduler

Enhancing Timeliness Attributes of AIR-II

The diagram of Fig. 3 also includes an AIR-II PMK specific module intended to enhance the timeliness attributes of the AIR architecture, which contributes to secure the overall robustness characteristics of the AIR-II technology. In AIR-II, the design of PMK incorporates two special timeliness enhancements: mode-based schedules support, and process deadline monitoring.

The *mode-based schedules* functionality, which stems from ARINC 653 Part 2 [4], comprises simple, yet highly effective, mechanisms allowing the ability to switch among multiple partition schedules; this is useful both to implement fault tolerance mechanisms and to optimize partition scheduling for different modes of operation or phases of a mission. Support for schedule change actions, which take place the first time each partition is dispatched after each schedule switch, is also included.

Process deadline monitoring is performed by an optimized POS-independent scheme, inserted at the PMK level before the POS native process scheduler, which bypasses the clock tick announcement routine of the POS. Only the earliest deadline is verified at each system clock tick, with the possibility of optimizing this process even further when it occurs immediately following a partition dispatch (since multiple clock ticks are to be announced at once). Violations of the process deadlines are reported to the health monitoring mechanisms.

Securing Robust Spatial Partitioning in AIR-II

A highly modular design approach has also been followed in the support of AIR spatial partitioning. Spatial partitioning requirements, specified in ARINC 653 configuration files with the assistance of development tools support, are described in run-time through a high-level processor independent abstraction layer. A set of descriptors is 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 sections (e.g. code, data and stack), as illustrated in the diagram of Fig. 4.

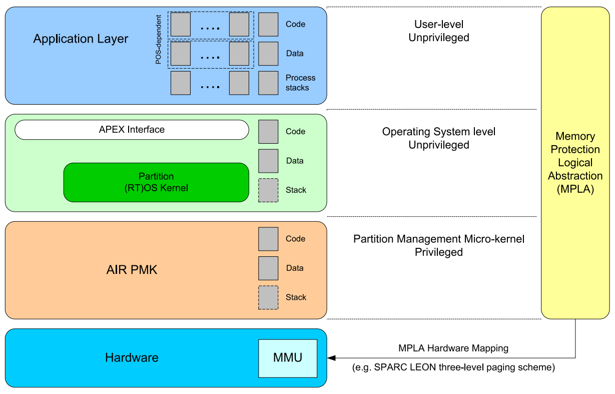

Figure 4. AIR-II Spatial Segregation Scheme

In AIR Technology, the definition of the high-level abstract spatial partitioning takes into account the semantics expected by user-level application programming. At each partition, the application environment inherits the execution model of the corresponding POS and/or its language run-time environment. This is true for system partitions and may be applied also to application partitions, using only the standard APEX interface.

The high-level abstract spatial partitioning description needs to be mapped in run-time to the specific processor memory protection mechanisms, possibly exploiting the availability of a memory management unit (MMU). Possible examples of such mapping are: the fence registers of the ATMEL AT697E SPARC V8 LEON2 processor or the Gaisler SPARC V8 LEON3 MMU core.

Mapping of high-level abstract partitioning also includes the management of privilege levels: only the AIR PMK is executed in privileged mode (cf. Fig. 4). The lack of multiple protection rings, such as it exists in the Intel IA-32 processor architecture, may be mitigated in the SPARC V8 architecture by granting access to a given level only during the execution of services belonging to that level (or lower ones). This may be achieved by activating the corresponding memory protection descriptors upon call of a service primitive, and deactivating them when service execution ends.

The provision of these mapping functions is under the scope of overall partition management as provided by the APEX core layer and by some specific AIR PMK components. For example, the mapping into processor specific descriptors needs to be updated in run-time when a partition change occurs. This has to be coordinated by the AIR PMK Partition Dispatcher, as illustrated in Fig. 5.

3.4. AIR Inter-partition Communication

Inter-partition communication was introduced in AIR-II 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 [3]. The core AIR inter-partition communication mechanisms are integrated at the APEX (core) level; memory protection and, if required, memory-to-memory copy mechanisms are managed at PMK-level (cf. Fig. 5).

