A portable ARINC 653 standard interface

Conference Paper · November 2008

DOI: 10.1109/DASC.2008.4702767 · Source: IEEE Xplore

A PORTABLE ARINC 653 STANDARD INTERFACE

Sérgio Santos, Skysoft Portugal – Software e Tecnologias de Informação SA, Lisboa, Portugal.
José Rufino[*]*, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal.*
Tobias Schoofs, Skysoft Portugal – Software e Tecnologias de Informação SA, Lisboa, Portugal.
Cássia Tatibana, Skysoft Portugal – Software e Tecnologias de Informação SA, Lisboa, Portugal.
James Windsor, ESA- European Space Agency, Noordwijk, The Netherlands.

Abstract

The ARINC 653 specification defines the functionality that an Operating System (OS) must guarantee to enforce robust spatial and temporal partitioning as well as an avionics application programming interface for the system.

The standard application interface – the ARINC 653 Application Executive (APEX) – is defined as a set of software services a compliant OS must provide to avionics application developers. The ARINC 653 specification defines the interfaces and the behavior of the APEX but leaves implementation details to OS vendors.

This paper describes an OS independent design approach of a Portable APEX interface. POSIX, as a programming interface available on a wide range of modern OS, will be used to implement the APEX layer. This way the standardization of the APEX is taken a step further: not only the definition of services is standardized but also its interface to the underlying OS. Therefore, the APEX operation does not depend on a particular OS but relies on a well defined set of standardized components.

Introduction

The ARINC 653 specification gives a clear definition of interfaces and behavior of the APEX. The implementation, on the other hand, is left to system suppliers. During the AMOBA project, developed by Skysoft Portugal and co-funded by the European Space Agency (ESA), the project team was confronted with the requirement of an APEX implementation that could be standardized as well.

AMOBA is a multi-platform ARINC 653 simulator. The main purpose of AMOBA is to provide to users an execution environment with the capability to execute and verify ARINC 653 applications [1]. The AMOBA Simulator aims to provide a low cost, yet effective, environment to develop space applications and verify their behaviour without having access to the final target platform and without the need for a real ARINC 653 Real-Time Operating System (RTOS).

A fundamental design attribute of AMOBA concerns the portability of the simulator. A multi-platform approach shall provide availability of the simulator on every POSIX-compliant operating system [2]. The AMOBA Simulator is intended to be executed on different platforms, so it must avoid dependency between APEX and the OS Kernel. This means that it shall not access directly the OS kernel specific interface unless it is strictly necessary.

During the project, a layered architecture was adopted. By means of this layered architecture, direct calls from the APEX to the underlying OS were drastically reduced and entirely encapsulated in a POSIX-dependent sub-component providing fundamental services to the APEX. The AMOBA Project team exploited the logical independence between the APEX and the underlying OS to enforce portability and flexibility properties.

This led to the advanced concept of the Portable APEX. The Portable APEX is currently further developed beyond the scope of the Simulator, to meet requirements of real ARINC 653 RTOS. The main goal of the Portable APEX is now to host real avionics applications on real partitioning systems.

The ARINC 653 Architecture

ARINC 653 defines support for robust partitioning in on-board systems, such that one processing unit, usually called a module, is able to host one or more avionics applications and to execute them independently [3,4]. This can be correctly achieved if the underlying system, often called the Module Operating System (MOS), provides separation of the avionics applications, usually for fault containment, such that a failure in one partitioned function cannot cause a failure in another partitioned function; in consequence the partitioning approach eases verification, validation and certification [3,5]. The unit of partitioning is called a partition. In a given sense, a partition is equivalent to a program in a single application environment: it comprises data, code and its own context configuration attributes (see Figure 1).