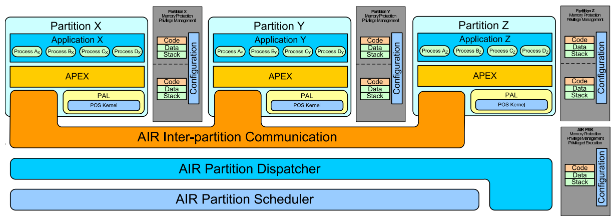

Figure 5. APEX (core) and AIR-PMK components and its relation with inter-partition communication

3.5. Flexible Portable APEX Interface

The APEX Interface implements the services defined in the ARINC 653 standard. The AIR APEX was improved in AIR-II, and consists of two components: the APEX Core Layer and the APEX Layer Interface. The relationship between such two components can be seen in the diagram of Fig. 6.

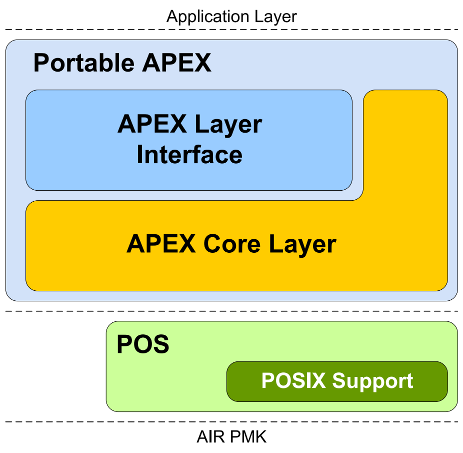

Figure 6. Architecture of the re-designed AIR-II APEX Interface

The APEX core Layer implements the advanced notion of *Portable APEX* intended to ensure portability between the different OS supported by AIR. It exploits the POSIX application programming interface that is currently available on most (RT)OS [12, 13]. An optimized implementation may invoke directly the native (RT)OS service primitives. In addition, it may also benefit from the availability of PAL-related functions (cf. Fig. 2).

On top of the APEX core layer, the partition and process management services, the intra-process and inter-partition communication services and the health monitoring services are built. The partition management, interpartition communication and health monitoring services rely additionally on the PMK service interface. In this respect, the PMK provides the partition-wise handling of memory protection descriptors.

The APEX also coordinates when required the interactions with the AIR Health Monitor, e.g. upon the detection of an error.

3.6. AIR Health Monitoring

The AIR Health Monitor introduced in AIR-II is responsible for handling hardware and software errors (like deadlines missed, memory protection violations, bounds violation or hardware failures). As much as possible, it will isolate the error propagation 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 by the partition Health Monitor table in the ARINC 653 configuration. The response action may be shutting down the entire partition, reinitializing the partition again or simply ignoring the error (cf. Fig. 7). Partition errors may be also raised as a consequence of process level errors that cannot be handled by the application error handler. Errors detected at system level may lead the entire system to be stopped or reinitialized.

The design of AIR allows Health Monitoring handlers to simply replace existing handlers or to be added to existing ones in pre- and/or post-processing modes.

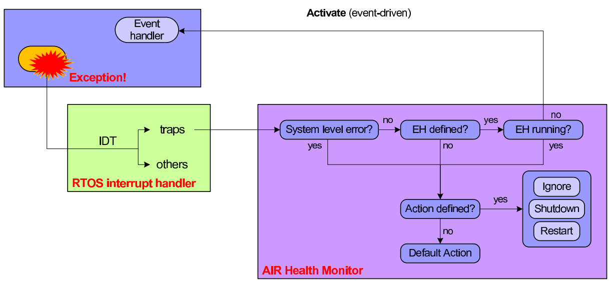

Figure 7. AIR-II Health Monitoring Mechanisms

4. CONCLUDING REMARKS AND FUTURE WORK

The AIR technology aims to provide the developers and the integrators of space on-board software with an environment that is standard and in strict conformity with the ARINC 653 specification [3]. The AIR solution is hardware and operating system independent. The AIR design allows in principle the versatile integration of both open-source and commercial operating system kernels. This paper has described the evolution of the AIR Technology, performed within the AIR-II Project, towards the integration of ARINC 653 based applications in space.