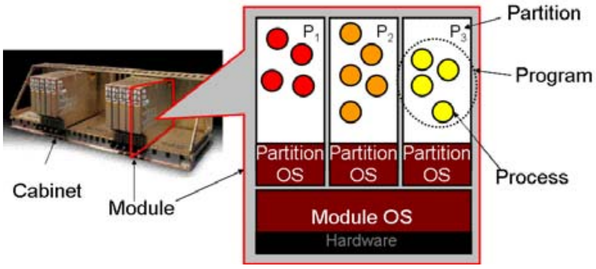

Figure 1: Space Partitioning

Partitioning separates applications in two dimensions: space and time. Spatial separation means that the memory of a partition is protected. No application can access memory out of the scope of its own partition. Temporal separation means that only one application at a time has access to system resources, including the processor; therefore only one application is executing at one point in time – there is no competition for system resources between partitioned applications.

ARINC 653 defines a static configuration where each partition is assigned a set of execution windows. The program in the partition associated with the current execution window gains access to the processor. When the execution window terminates, the program is preempted; when the next execution window starts, the program continues execution from the point it was previously preempted (see also Figure 2).

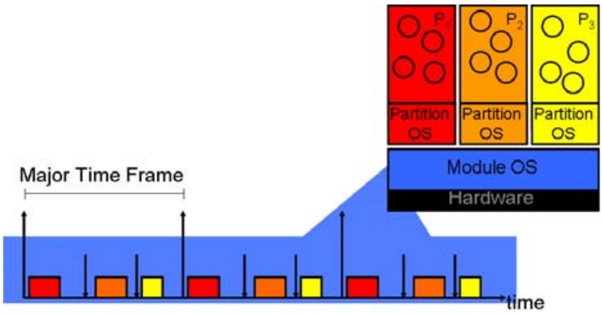

Figure 2: Time Partitioning

Processes within the scope of a partition are scheduled by a priority-based preemptive scheduler with first-in-first-out (FIFO) order for processes with the same priority. This second level scheduler is invoked whenever an execution window assigned to its partition starts and the partition gains access to the processor. The process scheduler is preempted by the first level partition scheduler when the execution window terminates.

The services defined by ARINC 653 are partition and process management services, partition-internal inter-process communication and synchronization means, like events, message buffers, blackboards and semaphores; inter-partition services via queuing and sampling ports; time services; an interface to the health monitor; and interfaces to the partition associated with the application and the processes it consists of. Additionally, optional services are provided by part 2 of the ARINC 653 specification, like file system services, shared memory blocks and naming services [6].

Portable APEX: Architectural Overview

As illustrated in Figure 1, the application within a partition is supported by an OS with partition scope, the Partition Operating System (POS). The POS provides functionalities like process management and scheduling, as well as inter-process communication (IPC) support for applications, and also interacts with the module operating system on behalf of the hosted application.

In commercial ARINC 653 compliant RTOS, like VxWorks 653 from Wind River or PikeOS by SysGo, the POS is usually an integrated part of the system [7,8,9]. But in principle, any kind of RTOS could be used to provide an API to the hosted avionics applications. In fact, the partitioning approach is often associated with virtualization concepts, where partitions are seen as virtual machines with their own para-virtualized operating system [8,9,10,18]. VxWorks 653 and PikeOS, for instance, use virtualization concepts to host general purpose operating systems based on Linux. Consequently, different operating systems can be used with different partitions forming different *personalities* [9] for applications that vary in requirements and APIs they rely on, like APEX or POSIX.

In the context of the AIR-II project (ARINC Interface in RTOS – Industrial Initiative) Skysoft, University of Lisbon and Thales Alenia Space sponsored by ESA are defining an ARINC 653 compliant RTOS that uses RTEMS as primary POS [11,12,13,14]. RTEMS (Real-Time Executive for Multiprocessor Systems) is an open source RTOS that provides three different APIs to application programmers: an RTEMS native API, a POSIX compliant API and an ITRON API [15,16]. But AIR is not limited to RTEMS. Other OS like eCos [17] or any embedded flavor of Linux or even general purpose Linux may be used as POS in the future, supporting this way a wide range of APIs and applications relying on them [18].

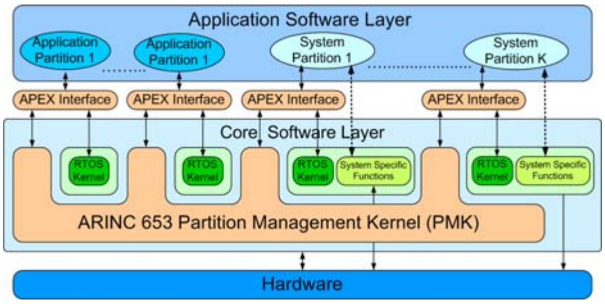

Figure 3: The AIR Architecture

Among the APIs available for AIR applications the most important is the ARINC 653 APEX interface. The AIR-II consortium decided not to implement an entirely new POS to provide APEX services to hosted applications but to build the APEX interface on top of available RTOS (see Figure 3).

The first approach was to target the RTEMS native API; the disadvantage of this solution is that, despite of having a wide range of RTOS available, only one would be able to host the APEX. Therefore, it was decided to adopt the AMOBA approach and to build the APEX interface using a standard API that is available on all relevant RTOS; this standard API is POSIX. With POSIX as baseline, the APEX inherits the main feature from this API: portability. Users of the AIR system can choose the POS that fits best their needs.

But the Portable APEX is also independent of any system in the sense it may be integrated with any ARINC 653 MOS, that i) hosts a POSIX compliant POS and ii) supports the interface the APEX uses to request services like inter-partition communication or setting and getting partition-related control variables.

This way competition on the implementation of the standard is taken a step further. With this approach, competition is possible not only on the implementation of the standard as a whole, but even on the implementation of components of an ARINC 653 system. The supplier of a given ARINC 653 MOS, such as the AIR Partition Management Kernel (PMK), may have a role similar to an IMA system integrator concerning the potentially different POS that host avionics applications [5].

The Portable APEX adopts the layered architecture from AMOBA to additionally strengthen portability (see Figure 4). The Portable APEX is divided into two layers: an APEX layer, implementing the APEX services themselves, and a Core layer that provides fundamental services to the APEX layer. The main purpose of the Core layer is to hide POSIX-dependent implementation details. APEX processes, for instance, are not directly based on POSIX Threads but on a thread concept that is provided by the Core layer. The Core threads actually use POSIX Threads but this is invisible to the APEX layer.

With this approach only Core depends directly on the POSIX API. The POSIX-dependent Core may be replaced by any other Core implementation, using another API – perhaps of yet another RTOS that does not support POSIX but ITRON, for instance. It is even possible to implement the Core layer itself as a standalone OS, e.g. for the support of new functionalities not available in traditional commercial-off-the-shelf OS.

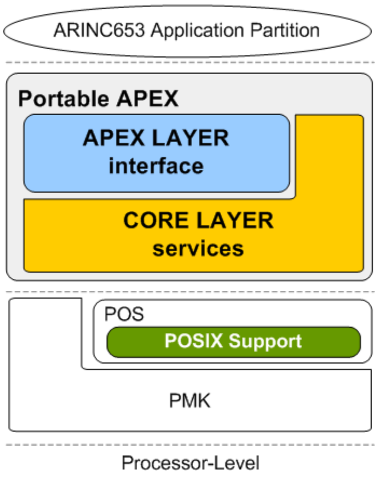

Figure 4: Portable APEX Architecture

Core Layer Abstractions

The Core layer is the basis for the Portable APEX implementation. Its main objective is to encapsulate the OS-, i.e. the POSIX-dependent parts of the APEX. It provides basic services to upper layers. The following abstractions are made available by the Core layer:

A **Thread Datatype** composed of a priority value and an entry point with the associated parameters. Besides thread creation, services provided include setting and getting a thread's priority.