Current development makes use of RTEMS (Real-Time Executive for Multiprocessor Systems) as a significant representative of RTOS kernels [9]. RTEMS is a real-time multitasking kernel qualified for use in space on-board software developments. In addition, an embedded version of Linux is currently being studied to be integrated in the AIR architecture. This allows non-time-critical applications to execute with a guaranteed time slice within the major time frame, without violating the overall safety and timeliness properties of the system [10].

The evolution of the AIR Technology has enforced the concept of robust temporal and spatial partitioning at the several levels of the system and has added some advanced architectural features such as: mode-based scheduling; process deadline run-time monitoring; flexible APEX interface definition; health monitoring functions for error processing at all the levels of the system.

Future developments may introduce security features (such as multiple and independent levels of security) and adaptation to the building block approach currently being pushed by ESA. Integration with the ESA Virtual Spacecraft Reference Facility [14] is also under scope.

REFERENCES

[1] Airlines Electronic Engineering Committee (AEEC), design guidance for integrated modular avionics (ARINC specification 651). ARINC, 1991.

[2] C.B. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *Proceedings of the IEEE/AIAA 26th Digital Avionics Systems Conference (DASC '07)*, October 2007.

[3] Airlines Electronic Engineering Committee (AEEC), Avionics application software standard interface, ARINC Specification 653-2, Part 1 (Required Services). ARINC, March 2006.

[4] Airlines Electronic Engineering Committee (AEEC), Avionics application software standard interface, ARINC Specification 653, Part 2 (Extended Services), Draft 5. ARINC, August 2006.

[5] J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms and assurance. NASA Contractor Report CR-1999-209347, SRI International, California, USA, June 1999.

[6] J-L. Terraillon and K. Hjortnaes. Technical note on on-board software. European Space Technology Harmonisation, Technical Dossier on Mapping, TOSE-2-DOS-1, ESA, February 2003.

[7] N. Diniz and J. Rufino. ARINC 653 in space. In *Proceedings of the DASIA 2005 "DAta Systems In*

Aerospace" Conference, Edinburgh, Scotland, June 2005. EUROSPACE.

[8] K. Hjortnaes. Time and space partitioning for space application: Working group recommendations. In *ESA Workshop on Avionics Data, Control and Software Systems (ADCSS)*, Noordwijk, The Netherlands, October 2008.

[9] J. Rufino, S. Filipe, M. Coutinho, S. Santos, and J. Windsor. ARINC 653 interface in RTEMS. In *Proceedings of the DASIA 2007 "DAta Systems In Aerospace" Conference*, Naples, Italy, June 2007. EUROSPACE.

[10] J. Craveiro, J. Rufino, C. Almeida, R. Covelo, and P. Venda. Embedded Linux in a partitioned architecture for aerospace applications. In *Proceedings of the 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2009)*, pages 132–138, Rabat, Morocco, May 2009.

[11] J. E. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann Publishers, 2005.

[12] S. Santos, J. Rufino, T. Schoofs, C. Tatibana, and J. Windsor. A portable ARINC 653 standard interface. In *Proceedings of the IEEE/AIAA 27th Digital Avionics Systems Conference (DASC '08)*, St. Paul, Minnesota, USA, October 2008.

[13] IEEE. *1996 (ISO/IEC) [IEEE/ANSI Std 1003.1, 1996 Edition] Information Technology — Portable Operating System Interface (POSIX®) — Part 1: System Application: Program Interface (API) [C Language]*. IEEE, New York, NY, USA, 1996.

[14] M. Schoen. Virtual spacecraft system architecture. Technical Note TOS-EMS-VSRF-TN-002, ESA Modelling and Simulation (EMS).