The Core thread priority model consists of three states for user processes: RUNNING, NON-RUNNING or IDLE. These states are used by the scheduler to control process execution. Additionally, there is a set of priorities for internal use, for the time manager and the process scheduler for instance. [19,20]

Mutex Mechanisms offering the basic locking and unlocking primitives; these internal locking primitives use protocols for priority inversion avoidance [25].

A **Time Representation** is provided to the above APEX layer, making it independent of a specific time notion. Services made available include time conversion between APEX layer time (as described in the standard [3]) and the abstract Core layer.

A **Signal Model** composed of signalling and signal waiting abstractions that facilitate communication between the Portable APEX system threads.

A **PMK Interface**, taken the AIR PMK as a significant representative of an ARINC 653 MOS, composed of a set of functions that service the APEX layer modules that directly interact with the MOS (e.g. the APEX layer Partition Management).

These Core functional components are intended to provide the services required for the APEX implementation. They hide the platform specifics from the APEX level enabling the APEX portability even to non-POSIX systems without any changes. The necessary changes are limited to the Core layer.

APEX Layer Services

Main objective of the APEX layer is to provide the applications with the set of primitives defined in the ARINC 653 standard specification and supply compatibility/portability to APEX based applications. This layer relies only on the Core layer. The APEX can be ported to another system without any changes. All platform-specific functionality is hidden in the Core layer.

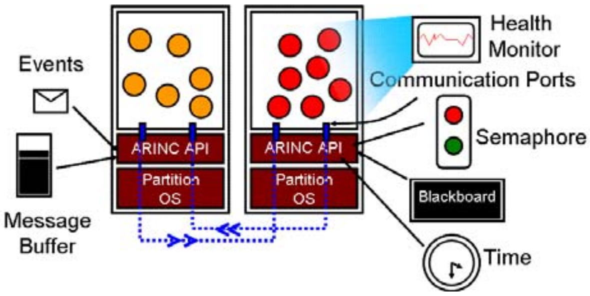

Figure 5: APEX Services

The APEX-Interface aims to provide services for (see Figure 5):

Partition management;

Process management;

Time management;

Intra-partition communication;

Inter-partition communication;

Health Monitoring.

A set of modules implements these services:

Partition Management Module: The standard services for partition management essentially provide means for modifying the operating mode of partitions. This module relies on the underlying Core layer PMK interface for partition schedule control and execution.

Process Management Module: The objective of this module is to implement the APEX process management and its core functionality, the process scheduler. The process scheduler must be safe and simple in terms of code, exhibiting low processing overheads and providing strict compliance with the ARINC 653 standard [3,6].

The adopted solution follows known scheduler design approaches [19], using the Core thread data type abstraction.

An ARINC 653 process can be in one of the four available states, these are as follows:

Dormant - process ineligible for scheduling;

Waiting - process is not able to execute;

Ready - process is able to be executed;

Running - process is currently executing.

The APEX processes are maintained in lists according to these states. The states are mapped to priorities of the Core thread data type: the running process has RUNNING priority, Ready, Waiting and Dormant processes have NON-RUNNING priority.

To avoid that a non-running process becomes eligible to the processor, an idle process with a middle priority is inserted. Its priority is lower than the running process priority but higher than the non-running process priority. Unless the scheduler explicitly selects a process to gain running priority it has no hypothesis to run [19,20].

Time Manager Module: The implementation of time management addressed ARINC 653 standard definition [3]: *Time is unique and independent of partition execution within a core module. All values or capacities are related to this unique time and are not relative to any partition execution.*

The time manager is an essential module of the APEX layer, in the sense that it shall provide all the fundamental time related support for APEX services. The main services provided by the Time Manager are:

- A function to retrieve the current system time; the APEX service `GET_TIME` is directly mapped to this routine.
- A wait and time-out mechanism. The APEX requires the capability to schedule an event in the future while the process proceeds execution. The occurrence of the event is verified by the time manager and must cause an action like raising an error or cause the change in the process state.
- A replenish service to increase the time budget of a process with a hard real-time deadline.

The time manager provides interfaces to register and cancel events, like wait intervals and time-outs. Events are scheduled by the Time Manager according to the time of their occurrence: events related to time points closer to the current time are placed in the head of the event list. The Time Manager waits until the first event of its schedule occurs.

Together with the Process Management services, the Time Manager services handling time-outs and wait intervals guarantee that processes are set to the appropriate state timely. This interaction of process and time management contributes to guarantee the real-time characteristics of the APEX and the applications running on top of it [21]. In this context, another important contribution of the AIR project concerns the use of traditional and improved process schedulability analysis for ARINC 653 based systems, thus securing the provisioning of real-time guarantees [26].

Intra-partition communication:

Intra-partition communication is used for communication and synchronization of processes within the same partition. Processes interoperate via Message Buffers, Blackboards, Events and Semaphores [3].

Processes may block for an optionally specified period of time when accessing a given intra-partition communication resource that is currently not available. Each resource manages its associated list of blocked processes, keeping them ordered either by priority or by time of arrival (First-In-First-Out). A process leaves the list of blocked processes if the resource becomes available to it or the specified time-out expires.

Inter-partition communication Interfaces:

This module implements the communication means between processes in different partitions. All inter-partition communication is conducted via messages. When sent by user applications, messages are copied into local buffers. The underlying PMK is then responsible for actually transferring a given message to the corresponding buffer in the target partition, providing the physical means to activate message exchange between the source process and the outer destination.

Conformance to the Standard

The conformance of the Portable APEX to the ARINC 653 standard is currently validated by means of the ARINC 653 Verification Test Suite (AVT) [22]. AVT is a verification tool, developed by Skysoft Portugal, S.A., that implements part 3 of the ARINC 653 standard, the Conformity Test Specification [23]. Passing this conformity test makes the Portable APEX, after SysGo's PikeOS and Wind River's VxWorks 653 2.1 and 2.2, the fourth system able to prove its compliance to the ARINC 653 standard.

Conclusions

The Portable APEX is a contribution to the idea of Open Standards. The ARINC 653 specification gives a clear definition of interfaces and behavior of the APEX. The POSIX standard on the other hand provides means, available on most modern commercial and open source Operating Systems, to implement these interfaces and their visible behavior.

The fact that the Portable APEX is based on POSIX interfaces does not imply that the implementation is just a simple mapping of ARINC 653-defined functionality on the POSIX counterparts. There is a need to control resources for recovery actions, for error detection and for further debugging and monitoring purposes. This gives room for implementations to compete.

In the Automotive Industry a standard for the integration of components by different vendors was defined. This standard is called AUTOSAR (AUTomotive Open System ARchitecture) [24]. The credo of the AUTOSAR initiative may be adopted for the given context; the goal is to "cooperate on standards and to compete on implementations". The same idea should be applied to the Portable APEX concept.

References

[1] E. Pascoal, J. Rufino, T. Schoofs, and J. Windsor, May 2008, AMOBA - ARINC 653 Simulator for Modular Space Based Applications. *DAta Systems In Aerospace. Conference*, Palma de Mallorca, Spain, EUROSPACE.

[2] IEEE Std 1003.1 - standard for information technology portable operating system interface (POSIX) system interfaces, 2004.

[3] Airlines electronic engineering committee (AEEC), avionics application software standard interface - ARINC specification 653 - part 1 (supplement 2 - required services), 2006, ARINC Inc.

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

[5] Airlines electronic engineering committee (AEEC), design guidance for integrated modular avionics - ARINC specification 651. 1991, ARINC Inc.

[6] Airlines electronic engineering committee (AEEC), avionics application software standard interface - ARINC specification 653 - part 2 (extended services), June 2007, ARINC, Inc.

[7] Wind River, 2003, *VxWorks Programmers Guide-5.5*, 2 edition, Alameda, CA, USA.

[8] R. Kaiser, 2007, Combining Partitioning and Virtualization for Safety-Critical Systems. PikeOS White Paper, SysGo. Klein-Winterheim.

[9] R. Kaiser, S. Wagner, 2007, The PikeOS Concepts. History and Design,. PikeOS White Paper, SysGo. Klein-Winterheim.

[10] G. Heiser, 2007, Virtualization for Embedded Systems. Technology White Paper, Open Kernel Labs Inc. Sydney.

[11] N. Diniz and J. Rufino. June 2005. ARINC 653 in space. In *Proceedings of the DASIA 2005 .DAta Systems In Aerospace. Conference,* Edinburgh, Scotland,. EUROSPACE.

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

[13] J. Rufino and S. Filipe, December 2007, AIR Project Final report. Technical Report TR 07-35, Faculdade de Ciências da Universidade de Lisboa, Departamento de Informática (FCUL/DI), Lisboa, Portugal.

[14] J. Rufino and J. Craveiro. July 2008. Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems. 1st INTERAC Plenary Workshop, Braga, Portugal.

[15] OAR - On-Line Applications Research Corporation. February 2008. *RTEMS C Users Guide*, Edition 4.8, for RTEMS 4.8 edition.

[16] OAR - On-Line Applications Research Corporation. February 2008. *RTEMS POSIX API Users Guide*, Edition 4.8, for RTEMS 4.8 edition.

[17] A. Massa, 2002. *Embedded Software Development with eCos*. Prentice-Hall. ISBN 0130354732.

[18] J. Smith and R. Nair. June 2005. *Virtual Machines: versatile platforms for systems and processes*. Morgan Kaufmann ISBN 1558609105.

[19] L. Peng, B. Ravindran, S. Suhaib, and S. Feizabadi. September 2004. Formally verified application-level framework for real-time scheduling on POSIX real-time operating systems. IEEE Transactions on Software Engineering, 30(9).

[20] K. Obenland. September 2000.The use of POSIX in real-time systems, assessing its effectiveness and performance. The MITRE Corporation.

[21] Digital. March 1996. *Digital Unix - Guide to Realtime Programming*. Digital Equipment Corporation.

[22] Skysoft Portugal S.A., April 2007. *ARINC 653-1 API/OS Verification Tool User Manual*, version 1.2 edition, Portugal.

[23] Airlines electronic engineering committee (AEEC), avionics application software standard interface - ARINC specification 653 - part 3 (conformity test specification). 2006, ARINC, Inc.

[24] H. Fennel, H. Heinecke, K. Schnelle. 2006. *Achievements and exploitation of the AUTOSAR development partnership*. AUTOSAR Partnership.

[25] L. Sha, R. Rajkumar and J. Lehoczky. September 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, (39(9). IEEE.

[25] M. Coutinho, J. Rufino, and C. Almeida. July 2008. Response time analysis of asynchronous periodic and sporadic tasks scheduled by a _xed-priority preemptive algorithm. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS2008)*, Prague, Czech Republic. IEEE.

Email Addresses and Web Site

Sérgio Santos: sergio.santos@skysoft.pt

José Rufino: ruf@di.fc.ul.pt

Tobias Schoofs: tobias.schoofs@skysoft.pt

Cássia Tatibana: cassia.tatibana@skysoft.pt

James Windsor: James.Windsor@esa.int

Faculdade de Ciências, Universidade de Lisboa (University of Lisbon): Project AIR Website, http://air.di.fc.ul.pt.

27th Digital Avionics Systems Conference
October 26-30, 2008

* This work was partially supported by FCT through the Multiannual Funding Programme. This work was partially supported by ESA (European Space Agency) through the ITI program, ESTEC Contract 21217/07/NL/CB , Project AIR-II, http://air.di.fc.ul.pt.